Syntax Role for Neural Semantic Role Labeling

Zuchao Li
Shanghai Jiao Tong University
Department of Computer Science and Engineering
charlee@sjtu.edu.cn

Hai Zhao*
Shanghai Jiao Tong University
Department of Computer Science and Engineering
zhaohai@cs.sjtu.edu.cn

Shexia He
Shanghai Jiao Tong University
Department of Computer Science and Engineering
heshexia@sjtu.edu.cn

Jiaxun Cai
Shanghai Jiao Tong University
Department of Computer Science and Engineering
caijiaxun@sjtu.edu.cn

Semantic role labeling (SRL) is dedicated to recognizing the semantic predicate-argument structure of a sentence. Previous studies in terms of traditional models have shown syntactic information can make remarkable contributions to SRL performance; however, the necessity of syntactic information was challenged by a few recent neural SRL studies that demonstrate impressive performance without syntactic backbones and suggest that syntax information becomes much less important for neural semantic role labeling, especially when paired with recent deep neural network and large-scale pre-trained language models. Despite this notion, the neural SRL field still lacks a systematic and full investigation on the relevance of syntactic information in SRL, for

* Corresponding author. This work was supported by the National Key Research and Development Program of China (No. 2017YFB0304100) and the Key Projects of National Natural Science Foundation of China (U1836222 and 61733011).

Submission received: 11 September 2020; revised version received: 11 February 2021; accepted for publication: 5 June 2021.

https://doi.org/10.1162/COLI_a_00408

© 2021 Association for Computational Linguistics
Published under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International (CC BY-NC-ND 4.0) license
both dependency and both monolingual and multilingual settings. This paper intends to quantify the importance of syntactic information for neural SRL in the deep learning framework. We introduce three typical SRL frameworks (baselines), sequence-based, tree-based, and graph-based, which are accompanied by two categories of exploiting syntactic information: syntax pruning-based and syntax feature-based. Experiments are conducted on the CoNLL-2005, -2009, and -2012 benchmarks for all languages available, and results show that neural SRL models can still benefit from syntactic information under certain conditions. Furthermore, we show the quantitative significance of syntax to neural SRL models together with a thorough empirical survey using existing models.

1. Introduction

Semantic role labeling (SRL), namely, semantic parsing, is a shallow semantic parsing task that aims to recognize the predicate-argument structure of each predicate in a sentence, such as who did what to whom, where and when, and so forth. Specifically, SRL seeks to identify arguments and label their semantic roles given a predicate. SRL is an important method for obtaining semantic information that is beneficial to a wide range of natural language processing (NLP) tasks, including machine translation (Shi et al. 2016), question answering (Berant et al. 2013; Yih et al. 2016), and discourse relation sense classification (Mihaylov and Frank 2016) and relation extraction (Lin, Liu, and Sun 2017).

SRL can be split into four subtasks: predicate detection, predicate disambiguation, argument identification, and argument classification. For argument annotation, there are two formulations (styles). One is based on constituents (i.e., phrase or span), latter the other is based on dependencies. The other, proposed by the CoNLL-2008 shared task (Surdeanu et al. 2008), is also called semantic dependency parsing and annotates the heads of arguments rather than phrasal arguments. Figure 1 shows example annotations.

In prior SRL work, considerable attention has been paid to feature engineering, which struggles to capture sufficient discriminative information compared to neural network models, which are capable of extracting features automatically. In particular, syntactic information, including syntactic tree features, has been known to be extremely beneficial to SRL since the large scale of empirical verification of Punyakanok, Roth, and Yih (2008). Despite their success, their work suffered from erroneous syntactic input, leading to an unsatisfactory performance.

To alleviate these issues, Marcheggiani, Frolov, and Titov (2017) and He et al. (2017) proposed a simple but effective neural model for SRL without syntactic input. Their work suggested that neural SRL does not have to rely on syntactic features, contradicting the belief that syntax is a necessary prerequisite for SRL, which was believed as early as Gildea and Palmer (2002). This dramatic contradiction motivated us to make a thorough exploration on syntactic contribution to SRL.

![Figure 1](https://i.imgur.com/5Q3Q5Q5.png)

Figure 1
Examples of annotations in span (above) and dependency (below) SRL.
Both span and dependency are effective formal representations for semantics, though it has been unknown which form, span or dependency, would be better for the convenience and effectiveness of semantic machine learning and later applications for a long time. This topic has been roughly discussed in Johansson and Nugues (2008a) and Li et al. (2019a), who both concluded that the (best) dependency SRL system at that time clearly outperformed the span-based (best) system through gold syntactic structure transformation; however, due to the different requirements of downstream task applications, span and dependency both remain focuses of research. Additionally, the two forms of SRL may benefit from each other’s joint (rather than separated) development. We, therefore, revisit role of syntax in SRL on a more solid empirical basis and investigate the role of syntax for the two SRL styles by supplying syntax knowledge of varying quality.

Recent work on syntax contributions has been limited to individual models and the ways in which syntax has been utilized. The conclusions drawn for syntax roles therefore have some limitations. In order to reduce these limitations, we explored three typical and strong baseline models and two categories of syntactic utilization methods. In addition, pre-trained language models, such as ELMo (Peters et al. 2018) and BERT (Devlin et al. 2019), that build contextualized representations, continue to provide gains on NLP benchmarks, and Hewitt and Manning (2019) showed that structure of syntax information emerges in the deep models’ word representation spaces. Whether neural SRL models can further benefit from explicit syntax information in addition to this implicit syntax information, however, is another issue we consider.

Besides, most of the SRL literature is dedicated to impressive performance gains on English, while other multiple languages receive relatively little attention. Although human languages have some basic commonalities in syntactic structure and even different levels of grammar, their differences are also very obvious. The study of syntactic roles needs to be examined in the context of multiple languages for verifying its effectiveness and applicability.

In order to quantitatively evaluate the contribution of syntax to SRL, we adopt the ratios between labeled F₁ score for semantic dependencies (Sem-F₁) the labeled attachment score (LAS) for syntactic dependencies, and the F₁ score for syntactic constituents. This ratio was first introduced by CoNLL-2008 (Surdeanu et al. 2008) Shared Task as an evaluation metric. Because different syntactic parsers contribute different syntactic inputs with varying levels of quality, different syntactically driven SRL systems are based on different syntactic foundations. Therefore, our proposed ratio offers a fairer comparison between different syntactically driven SRL systems, which our empirical study surveys.

2. Background

SRL was first pioneered by Gildea and Jurafsky (2000) based on the FrameNet semantic labeling project (Baker, Fillmore, and Lowe 1998). PropBank (Palmer, Gildea, and Kingsbury 2005) is one of the most commonly used labeling schemes for this task. This involves two variants: span-based labeling (span SRL), where arguments are characterized as word spans (Carreras and Màrquez 2005; Pradhan et al. 2012), and head-based

---

1 It is worth noting that the syntax studied in this paper is limited to syntactic knowledge in the narrow sense by using syntactic relationships (via dependency and constituency trees) but does not include lemma, part-of-speech, etc.
labeling (dependency SRL), which only labels head words and relies on syntactic parse trees (Hajič et al. 2009).

Conventionally, when identifying predicates, span SRL decomposes to two sub-tasks: argument identification and argument classification. The former identifies the arguments of a predicate, and the latter assigns them semantic role labels, determining the relations between arguments and predicates. PropBank defines a set of semantic roles for labeling arguments. These roles fall into two categories: core and non-core roles. The core roles (A0–A5 and AA) indicate different semantics in predicate-argument structure, while the non-core roles are modifiers (AM-adj), where adj specifies the adjunct type, such as in temporal (AM-TMP) and locative (AM-LOC) adjuncts. For the example shown in Figure 1, A0 is a proto-agent, representing the borrower.

Slightly different from span SRL in argument annotation, dependency SRL labels the head words of arguments rather than of entire phrases, a practice popularized by the CoNLL-2008 and CoNLL-2009 shared tasks (Surdeanu et al. 2008; Hajič et al. 2009). Furthermore, when no predicate is given, two other indispensable subtasks of dependency SRL are required: predicate identification and predicate disambiguation. The former identifies all predicates in a sentence, and the latter determines the word senses, the specific contextual meanings, of predicates. In the example shown in Figure 1, 01 indicates the first sense from the PropBank sense repository for predicate borrowed in the sentence.

Johansson and Nugues (2008c) demonstrated that in conventional SRL models, syntactic trees provide a good form of representation for the assigning of semantic role labels. The successful application of neural networks to SRL (Zhou and Xu 2015; He et al. 2017; Marcheggiani, Frolov, and Titov 2017; Cai et al. 2018) mitigated conventional SRL models’ need for comprehensive feature engineering based on syntax trees (Zhao et al. 2009a) and resulted in syntax-agnostic neural SRL models that achieved competitive performance. Recent work has built on this and explored the inclusion of syntax in neural SRL. Including syntax in SRL has three main benefits that have been common motivations for recent work:

• Arguments are often dispersed around the predicates in syntax trees (Xue and Palmer 2004; Zhao and Kit 2008; He et al. 2018b; He, Li, and Zhao 2019).

• Some predicate-argument arcs in semantic dependency graphs are mirrored by head-dependent arcs in their corresponding dependency parse trees, and there is a deterministic mapping between these syntactic relationships and semantic role labels (Surdeanu et al. 2008; Lang and Lapata 2010; Marcheggiani and Titov 2017; Li et al. 2018; Cai and Lapata 2019b; Marcheggiani and Titov 2020).

• Syntax parse trees can strengthen language representations (Johansson and Nugues 2008c; Strubell et al. 2018; Kasai et al. 2019).

2 The head word for a span refers to the word serves as a dependency relation’s modifier for words outside the span and a dependency head for words inside the span. This is different from syntactic heads in head-dependent relationships.

3 CoNLL-2008 is an English-only task, while CoNLL-2009 extends to a multilingual one. Their main difference is that predicates have been beforehand indicated for the latter. Or rather, CoNLL-2009 does not need predicate identification, but it is an indispensable subtask for CoNLL-2008.
In this paper, since the third benefit is a general improvement for downstream tasks and not limited to SRL, we explore the exploitation of the first two benefits for use in neural SRL.

3. Methodology

To fully disclose the predicate-argument structure, typical SRL systems have to perform four subtasks step-by-step or jointly learn and predict the four targets. In order to research the role of syntax, we evaluate our systems in two separate settings: being given the predicate and not being given the predicate. For the first setting, our backbone models all only focus on the identification and labeling of arguments. We use the pre-identified predicate information when the predicate is provided in the corpus and adopt a sequence tagging model to perform predicate disambiguation. In the second condition, we do the work of predicate identification and disambiguation in one sequence tagging model. In summary, we focus on three backbone models for argument identification and disambiguation and feed the predicates into the models as features.

3.1 Factorization and Modeling

We summarize and present three typical baseline models, which are based on the strategies of factoring and modeling of semantic graphs in the SRL: Sequence-based, Tree-based, and Graph-based.

**Formalization.** Given a sequence of tokens \( X = (w_1, w_2, \ldots, w_n) \), a span SRL graph can be defined as a collection of labeled predicate-argument pairs over these tokens: \( S = \{(p, (i, j), r^s), 1 \leq p \leq n, 1 \leq i \leq j \leq n, r^s \in \mathcal{R}^s\} \), where \( S \) represents a labeled predicate-argument pair for predicate \( p \) and the argument span located between sentence fencepost positions \( i \) and \( j \) and with label \( r^s \). A dependency SRL semantic graph for the sentence can be defined as \( D = \{(p, a, r^d), 1 \leq p \leq n, 1 \leq d \leq n, r^d \in \mathcal{R}^d\} \), where \((p, a, r^d)\) consists of a predicate \( (x_p) \), an argument \( (x_a) \), and the type of the semantic role \( r^d \), which is in label set \( \mathcal{R}^d \).
Sequence-based. As shown in Figure 2, the semantic dependency graph of SRL is decomposed by predicates. The arguments for each predicate consist of a sequence of either dependency-style or span-style. Notably, an extra Begin-Inside-Outside (BIO) conversion step is required for span-style argument labels. This decomposition is very simple and efficient. In the baseline model of this factorization, the predicate needs to be input as a source feature, which allows the model to produce different inputs for different target argument sequences. Predicate-specific embeddings are usually used for this reason. In our previous work (He et al. 2018b; Li et al. 2018; Munir, Zhao, and Li 2021), we presented models that recognized and classified arguments as in a sequence labeling task. The predicate-argument pairs were then constructed by performing multiple rounds of sequence labeling according to the number of predicates to obtain a final semantic graph.

In these models, the identification and classification of predicates and the recognition and classification of arguments in the sequence-based modeling are separated into two processes. Formally, the model first identifies the predicate (if not given) and obtains predicate set $P = \{p_1, p_2, \ldots, p_m\}$. Then, for each $p_i \in P$ in the predicate set, a sequence labeling model is adopted to predict the argument label of each token:

$$r_1, r_2, \ldots, r_n = \arg\max_{r \in R}(P(r|w_1, w_2, \ldots, w_n; p_i; \theta)),$$

where $\theta$ represent the model parameters, $r$ and $R$ represent $r^s$ and $R^s$ in span SRL or $r^d$ and $R^d$ in dependency SRL, and empty label $\phi$ ($\phi = \text{null}$ in dependency SRL, $\phi = O$ in span SRL) is used to indicate non-arguments. In dependency SRL, $a$ and $r^d$ are obtained after removing the empty labels, while in span SRL, after removing the empty labels, the BIO-converted label should be decoded to get the start and end positions $i$ and $j$ and the span’s role label $r^s$.

Tree-based. Embedding differentiation by relying on the predicate-indicating inputs is only a soft constraint and prompt. This feature may be lost due to forgetting mechanisms such as dropout in the encoder, which potentially limit SRL model performance. For further help in predicate clue integrating, the tree-based method also decomposes the semantic dependency graph to trees with a depth of 2 according to the predicate which is the child node of ROOT; all other nodes are child nodes of the predicate, as

Figure 2
An example of sequence-based factorization.
shown in Figure 3. An empty relation $\phi = \text{null}$ is set between the non-arguments and the predicate in order to fill the tree. The tree-based factorization can be thought of as an enhanced version of the sequence-based factorization, as the predicate is more prominent and obvious to specify. Also to emphasize a given predicate being handled, predicate-specific embeddings are applied. In our previous work (Cai et al. 2018), the predicate identification and classification and the recognition and classification of arguments are still viewed as two separate processes. Predicate-argument pairs for each identified predicate were scored using the following equation, which follows dependency parsers’ head-dependent scoring model rather than scoring the likelihood of a position being an argument:

$$r_1, r_2, \ldots, r_n = \arg \max_{r \in R}(P(r|\{w_1, w_2, \ldots, w_n\} \otimes p_i; \theta))$$

In tree-based modeling, progressive decoding is performed to output all possible arguments for each predicate.

**Graph-based.** Sequence-based and tree-based models score the $\langle\text{argument}, \text{label}\rangle$ tuple structure for a determined predicate. The graph-based method further extends this
mode; specifically it accommodates undetermined predicates and models the semantic dependency graph directly to output a \( \langle \text{predicate}, \text{argument}, \text{label} \rangle \) triple structure, allowing the model handle to label multiple predicates and arguments at the same time. This mode not only handles instances without given predicates but also allows instances with given predicates to be enhanced by predicate-specific embeddings. Using dependency-style, the graph-based method is a trivial extension of the tree-based method. Span-style is not so simple because of its argument structure. To account for this, graph-based models enumerate and sort all possible spans, takes them as candidate arguments, and scores them with the candidate sets of predicates. In our previous work (Li et al. 2019a, 2020), we considered the predicates and arguments jointly and explicitly scored the predicate-argument pairs before classifying their relationships. This modeling objective can be represented as:

\[
\{(p, a, r)\} = \arg \max_{p \in \mathcal{P}, a \in \mathcal{A}, r \in \mathcal{R}} \left( \mathcal{P}(p, a, r | w_1, w_2, \ldots, w_n; \theta) \right)
\]

where \( \mathcal{P} = \{w_1, w_2, \ldots, w_n\} \) is the set of all predicate candidates, which is used in graph-based modeling instead of relying on the predictions of an additional predicate identification model. In dependency SRL, the argument candidates set \( \mathcal{A} \) consists of all words in the sentence, \( \mathcal{A} = \{w_1, w_2, \ldots, w_n\} \), and in span SRL, \( \mathcal{A} \) consists of all possible spans, \( \mathcal{A} = \{(w_i, w_j), 1 \leq i \leq j \leq n\} \).

The above methods cover most mainstream neural SRL models based on semantic dependency graph modeling to the best of our knowledge. There are some modeling approaches, such as transition-based SRL (Choi and Palmer 2011; Fei et al. 2021), that are not based on semantic dependency graphs and hence not the focus of this paper. In the sequence-based and tree-based methods, the BIO conversion is adopted when using span-style, and some works use Conditional Random Fields (CRFs) to model this constraint.

### 3.2 Baseline Implementation

This subsection presents basic neural SRL models under the three previous aforementioned methods. In order to make fair comparisons with our experiments, we make the architectures of these models as similar as possible.

**Word Representation.** We produce a predicate-specific word representation \( e_i \) for each word \( w_i \) in the sequence \( w = \{w_1, \ldots, w_n\} \), where \( i \) stands for the word position in an input sequence, and \( n \) is the length of this sequence, following Marcheggiani, Frolov, and Titov (2017). In this work, word representation \( e_i \) is the concatenation of four types of features: a predicate-specific feature and character-level, word-level and linguistic features. Since previous works demonstrated that the predicate-specific feature is helpful in promoting the role labeling process, we leverage a predicate-specific indicator embedding \( e_i^{\text{pred}} \) to indicate whether a word is a predicate when predicting and labeling the arguments for each given predicate. At the character level, we exploit a convolutional neural network (CNN) with a bidirectional LSTM (BiLSTM) to learn character embedding \( e_i^{\text{ce}} \). As shown in Figure 5, the representation calculated by the CNN is fed as input to the BiLSTM. At the word level, we use a randomly initialized word embedding \( e_i^{\text{we}} \) and a pre-trained word embedding \( e_i^{\text{pe}} \). For linguistic features, we employ a randomly initialized lemma embedding \( e_i^{\text{le}} \) and a randomly initialized POS tag embedding \( e_i^{\text{pos}} \). To further enhance the word representation, we leverage an
optimal external representation $e_{i}^{plm}$ from pre-trained language models. The resulting word representation is concatenated as $e_i = [e_i^{ie}, e_i^{re}, e_i^{pe}, e_i^{ce}, e_i^{ie}, e_i^{re}, e_i^{pe}, e_i^{ce}, e_i^{pos}, e_i^{plm}]$.

Sequence Encoder. As Long Short-Term Memory (LSTM) networks (Hochreiter and Schmidhuber 1997), specifically BiLSTMs, have shown significant representational effectiveness for NLP tasks (Sutskever, Vinyals, and Le 2014; Vinyals et al. 2015), we thus use a BiLSTM as the sentence encoder. Given a sequence of word representations $x = \{e_1, e_2, \cdots, e_n\}$ as input, the $i$-th hidden state $g_i$ is encoded as follows:

$$g_i^f = \text{LSTM}^F(e_i, g_{i-1}^f), \quad g_i^b = \text{LSTM}^B(e_i, g_{i+1}^b), \quad g_i = g_i^f \oplus g_i^b$$

where $\text{LSTM}^F$ denotes the forward LSTM transformation and $\text{LSTM}^B$ denotes the backward LSTM transformation. $g_i^f$ and $g_i^b$ are the hidden state vectors of the forward LSTM and backward LSTM, respectively. Specifically, we initialize hidden states $g_0$ and $g_{n+1}$ as zero tensors.

Scorer in the Sequence-based Model. In the sequence-based model, namely, the sequence tagging model, to get the final predicted semantic roles, stacked multilayer perceptron (MLP) layers on the top of BiLSTM networks are usually exploited, which take as input
the hidden representation $h_i$ of all time steps and employ ReLU activations between the hidden layers. Finally, A softmax layer is used over the outputs to maximize the likelihood of labels.

Scorer in the Tree-based Model. As in the sequence-based model, to predict and label arguments for a given predicate, a role classifier is employed on top of the BiLSTM encoder. Some work like Marcheggiani, Frolov, and Titov (2017) shows that incorporating the predicate’s hidden state in their role classifier enhances the model performance, while we argue that a more natural way to incorporate the syntactic information carried by the predicate is to use the attentional mechanism. We adopt the recently introduced biaffine attention (Dozat and Manning 2017) to enhance our role scorer. Biaffine attention is a natural extension of bilinear attention (Luong, Pham, and Manning 2015), which is widely used in neural machine translation (NMT).

Nonlinear Affine Transformation. Usually, a BiLSTM decoder takes the concatenation $g_i$ of the hidden state vectors as output for each hidden state; however, in the SRL context, the encoder is supposed to distinguish the currently considered predicate from its candidate arguments. As noted in Dozat and Manning (2017), applying a MLP to the recurrent output states before the classifier has the advantage of stripping away irrelevant information for the current decision. Therefore, to distinguish the currently considered predicate from its candidate arguments in an SRL context, we perform two distinct affine transformations with a nonlinear activation on the hidden state $g_i$, mapping it to vectors with smaller dimensionality:

$$h_i^{(pred)} = \text{ReLU}(W^{(pred)}g_i + b^{(pred)})$$
$$h_i^{(arg)} = \text{ReLU}(W^{(arg)}g_i + b^{(arg)})$$

where ReLU is the rectilinear activation function (Nair and Hinton 2010), and $h_i^{(pred)}$ is the hidden representation for the predicate and $h_i^{(arg)}$ is the hidden representation for the candidate arguments.

By performing such transformations over the encoder output to feed the scorer, the scorer may benefit from deeper feature extraction. This leads to two benefits. First, instead of keeping both features learned by the two distinct LSTMs, the scorer ideally is now able to learn features composed from both recurrent states with reduced dimensionality. Second, it provides the ability to map the predicates and the arguments into two distinct vector spaces, which is essential for our tasks, since some words can be labeled as predicates and arguments simultaneously. Mapping a word into two different vectors can help the model disambiguate its role in different contexts.

Biaffine Scoring In the standard NMT context, given a target recurrent output vector $h_i^{(t)}$ and a source recurrent output vector $h_j^{(s)}$, a bilinear transformation calculates a score $s_{ij}$ for the alignment:

$$s_{ij} = h_i^{(t)\top}W_{ij}^{(s)}$$

However, in a traditional classification task, the distribution of classes is often uneven, and the output layer of the model normally includes a bias term designed to capture the prior probability $P(y_i = c)$ of each class, with the rest of the model focusing on learning the likelihood of each class given the data $P(y_i = c|\mathbf{x}_i)$. Dozat and Manning
(2017) incorporated the bias terms into the bilinear attention to address this uneven problem, resulting in a biaffine transformation, a natural extension of the bilinear transformation and the affine transformation. In the SRL task, the distribution of the role labels is similarly uneven, and the problem worsens after introducing the additional ROOT node and null label; directly applying the primitive form of bilinear attention would fail to capture the prior probability \( P(y_i = c_k) \) for each class. Thus, introducing the biaffine attention in our model would be extremely helpful for semantic role prediction.

It is worth noting that in our model, the scorer aims to assign a score for each specific semantic role. Besides learning the prior distribution for each label, we wish to further capture the preferences for the label that a specific predicate-argument pair can take. Thus, our biaffine attention contains two distinct bias terms:

\[
s_{i,j} = \text{Biaffine}(h_{i}^{\text{pred}}, h_{j}^{\text{arg}}) = h_i^{\text{arg}} W^{\text{role}} h_j^{\text{pred}} + U^{\text{role}} h_i^{\text{arg}} ⊕ h_j^{\text{pred}} + b^{\text{role}}
\]

where \( W^{\text{role}} \), \( U^{\text{role}} \), and \( b^{\text{role}} \) are parameters that will be updated by gradient descent methods in the learning process. There are several points that should be noted in the above biaffine transformation. First, because our goal is to predict the label for each pair of \( h_i^{\text{arg}} \) and \( h_j^{\text{pred}} \), the output of our biaffine transformation should be a vector of dimensionality \( N_r \) instead of a real value, where \( N_r \) is the number of all the candidate semantic labels. Thus, the bilinear transformation in Equation (1) maps two input vectors into another vector. This can be accomplished by setting \( W^{\text{role}} \) as a \((d_h \times N_r \times d_h)\) matrix, where \( d_h \) is the dimensionality of the hidden state vector. Similarly, the output of the linear transformation in Equation (2) is also a vector by setting \( U^{\text{role}} \) as a \((N_r \times 2d_h)\) matrix. Second, Equation (2) captures the preference of each role (or sense) label and is conditioned on taking the \( j \)-th word as a predicate and the \( i \)-th word as an argument. Third, the last term \( b^{\text{role}} \) captures the prior probability of each class \( P(y_i = c_k) \). Notice that Equation (2) and (3) capture different kinds of bias for the latent distribution of the label set.

Given a sentence of length \( n \), for one of its predicates \( w_j \), the scorer outputs a score vector \( \{s_{1,j}, s_{2,j}, \ldots, s_{n,j}\} \). Then, our model picks as its output the label with the highest score from each score vector: \( y_{ij} = \arg \max_{1 \leq k \leq N_r} (s_{ij}[k]) \), where \( s_{ij}[k] \) denotes the score of the \( k \)-th candidate in the semantic label vocabulary with size \( N_r \).

**Scorer in the Graph-based Model.** As in the scorer of the tree-based model, the graph-based model also uses the biaffine scorer to score the predicate-argument structure. Similarly, we also use a nonlinear affine transformation on the top of the BiLSTM encoder. In the sequence-based and tree-based models, dependency- and span-style arguments are converted into a consistent label sequence, while the graph-based model treats arguments as independent graph nodes. In order to unify the two styles of models, we introduce a unified argument representation that can handle both styles of SRL tasks.

In the sentence \( w_1, w_2, \ldots, w_n \), the model aims to predict a set of predicate-argument-relation tuples \( \mathcal{Y} \in \mathcal{P} \times \mathcal{A} \times \mathcal{R} \), where \( \mathcal{P} = \{w_1, w_2, \ldots, w_n\} \) is the set of all possible predicate tokens, \( \mathcal{A} = \{(w_i, \ldots, w_j) | 1 \leq i \leq j \leq n\} \) includes all the candidate...
argument spans or dependencies, and is the set of the semantic roles. For dependency SRL, we assume single word argument spans and thus limit the length of candidate arguments to be 1, so our model uses \( h^{arg} \) to construct the final argument representation \( h^{(arg)'} \) directly. For span SRL, we utilize the span representation from Lee et al. (2017). Each candidate span representation \( h^{(arg)'} \) is built by

\[
h^{(arg)'} = [h^{arg}_{START}, h^{arg}_{END}, h_\lambda, size(\lambda)]
\]

where \( h^{arg}_{START} \) and \( h^{arg}_{END} \) are boundary representations, \( \lambda \) indicates a span, \( size(\lambda) \) is a feature vector encoding the size of span, and \( h_\lambda \) is the specific notion of headedness learned by the attention mechanism (Bahdanau, Cho, and Bengio 2015) over words in each span (where \( t \) is the position inside span) as follows:

\[
\mu_t = w_{attn} \cdot MLP_{attn}(h^{(arg)}_t), \quad \nu_t = \frac{\exp(\mu_t)}{\sum_{k=START}^{END} \exp(\mu_k)}
\]

\[
h_\lambda = \sum_{t=START}^{END} \nu_t \cdot h^{(arg)}_t
\]

**Candidate Pruning** The number of candidate arguments for a sentence of length \( l \) is \( O(l^2) \) for span SRL and \( O(l) \) for dependency. As the model deals with \( O(l) \) possible predicates, the computational complexity is \( O(l^3 \cdot |R|) \) for span and \( O(l^2 \cdot |R|) \) for dependency, both of which are too computationally expensive.

To address this issue, we attempt to prune candidates using two beams for storing the candidate arguments and predicates with size \( \beta_p n \) and \( \beta_a n \) where \( \beta_p \) and \( \beta_a \) are

---

4 When \( i = j \), span reduces to dependency.
two manually set thresholds, a method inspired by He et al. (2018a). First, the predicate and argument candidates are ranked according to their predicted scores ($\phi_p$ and $\phi_a$, respectively), and then we reduce the predicate and argument candidates with defined beams. Finally, we take the candidates from the beams for use in label prediction. Such pruning will reduce the overall number of candidate tuples to $O(n^2 \cdot |R|)$ for both types of tasks. Furthermore, for span SRL, we set the maximum length of candidate arguments to $L$, which may decrease the number of candidate arguments to $O(n)$. Specifically, for predicates and arguments, we introduce two unary scores based on their candidates for ranking:

$$\phi_p = w_p \text{MLP}_p(g^p), \quad \phi_a = w_a \text{MLP}_a(g^a)$$

After pruning, we also adopt the biaffine scorer as in the tree-based models:

$$\Phi_r(p, a) = Biaffine(h^{(pred)}, h^{(arg)'}) \quad (4)$$

4. Syntax Utilization

In this section, we present two types of syntax utilization: syntax-based argument pruning and syntax feature integration.

4.1 Syntax-based Argument Pruning

*Hard Pruning.* The argument structure for each known predicate will be discovered by our argument labeler using the possible arguments (candidates) set. Most SRL works

---

5 Notably, “hard pruning” involves the removal of words from a sentence. This does not reflect how pruning techniques have been applied in previous work. Traditionally, pruning in SRL simply meant that an argument span was not considered as a candidate for a given predicate. Sentence structure is typically
Algorithm 1 The k-order argument pruning algorithm.

Input: A predicate p, the root node r given a syntactic dependency tree T, the order k
Output: The set of argument candidates S

1: initialization set p as current node c, c = p
2: for each descendant n_i of c in T do
3:   if D(c, n_i) \leq k and n_i \notin S then
4:      S = S + n_i
5:   end if
6: end for
7: find the syntactic head c_h of c, and let c = c_h
8: if c = r then
9:    S = S + r
10: else
11:   goto step 2
12: end if
13: return argument candidates set S

(Xue and Palmer 2004; Zhao and Kit 2008) in the pre-NN era selected words surrounding the predicate word in a syntactic parse tree and pruned these words. We refer to this strategy as hard pruning. In the NN model, we can also borrow this hard pruning strategy to enhance the SRL baseline, and it is one way of using syntax information. Specifically, before inputting to the model, we use the argument pruning algorithm to get a filtered sequence \( w_f = \{w_1, \ldots, w_f\} \) for each predicate. Then, we replace the original sequence with this one and input it to the SRL model.

As noted by Punyakanok, Roth, and Yih (2008), syntactic information is most relevant in identifying the arguments, and the most crucial contribution of full parsing is in the pruning stage. In this paper, we propose a k-order argument hard pruning algorithm inspired by Zhao, Chen, and Kit (2009). First, for node n and its descendant n_d in a syntactic dependency tree, we define the order to be the distance between the two nodes, denoted as \( D(n, n_d) \). Then, we define k-order descendants of n as descendants that satisfy \( D(n, n_d) = k \), and we define a k-order traversal as one that visits each node from the given node to its descendant nodes within k-th order. Note that the definition of k-order traversal is somewhat different from a traditional tree traversal in terminology.

A brief description of the proposed k-order pruning algorithm is given as follows. Initially, we set a given predicate as the current node in a syntactic dependency tree. Then, we collect all its argument candidates using a k-order traversal. Afterward, we reset the current node to its syntactic head and repeat the previous step until we reach the root of the tree. Finally, we collect the root and stop. The k-order argument algorithm is presented in Algorithm 1 in detail. An example of a syntactic dependency tree for the sentence She began to trade the art for money is shown in Figure 8.

The main reasons for applying the extended k-order argument pruning algorithm are two-fold. First, previous standard pruning algorithms may impede the argument coverage too much, even though arguments do usually tend to surround their predicates at a close distance. As a sequence tagging model that has been applied, the

not affected by this and pruned argument spans are, in fact, still used in the computation of features for other (unpruned) candidates.
algorithm can effectively handle the imbalanced distribution between arguments and non-arguments, which would be poorly handled by early argument classification models that commonly adopt the standard pruning algorithm. Second, the extended pruning algorithm provides a better trade-off between computational cost and performance by carefully tuning $k$.

**Soft Pruning.** For word pair classification modeling, one major performance bottleneck is caused by unbalanced data. This is especially pertinent for SRL, where more than 90% of argument candidates are non-arguments. The syntax-based hard pruning methods are thus proposed to alleviate the imbalanced distribution; however, these do not extend well to other baselines and languages and even hinder syntax-agnostic SRL models, as Cai et al. (2018) demonstrated using different $k$ values on English. This hindrance might result because this pruning method breaks up the whole sentence, leading the BiLSTM encoder to take the incomplete sentence as input and fail to learn sentence representation sufficiently.

To alleviate such a drawback from the previous syntax-based pruning methods, we propose a novel pruning rule extraction method based on syntactic parse trees that generally suits diverse baselines at the same time. In detail, we add an argument pruning layer guided by syntactic rules following BiLSTM layers, which can absorb the syntactic clues simply and effectively.

**Syntactic Rule.** All arguments are specific to a particular predicate. Researchers have found that in syntax trees, the distance between predicates and their arguments generally falls within a certain range for each language; in other words, the arguments of a predicate are typically close to their predicate in their syntactic parse tree (Xue and Palmer 2004; Zhao and Kit 2008; He et al. 2018b; He, Li, and Zhao 2019). Therefore, we introduce a language-specific rule based on syntactic dependency parses to prune some
unlikely arguments. We call this rule the **syntactic rule**. Specifically, given a predicate \( p \) and its argument \( a \), we define \( d_p \) and \( d_a \) to be the distance from \( p \) and \( a \) to their nearest common ancestor node (namely, the root of the minimal subtree that includes \( p \) and \( a \)), respectively. For example, 0 denotes that a predicate or argument itself is their nearest common ancestor, while 1 represents that their nearest common ancestor is the parent of the predicate or argument. Then, we use the distance tuple \((d_p, d_a)\) as their relative position representation inside the parse tree. Finally, we make a list of all tuples ordered according to how many times that each distance tuple occurs in the training data, which is counted for each language independently.

It is worth noting that our syntactic rule is determined by the top-\( k \) frequent distance tuples. During training and inference, the syntactic rule takes effect by excluding all candidate arguments whose predicate-argument relative positions in the parse tree are not in the list of top-\( k \) frequent tuples.

Figure 9 shows simplified examples of a syntactic dependency tree. Given an English sentence in Figure 9(a), the current predicate is *likes*, whose arguments are *cat* and *fish*. For *likes* and *cat*, the predicate (*likes*) is their common ancestor (denoted as Root\(_{arg}\)) according to the syntax tree. Therefore, the relative position representation of the predicate and argument is (0, 1), and it is the same for *likes* and *fish*. As for the right side in Figure 9, suppose the marked predicate has two arguments—\( arg_1 \) and \( arg_2 \). The common ancestors of the predicate and arguments are, respectively, Root\(_{arg_1}\) and Root\(_{arg_2}\). In this case, the relative position representations are (0, 1) and (1, 2).

**Argument Pruning Method.** To maintain the sequential inputs through the whole sentence, we propose a novel syntax-based method to softly prune arguments, which is unlike most existing works (Xue and Palmer 2004; Zhao et al. 2009a; He et al. 2018b) with hard pruning strategies that prune argument candidates in the pre-processing stage. Our soft pruning strategy is very straightforward. In the argument pruning layer, our model drops these candidate arguments (more exactly, their BiLSTM representations) that do not comply with the syntactic rule. In other words, only the predicates and arguments that satisfy the syntactic rule will be output to the next layer. Notably, whereas hard pruning removes some of words from each sentence and tasks the model with processing an incomplete sentence, with soft pruning, the model is given the full original sentence, and by applying a mask instead of discarding part of the inputs. While we do use a “hard” 0/1 binary mask for our “soft” pruning, this step can also

![Figure 9](http://direct.mit.edu/coli/article-pdf/doi/10.1162/coli_a_00408/1959757/coli_a_00408.pdf)

**Figure 9**
Syntactic parse tree examples (dependency relations are omitted). Red represents the current predicate, and blue indicates its arguments.
be softened to other preset probabilities such as 0.1/0.9 so that the pruned parts can still pass some information. We leave this as an exploration for future work.

**Constituent Pruning.** In dependency SRL, argument candidates are pruned by a heuristic search over the dependency syntax tree. Constituent syntax trees, which represent the phrasal compositions of sentences, and span SRL have a different relationship than do dependency syntax trees and dependency SRL, which have similarities in their dependency arcs and dependency semantic relations. Since the argument span boundary in span SRL is consistent with that of the phrase in constituent syntactic tree, we adopt a new constituent-based argument pruning method.

Constituency syntax breaks a sentence into constituents (i.e., phrases or spans), which naturally form a constituency tree in a top–down fashion. In contrast with the dependency syntax tree, words can only be the terminals in a constituency tree, while the non-terminals are phrases with types. In span SRL, each argument corresponds to a constituent in a constituency tree, which can thus be used to generate span argument candidates given the predicates (Xue and Palmer 2004; Carreras and Márquez 2005). Punyakanok, Roth, and Yih (2005) showed that constituency trees offer high-quality argument boundaries.

Considering that span SRL models only occasionally violate the syntactic constraints (some candidate arguments may not be constituents), we attempt to prune unlikely arguments based on these constraints, essentially ruling out the likely impossible candidates, albeit at the cost of missing some of the rare violating arguments.

In order to utilize such constituent boundaries in the constituency tree and help decide argument candidates, we extract all boundaries for a constituent $c$ to form a set $\text{boundaryset} = \{(\text{START}(c), \text{END}(c))\}$. We also define an argument pruning layer that drops candidate arguments whose boundaries are not in this set. It is worth noting that because span arguments are converted to BIO labels under the sequence-based and tree-based modeling approaches of span SRL, there is no explicit correspondence between the existing arguments and the constituents, so constituent-based argument pruning is not applicable to the sequence-based and tree-based modeling approaches. We only consider this syntax enhancement when using graph-based modeling.

### 4.2 Syntax Feature Integration

In addition to guiding argument pruning, another major use of syntax information is serving as a syntax-aware feature in addition to the contextualized representation, thereby enhancing the argument labeler. To integrate the syntactic information into sequential neural networks, we use a syntactic encoder on top of the BiLSTM encoder.

Specifically, given a syntactic dependency tree $T$, for each node $n_k$ in $T$, let $C(k)$ denote the syntactic children set of $n_k$, $H(k)$ denote the syntactic head of $n_k$, and $L(k, \cdot)$ denote the dependency relation between node $n_k$ and those that have a direct arc from or to $n_k$. Then, we formulate the syntactic encoder as a transformation $f^\tau$ over the node $n_k$, which may take some of $C(k)$, $H(k)$, or $L(k, \cdot)$ as input and compute a syntactic representation $v_k$ for node $n_k$, namely, $v_k = f^\tau(C(k), H(k), L(k, \cdot), x_k)$. When not otherwise specified, $x_k$ denotes the input feature representation of $n_k$, which may be either the word representation $e_k$ or the output of BiLSTM $h_{n_k}$. $\sigma$ denotes the logistic sigmoid function, and $\odot$ denotes the element-wise multiplication.

In practice, the transformation $f^\tau$ can be any syntax encoding method. In this paper, we will consider three types of syntactic encoders: syntactic graph convolutional
network (Syntactic GCN), syntax aware LSTM (SA-LSTM), and tree-structured LSTM (Tree-LSTM).

**Syntactic GCN.** The GCN (Kipf and Welling 2017) was proposed to induce the representations of nodes in a graph based on the properties of their neighbors. Given its effectiveness, Marcheggiani and Titov (2017) introduced a generalized version for the SRL task, namely, syntactic GCN, and showed that the syntactic GCN is effective in incorporating syntactic information into neural models.

The syntactic GCN captures syntactic information flowing in two directions: one from heads to dependents (along), and the other from dependents to heads (opposite). Additionally, it also models the information flows from a node to itself; that is, it assumes that a syntactic graph contains a self-loop for each node. Thus, the syntactic GCN transformation of a node $n_k$ is defined on its neighborhood $N(k) = C(k) \cup H(k) \cup \{n_k\}$. For each edge that connects $n_k$ and its neighbor $n_j$, we can compute a vector representation,

$$u_{k,j} = W^{\text{dir}(k,j)}x_j + b^{L(k,j)}$$

where $\text{dir}(k,j)$ denotes the direction type (along, opposite, or self-loop) of the edge from $n_k$ to $n_j$, $W^{\text{dir}(k,j)}$ is the direction-specific parameter, and $b^{L(k,j)}$ is the label-specific parameter. Considering that syntactic information from all the neighboring nodes may make different contributions to semantic role labeling, the syntactic GCN introduces an additional edge-wise gate for each node pair $(n_k, n_j)$ as

$$g_{k,j} = \sigma(W_g^{\text{dir}(k,j)}x_j + b_g^{L(k,j)})$$

The syntactic representation $v_k$ for a node $n_k$ can be then computed as:

$$v_k = \text{ReLU}(\sum_{j \in N(k)} g_{k,j} \odot u_{k,j})$$

**SA-LSTM.** The SA-LSTM (Qian et al. 2017) is an extension of the standard BiLSTM architecture, which aims to simultaneously encode the syntactic and contextual information for a given word. On the one hand, the SA-LSTM calculates the hidden state in timestep order as does the standard LSTM,

$$i_s = \sigma(W^{(i)}x_k + U^{(i)}h_{k-1} + b^{(i)})$$
$$f_s = \sigma(W^{(f)}x_k + U^{(f)}h_{k-1} + b^{(f)})$$
$$o_s = \sigma(W^{(o)}x_k + U^{(o)}h_{k-1} + b^{(o)})$$
$$u = f(W^{(u)}x_k + U^{(u)}h_{k-1} + b^{(u)})$$
$$c_k = i_s \odot u + f_s \odot c_{k-1}$$
On the other hand, it further incorporates the syntactic information into the representation of each word by introducing an additional gate,

\[
\begin{align*}
    s_g &= \sigma(W^{(s)}x_k + U^{(s)}h_{k-1} + b^{(s)}) \\
    h_k &= o_g \odot f(c_k) + s_g \odot \tilde{h}_k
\end{align*}
\]

where \(f(\cdot)\) and \(\sigma(\cdot)\) represent the tanh and sigmoid activation functions, \(\tilde{h}_k = f(\sum_{t_j < t_k} \alpha_j \times h_j)\) is the weighted sum of all hidden state vectors \(h_j\) in position \(t_j\) (that correspond to previous node [word] \(n_j\)) to current timestep \(t_k\), and the weight factor \(\alpha_j\) is actually a trainable weight related to the dependency relation \(L(k, \cdot)\) when there exists a directed edge from \(n_j\) to \(n_k\).

Note that \(\tilde{h}_k\) is always the hidden state vector of the syntactic head of \(n_k\) according to the definition of \(\alpha_j\). Because a word will be assigned a single syntactic head, such a strict constraint prevents the SA-LSTM from incorporating complex syntactic structures. Inspired by the GCN, we relax the directed constraint of \(\alpha_j\) whenever there is an edge between \(n_j\) and \(n_k\).

After the SA-LSTM transformation, the outputs of the SA-LSTM layer from both directions are concatenated and taken as the syntactic representation of each word \(n_k\), that is, \(v_k = [\vec{h}_k, \hat{h}_k]\). Different from the syntactic GCN, SA-LSTM encodes both syntactic and contextual information in a single vector \(v_k\).

Tree-LSTM. The Tree-LSTM (Tai, Socher, and Manning 2015) can be considered an extension of the standard LSTM and aims to model tree-structured topologies. At each timestep, it composes an input vector and the hidden states from arbitrarily many child units. Specifically, the main difference between the Tree-LSTM unit and the standard one is that the memory cell updating and the calculation of gating vectors are dependent on multiple child units. A Tree-LSTM unit can be connected to an arbitrary number of child units, and it assigns a single forget gate for each child unit. This provides Tree-LSTM the flexibility to incorporate or drop the information from each child unit.

Given a syntactic tree, the Tree-LSTM transformation is defined on node \(n_k\) and its children set \(C(k)\) and is formulated as follows Tai, Socher, and Manning (2015):

\[
\begin{align*}
    \tilde{h}_k &= \sum_{j \in C(k)} h_j \tag{5} \\
    i_g &= \sigma(W^{(i)}x_k + U^{(i)}\tilde{h}_k + b^{(i)}) \\
    f^{kj}_g &= \sigma(W^{(f)}x_k + U^{(f)}h_j + b^{(f)}) \\
    o_g &= \sigma(W^{(o)}x_k + U^{(o)}\tilde{h}_k + b^{(o)}) \\
    u &= \tanh(W^{(u)}x_k + U^{(u)}\tilde{h}_k + b^{(u)}) \\
    c_k &= i_g \odot u + \sum_{j \in C(k)} f^{kj}_g \odot c_j \\
    h_k &= o_g \odot \tanh(c_k)
\end{align*}
\]
where $j \in C(k)$, $h_j$ is the hidden state of the $j$-th child node, $c_k$ is the memory cell of the head node $k$, and $h_k$ is the hidden state of node $k$. Note that in Equation (6), a single forget gate $f_{g}^{k,j}$ is computed for each hidden state $h_j$.

Note that the primitive form of Tree-LSTM does not take the dependency relations into consideration. Given the importance of dependency relations in the SRL task, we further extend the Tree-LSTM by adding an additional gate $r_g$ and reformulate Equation (5),

\[
\begin{align*}
    r_{g}^{k,j} &= \sigma(W^{(r)}x_k + U^{(r)}h_j + b^{(l(k,j))}) \\
    \tilde{h}_k &= \sum_{j \in C(k)} r_{g}^{k,j} \odot h_j
\end{align*}
\]

where $b^{(l(k,j))}$ is a relation label-specific bias term. After the Tree-LSTM transformation, the hidden state of each node in the dependency tree is taken as its syntactic representation, that is, $v_k = h_k$.

### 4.3 Constituent Composition and Decomposition

Due to the difference in structure between constituent and dependency syntax trees, tree encoders (GCN, SA-LSTM, Tree-LSTM, etc.) cannot be used to encode the constituent tree directly. In order for the constituent syntax to be encoded into the SRL model as the dependency syntax tree was, inspired by Marcheggiani and Titov (2020), we introduce two processes: constituent tree conversion and feature decomposition.

A constituency tree is composed of terminal nodes and non-terminal nodes, as shown in Figure 10(a). Since the words in a constituent tree all are terminal nodes, if the constituent tree is directly encoded by the tree encoder, the syntax tree structural information cannot be encoded into the words fully. Therefore, we convert the constituent tree to a dependency-like tree, in which the original terminal nodes are removed and the remaining non-terminal nodes are replaced by units consisting of the start and end tokens (words) of the spans they represented. The constituent labels are modified to mimic dependency arcs as in dependency trees, as shown in Figure 10(b).

In a dependency tree, the nodes in the tree are the words in the sentence, so the syntax features output from the tree encoder can be mapped directly to the linear order of the sentences. In our converted constituent tree, the nodes in the tree correspond to the start and end words in the sentence, so we need an additional decomposition process to map this feature back to the word level. As shown in the dashed line in Figure 10(b), every node passes the feature to the first and the last words in their spans, and an extra indicator embedding $e_{\text{pos}}$ is appended to distinguish features as starts or ends. Then, these features are input to BiLSTM encoders to obtain the final syntax features for each word.

Specifically, before the tree encoding process, we concatenate the BiLSTM contextualized representations for the start and end positions of the span as the initial node representation, the start and end positions of leaf nodes are both positions themselves. For span $(i,j)$, the initial node representation is:

\[
h_{i,j}^i = \text{FFN}([h_i; h_j])
\]
Figure 10
The structure of an original constituent tree and its dependency-like constituent tree after conversion. The dashed lines map the converted tree encoded features back to the original positions of words in the sentence, i.e., feature decomposition.

where $[; ;]$ represents the concatenation operation, and FFN is a feed-forward layer that is used to keep the model dimension. The syntactic tree is then encoded by the adopted tree encoder to obtain the final tree node representations $h_{n}^t$. We separate the representations of the nodes by splicing different position representations to distinguish whether they come from span starts or ends, and then accumulate all of these representations from span starting or ending respectively:

$$h_i^t = \text{FFN}([ \sum_{i=n, start} [h_{n, start}^{spos}]; \sum_{i=n, end} [h_{n, end}^{spos}]])$$

where $h_i^t$ represents the syntactic tree features for word $w_i$ and $n$ is a tree node.

5. Experimental Analysis and Syntax Role Study

In this section, we investigate the proposed methods empirically in comparison to the latest SRL models. Moreover, we further explore the syntax role for neural SRL in various architectures. The SRL models are evaluated on the popular CoNLL-2005, CoNLL-2009, and CoNLL-2012 shared tasks following the standard training, development, and test splits. For the SRL task, because the predicate identification subtask is easier than other subtasks, some works only focus on the semantic role prediction with pre-identified predicates, which we name \textit{w/ pred}. There are also many studies that tend to examine settings closer to real-world scenarios, where the predicates are not given and the proposed systems are required to output both the predicates and their corresponding argument. We call this setting \textit{w/o pred}.

\begin{itemize}
  \item \textit{w/ pred}:
  \item \textit{w/o pred}:
\end{itemize}

\footnote{In the SA-LSTM tree encoder, a linear sequence of tree nodes is required, so we linearize the nodes of the constituent trees with a depth-first algorithm.}
The hyperparameters in our model were selected based on the development set. In our experiments, all real vectors are randomly initialized, including 100-dimensional word, lemma, POS tag embeddings, and 16-dimensional predicate-specific indicator embeddings (He et al. 2018b). The pre-trained word embeddings are 100-dimensional GloVe vectors (Pennington, Socher, and Manning 2014) for English and 300-dimensional fastText vectors (Grave et al. 2018) trained on Common Crawl and Wikipedia for other languages. The dimensions of ELMo and BERT word embeddings are of size 1024. Additionally, we use a 3-layer BiLSTM with 400-dimensional hidden states and apply dropout with an 80% keep probability between timesteps and layers. For the biaffine scorer, we use two 300-dimensional affine transformations with the ReLU non-linear activation and also set the dropout probability to 0.2. During training, we use the categorical cross-entropy as the objective and use the Adam optimizer (Kingma and Ba 2015) with an initial learning rate $2^{-3}$. All models are trained for up to 50 epochs with batch size 64.

In our syntax-based pruning experiments with dependency SRL, we used the predicted syntactic trees given by the CoNLL-2009 data sets (including the multilingual settings) due to the relative insignificance of the syntactic quality in the pruning algorithm. For our syntax feature integration experiments in dependency SRL, because the tree encoder has a minimum syntax tree quality requirement before it can provide any performance improvement, we obtained the predicted dependency syntax tree with the Biaffine Parser (Dozat and Manning 2017) we trained on the golden syntax annotations provided by CoNLL20-09 data sets. In span SRL, following the practice of He et al. (2017), a leading constituency parser (Choe and Charniak 2016) is used to parse the constituent trees. In our experiments, the data sets for the shared task of CoNLL-2009 include predicted POS tags or lemmas, while the data sets for CoNLL-2005 and CoNLL-2012 do not provide them. We remedy this using NLTK to obtain the predicted POS tags and lemmas to keep the input form consistent.

5.1 Data sets

Span-based Data. The CoNLL-2005 shared task focused on verbal predicates for only English. The CoNLL-2005 data set takes sections 2-21 of Wall Street Journal (WSJ) data 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. The larger CoNLL-2012 data set is extracted from OntoNotes v5.0 corpus, which contains both verbal and nominal predicates.

Dependency-based Data. The CoNLL-2009 shared task is focused on dependency-based SRL in multiple languages and merges two treebanks, PropBank and NomBank. NomBank is a complement to PropBank and uses a similar semantic convention for nominal predicate-argument structure annotation. The training, development, and test splits of the English data are identical to those of CoNLL-2005.

5.2 Preprocessing

Hard Pruning. During the pruning of argument candidates, we use the officially predicted syntactic parses provided by CoNLL-2009 shared-task organizers on both English and Chinese. Figure 11 shows changing curves of coverage and reduction following $k$ on the English train set. According to our statistics, the number of non-arguments is ten times more than that of arguments, meaning the data distribution
Figure 11
Changing curves of coverage and reduction with different $k$ values on the English training set. The coverage rate is the proportion of true arguments in the pruning output, while the reduction is the proportion of pruned argument candidates in total tokens.

is fairly unbalanced; however, a proper pruning strategy could alleviate this problem. Accordingly, the first-order pruning reduces more than 50% candidates at the cost of missing 5.5% true ones on average, and the second-order prunes about 40% candidates with nearly 2.0% loss. The coverage of third-order achieves 99% , and it reduces the size of the corpus by approximately one third.

It is worth noting that when $k$ is larger than 19, full coverage is achieved on all argument candidates for the English training set, which allows our high-order pruning algorithm to reduce to a syntax-agnostic setting. In this work, we use tenth-order pruning for best performance.

Soft Pruning. For the syntactic rule used in soft argument pruning, to ensure more than 99% coverage of true arguments in pruning output, we use the top-120 distance tuples on Japanese and top-20 on other languages for a better trade-off between computation and coverage.

Candidate Pruning. In graph-based modeling, the pruning of all predicate-argument pair candidates does not rely on hard pruning or soft pruning based on syntactic trees. Rather, it limits the maximum length of the argument span while also ranking the predicate and argument candidates separately using scores from the neural network scorer and then taking the top-$k$ candidates to reduce the number of candidates. Specifically, we follow the settings of He et al. (2018a), modeling spans up to length $L = 30$ for span SRL and $L = 1$ for dependency SRL, using $\beta_p = 0.4$ for pruning predicates and $\beta_a = 0.8$ for pruning arguments.

5.3 Dependency SRL Results

Undoubtedly, dependency SRL offers a number of advantages from a practical perspective, and the efficient dependency parsing algorithms enable SRL models to achieve state-of-the-art results. Therefore, we begin our exploration of syntax roles for neural
SRL with it. In Table 2, we outlined the performance of the current leading dependency SRL models and compared the performance of our three baselines and syntax-enhanced models with different integration approaches on the CoNLL-2009 English in-domain (WSJ) and out-of-domain (Brown) test sets.

In the sequence-based approaches, we utilized another sequence labeling model to tackle the predicate identification and disambiguation subtasks required for the different settings (\textit{w/ pred} and \textit{w/o pred}). The predicate disambiguation model achieves accuracies of 95.01% and 95.58% on the development and test (WSJ) sets, respectively, giving a slightly better accuracy than Roth and Lapata (2016),

<table>
<thead>
<tr>
<th>System</th>
<th>PLM</th>
<th>SYN</th>
<th>\textit{w/ pred} (WSJ)</th>
<th>\textit{w/o pred} (WSJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>P</td>
<td>R</td>
</tr>
<tr>
<td>(Zhao et al. 2009b)</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>85.8</td>
</tr>
<tr>
<td>(Zhao et al. 2009a)</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>86.1</td>
</tr>
<tr>
<td>(Lei et al. 2015)</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>86.6</td>
</tr>
<tr>
<td>(FitzGerald et al. 2015)</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>87.8</td>
</tr>
<tr>
<td>(Roth and Lapata 2016)</td>
<td>Y</td>
<td>90.0</td>
<td>89.5</td>
<td>78.6</td>
</tr>
<tr>
<td>(Swayamdipta et al. 2016)</td>
<td>Y</td>
<td>90.3</td>
<td>89.5</td>
<td>78.7</td>
</tr>
<tr>
<td>(Marcheggiani and Titov 2017)</td>
<td>Y</td>
<td>89.0</td>
<td>88.7</td>
<td>78.0</td>
</tr>
<tr>
<td>(Marcheggiani, Frolov, and Titov 2017)</td>
<td>N</td>
<td>88.7</td>
<td>86.8</td>
<td>79.4</td>
</tr>
<tr>
<td>(Zhang, Wang, and Si 2019)</td>
<td>Y</td>
<td>89.6</td>
<td>86.0</td>
<td>80.7</td>
</tr>
<tr>
<td>(Lyu, Cohen, and Titov 2019)</td>
<td>Y</td>
<td>90.7</td>
<td>91.4</td>
<td>82.8</td>
</tr>
<tr>
<td>(Chen, Lyu, and Titov 2019)</td>
<td>+E</td>
<td>N</td>
<td>88.7</td>
<td>89.8</td>
</tr>
</tbody>
</table>

### Table 2

Dependency SRL Results with pre-identified predicates (\textit{w/ pred}) and without pre-identified predicates (\textit{w/o pred}) on the CoNLL-2009 English in-domain (WSJ) and out-of-domain (Brown) test sets. The “PLM” column indicates whether a and which pre-trained language model is used, the “SYN” column indicates whether syntax information is employed, “+E” in the “PLM” column shows that the model leverages ELMo for pre-trained language model features. [Ens.] is used to specify the ensemble system, [Semi.] indicates semi-supervised training is adopted, and [Joint] means joint learning with other tasks.
which had 94.77% and 95.47% accuracy on development and test sets, respectively. As for the w/o pred setting, the $F_1$ score of our predicate labeling model is 90.11% and 90.53% on development and test (WSJ) sets, respectively. With the help of the ELMo pre-trained language model, our sequence-based baseline model has achieved competitive results when compared to the other leading SRL models. Compared to the closest work (Marcheggiani, Frolov, and Titov 2017), ELMo brought a 1.0% improvement for our baseline model, which verifies it is a strong baseline. In this case, the syntax enhancement gave us a performance improvement of 0.8%–1.1% (w/ pred, in-domain test set), demonstrating that both hard/soft pruning and syntax encoders effectively exploit syntax for sequence-based neural SRL models.

In the tree-based approaches, the predicate disambiguation subtask is unified tackled with argument labeling by making predictions on the ROOT node of the factorized tree. The disambiguation precision is 95.0% in the w/ pred setting, whereas in the w/o pred setting, we first attach all the words in the sentence to the ROOT node and label the word that is not a predicate with the null role label. It should be noted that in the w/o pred setting, we just attach the predicates to the ROOT node, since we do not need to distinguish the predicate from other words. The training scheme remains the same as in the w/ pred setting, whereas in the inference phase, an additional procedure is performed to find out all the predicates of a given sentence. The $F_1$ score on predicates identification and labeling of this process is 89.43%. Based on the tree-based baseline, the hard pruning syntax enhancement fails to improve on the baseline despite the hard pruning method’s ability to alleviate the imbalanced label distribution caused by the null role labels. We suspect the possible reason is the use of a biaffine attention structure, which already alleviates imbalanced label distribution issue. This is problematic because both the biaffine attention and hard pruning work to balance the label distribution, and after the biaffine attention balances it to some degree, hard pruning is much more likely to incorrectly prune true labels, which potentially even leads to a decrease in performance. Compared with hard pruning, soft pruning can greatly reduce the incorrect pruning of true arguments, which serve as clues in the model. Because of this, the soft pruning algorithm applied to the tree-based model can obtain performance improvement similar to that of the tree-based baseline. In addition, the performance improvements of syntax encoders in the tree-based model are similar to those of the sequence-based model.

In the graph-based approaches, because the introduction of candidate pruning, argument pruning is directly controlled by the neural network scorer, both syntax-based hard and soft pruning methods lose the effects they provided in the sequence-based and tree-based models. The syntax encoder, however, can provide quite stable performance improvement as it does in sequence-based and tree-based models.

Though most SRL literature is dedicated to impressive performance gains on the English benchmark, exploring syntax’s enhancing effects on diverse languages is also important for examining the role of syntax in SRL. Table 3 presents all in-domain test results on seven languages of CoNLL-2009 data sets. Compared with previous methods, our baseline yields strong performance on all data sets. Nevertheless, applying the syntax information to the strong syntax-agnostic baseline can still boost the model performance in general, which demonstrates the effectiveness of syntax information. On the other hand, the similar performance impact of hard/soft argument pruning on the baseline models indicates that syntax is generally beneficial to multiple languages and can enhance multilingual SRL performance with effective syntactic integration.

Based on the above results and analysis, we conclude that the syntax-based pruning algorithms proposed before the era of neural network can still play a role under certain
Table 3
Dependency SRL results on the CoNLL-2009 multilingual in-domain test sets with pre-identified predicates (w/pred) setting. The first row is the best result of the CoNLL-2009 shared task (Hajič et al. 2009). The “PLM” column indicates whether and which pre-trained language models is used, the “SYN” column indicates whether syntax information is employed, “+E” in the “PLM” column indicates the model leverages pre-trained ELMo features for all languages, “+E†” in the “PLM” column represents ELMo is only used for English.

<table>
<thead>
<tr>
<th>System</th>
<th>PLM</th>
<th>SYN</th>
<th>CA</th>
<th>CS</th>
<th>DE</th>
<th>EN</th>
<th>ES</th>
<th>JA</th>
<th>ZH</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoNLL-2009 best</td>
<td>Y</td>
<td>80.3</td>
<td>85.6</td>
<td>79.7</td>
<td>86.2</td>
<td>80.5</td>
<td>78.3</td>
<td>78.6</td>
<td>81.4</td>
<td></td>
</tr>
<tr>
<td>(Zhao et al. 2009a)</td>
<td>Y</td>
<td>80.3</td>
<td>85.2</td>
<td>76.0</td>
<td>85.4</td>
<td>80.5</td>
<td>78.2</td>
<td>77.7</td>
<td>80.5</td>
<td></td>
</tr>
<tr>
<td>(Roth and Lapata 2016)</td>
<td>Y</td>
<td>–</td>
<td>–</td>
<td>80.1</td>
<td>87.7</td>
<td>80.2</td>
<td>–</td>
<td>79.4</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>(Marcheggiani and Titov 2017)</td>
<td>Y</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>88.0</td>
<td>–</td>
<td>–</td>
<td>82.5</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>(Marcheggiani, Frolov, and Titov 2017)</td>
<td>N</td>
<td>–</td>
<td>86.0</td>
<td>–</td>
<td>87.7</td>
<td>80.3</td>
<td>–</td>
<td>81.2</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>(Mulcaire, Swayamdipta, and Smith 2018)</td>
<td>N</td>
<td>79.5</td>
<td>85.1</td>
<td>70.0</td>
<td>87.2</td>
<td>77.3</td>
<td>76.0</td>
<td>81.9</td>
<td>79.6</td>
<td></td>
</tr>
<tr>
<td>(Kasai et al. 2019)</td>
<td>+E</td>
<td>Y</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>90.2</td>
<td>–</td>
<td>83.0</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>(Cai and Lapata 2019a)</td>
<td>N</td>
<td>–</td>
<td>83.3</td>
<td>90.7</td>
<td>82.1</td>
<td>–</td>
<td>–</td>
<td>84.6</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>(Semli) (Cai and Lapata 2019a)</td>
<td>N</td>
<td>–</td>
<td>83.8</td>
<td>91.2</td>
<td>82.9</td>
<td>–</td>
<td>–</td>
<td>85.0</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>(Zhang, Wang, and Si 2019)</td>
<td>Y</td>
<td>–</td>
<td>–</td>
<td>87.7</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>84.2</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>(Lyu, Cohen, and Titov 2019)</td>
<td>+E†</td>
<td>N</td>
<td>80.9</td>
<td>87.6</td>
<td>75.9</td>
<td>91.0</td>
<td>80.5</td>
<td>82.5</td>
<td>83.3</td>
<td>83.1</td>
</tr>
<tr>
<td>(Chen, Lyu, and Titov 2019)</td>
<td>+E†</td>
<td>N</td>
<td>81.7</td>
<td>88.1</td>
<td>76.4</td>
<td>91.1</td>
<td>81.3</td>
<td>81.3</td>
<td>81.7</td>
<td>83.1</td>
</tr>
<tr>
<td><strong>Sequence-based</strong> (2018b; 2018)</td>
<td>+E</td>
<td>N</td>
<td>84.0</td>
<td>87.8</td>
<td>76.8</td>
<td>88.7</td>
<td>82.9</td>
<td>82.8</td>
<td>83.1</td>
<td>83.7</td>
</tr>
<tr>
<td>+K-order Hard Pruning (2018b)</td>
<td>+E</td>
<td>Y</td>
<td>84.5</td>
<td>88.3</td>
<td>77.3</td>
<td>89.5</td>
<td>83.3</td>
<td>82.9</td>
<td>82.8</td>
<td>84.1</td>
</tr>
<tr>
<td>+SynRule Soft Pruning</td>
<td>+E</td>
<td>Y</td>
<td>84.4</td>
<td>88.2</td>
<td>77.5</td>
<td>89.5</td>
<td>83.2</td>
<td>83.0</td>
<td>83.3</td>
<td>84.2</td>
</tr>
<tr>
<td>+GCN Syntax Encoder (2018)</td>
<td>+E</td>
<td>Y</td>
<td>84.6</td>
<td>88.5</td>
<td>77.2</td>
<td>89.8</td>
<td>83.6</td>
<td>83.2</td>
<td>83.8</td>
<td>84.4</td>
</tr>
<tr>
<td>+SA-LSTM Syntax Encoder (2018)</td>
<td>+E</td>
<td>Y</td>
<td>84.3</td>
<td>88.5</td>
<td>77.0</td>
<td>89.7</td>
<td>83.5</td>
<td>83.1</td>
<td>83.5</td>
<td>84.2</td>
</tr>
<tr>
<td>+Tree-LSTM Syntax Encoder (2018)</td>
<td>+E</td>
<td>Y</td>
<td>84.1</td>
<td>88.3</td>
<td>76.9</td>
<td>89.4</td>
<td>83.2</td>
<td>82.9</td>
<td>83.4</td>
<td>84.0</td>
</tr>
<tr>
<td><strong>Tree-based</strong> (2018)</td>
<td>+E</td>
<td>N</td>
<td>84.1</td>
<td>88.4</td>
<td>78.4</td>
<td>89.9</td>
<td>83.5</td>
<td>83.0</td>
<td>84.0</td>
<td>84.5</td>
</tr>
<tr>
<td>+K-order Hard Pruning</td>
<td>+E</td>
<td>Y</td>
<td>84.2</td>
<td>88.5</td>
<td>78.4</td>
<td>89.9</td>
<td>83.4</td>
<td>82.8</td>
<td>84.2</td>
<td>84.7</td>
</tr>
<tr>
<td>+SynRule Soft Pruning</td>
<td>+E</td>
<td>Y</td>
<td>84.4</td>
<td>88.8</td>
<td>78.5</td>
<td>90.0</td>
<td>83.7</td>
<td>83.1</td>
<td>84.6</td>
<td>84.7</td>
</tr>
<tr>
<td>+GCN Syntax Encoder</td>
<td>+E</td>
<td>Y</td>
<td>84.8</td>
<td>89.3</td>
<td>78.1</td>
<td>90.2</td>
<td>84.0</td>
<td>83.3</td>
<td>85.0</td>
<td>85.0</td>
</tr>
<tr>
<td>+SA-LSTM Syntax Encoder</td>
<td>+E</td>
<td>Y</td>
<td>84.6</td>
<td>89.0</td>
<td>78.8</td>
<td>90.0</td>
<td>83.7</td>
<td>83.1</td>
<td>84.8</td>
<td>84.9</td>
</tr>
<tr>
<td>+Tree-LSTM Syntax Encoder</td>
<td>+E</td>
<td>Y</td>
<td>84.4</td>
<td>88.9</td>
<td>78.6</td>
<td>89.9</td>
<td>83.6</td>
<td>83.0</td>
<td>84.5</td>
<td>84.7</td>
</tr>
<tr>
<td><strong>Graph-based</strong> (2019a)</td>
<td>+E</td>
<td>N</td>
<td>85.0</td>
<td>90.2</td>
<td>76.0</td>
<td>90.0</td>
<td>83.8</td>
<td>82.7</td>
<td>85.7</td>
<td>84.8</td>
</tr>
<tr>
<td>+K-order Hard Pruning</td>
<td>+E</td>
<td>Y</td>
<td>84.9</td>
<td>90.2</td>
<td>75.7</td>
<td>89.8</td>
<td>83.5</td>
<td>82.8</td>
<td>85.8</td>
<td>84.7</td>
</tr>
<tr>
<td>+SynRule Soft Pruning</td>
<td>+E</td>
<td>Y</td>
<td>85.2</td>
<td>90.3</td>
<td>76.2</td>
<td>90.1</td>
<td>84.0</td>
<td>82.9</td>
<td>85.8</td>
<td>84.9</td>
</tr>
<tr>
<td>+GCN Syntax Encoder</td>
<td>+E</td>
<td>Y</td>
<td>85.5</td>
<td>90.5</td>
<td>76.6</td>
<td>90.4</td>
<td>84.3</td>
<td>83.2</td>
<td>86.1</td>
<td>85.2</td>
</tr>
<tr>
<td>+SA-LSTM Syntax Encoder</td>
<td>+E</td>
<td>Y</td>
<td>85.2</td>
<td>90.5</td>
<td>76.4</td>
<td>90.3</td>
<td>84.1</td>
<td>83.2</td>
<td>86.0</td>
<td>85.1</td>
</tr>
<tr>
<td>+Tree-LSTM Syntax Encoder</td>
<td>+E</td>
<td>Y</td>
<td>85.0</td>
<td>90.3</td>
<td>76.2</td>
<td>90.3</td>
<td>84.0</td>
<td>83.0</td>
<td>85.8</td>
<td>84.9</td>
</tr>
</tbody>
</table>

conditions in the neural network era; however, when there are neural structures whose motivation is consistent with the original intention of these syntax-based pruning algorithms emerges, the effects of these algorithms are quite limited, and they can even have negative effects. Despite this limitation, the neural syntax encoder is beneficial in that it delegates the decisions of how to include syntax and what kind of syntax to use to neural networks, which reduces the number of manually defined features. This is an important result of the transition from the pre-neural network era to the neural network era. Handcrafted features may be advantageous when compared to poorly designed neural networks, but well-designed neural networks can easily outperform models relying on handcrafted features. In addition, neural models can also benefit from handcrafted features, so the two do not necessarily have to be directly compared, even though neural networks reduce the need for handcrafted features.

5.4 Span SRL Results
Apart from the dependency SRL experiments, we also conducted experiments to compare different syntax utilization on span SRL models. Table 4 shows results on the
Table 4
Span SRL results with pre-identified predicates on the CoNLL-2005 and CoNLL-2012 test sets. The “PLM” column indicates whether a and which pre-trained language model is used, the “SYN” column indicates whether syntax information is employed, “+E” in the “PLM” column shows that the model leverages the ELMo for pre-trained language model features. [Ens.] is used to specify the ensemble system and [Joint] means joint learning with other tasks.

<table>
<thead>
<tr>
<th>System</th>
<th>PLM</th>
<th>SYN</th>
<th>CoNLL05 WSJ P</th>
<th>R</th>
<th>F</th>
<th>CoNLL05 Brown P</th>
<th>R</th>
<th>F</th>
<th>CoNLL12 P</th>
<th>R</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[Ens.] (Punyakanok, Roth, and Yih 2008)</td>
<td>Y</td>
<td>82.3</td>
<td>76.8</td>
<td>79.4</td>
<td>73.4</td>
<td>62.9</td>
<td>67.8</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(Toutanova, Haghihi, and Manning 2008)</td>
<td>Y</td>
<td>81.9</td>
<td>78.8</td>
<td>80.3</td>
<td>-</td>
<td>-</td>
<td>68.8</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(Pradhan et al. 2013)</td>
<td>Y</td>
<td>82.3</td>
<td>77.6</td>
<td>79.9</td>
<td>74.3</td>
<td>68.6</td>
<td>71.5</td>
<td>80.6</td>
<td>78.2</td>
<td>79.4</td>
<td></td>
</tr>
<tr>
<td>(Täckström, Ganchev, and Das 2015)</td>
<td>Y</td>
<td>82.5</td>
<td>78.2</td>
<td>80.3</td>
<td>74.5</td>
<td>70.0</td>
<td>72.2</td>
<td>81.2</td>
<td>79.0</td>
<td>80.1</td>
<td></td>
</tr>
<tr>
<td>[Ens.] (FitzGerald et al. 2015)</td>
<td>Y</td>
<td>83.1</td>
<td>83.0</td>
<td>83.1</td>
<td>72.9</td>
<td>71.4</td>
<td>72.1</td>
<td>81.7</td>
<td>81.6</td>
<td>81.7</td>
<td></td>
</tr>
<tr>
<td>(He et al. 2017)</td>
<td>Y</td>
<td>83.0</td>
<td>84.3</td>
<td>84.6</td>
<td>74.9</td>
<td>72.4</td>
<td>76.6</td>
<td>83.3</td>
<td>83.3</td>
<td>83.4</td>
<td></td>
</tr>
<tr>
<td>(Yang and Mitchell 2017)</td>
<td>N</td>
<td>81.9</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(Tan et al. 2018)</td>
<td>Y</td>
<td>84.5</td>
<td>85.2</td>
<td>84.8</td>
<td>73.5</td>
<td>74.6</td>
<td>74.1</td>
<td>81.9</td>
<td>83.6</td>
<td>82.7</td>
<td></td>
</tr>
<tr>
<td>[Ens.] (Toutanova, Haghighi, and Manning 2008)</td>
<td>Y</td>
<td>81.8</td>
<td>77.3</td>
<td>79.4</td>
<td>73.8</td>
<td>68.8</td>
<td>71.2</td>
<td>80.9</td>
<td>78.4</td>
<td>79.6</td>
<td></td>
</tr>
<tr>
<td>(Punyakanok, Roth, and Yih 2008)</td>
<td>N</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(Toutanova, Haghighi, and Manning 2008)</td>
<td>Y</td>
<td>82.9</td>
<td>82.8</td>
<td>82.8</td>
<td>70.7</td>
<td>68.2</td>
<td>69.4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(FitzGerald et al. 2015)</td>
<td>Y</td>
<td>82.5</td>
<td>78.2</td>
<td>80.3</td>
<td>74.5</td>
<td>70.0</td>
<td>72.2</td>
<td>81.2</td>
<td>79.0</td>
<td>80.1</td>
<td></td>
</tr>
<tr>
<td>(He et al. 2017)</td>
<td>Y</td>
<td>83.1</td>
<td>83.0</td>
<td>83.1</td>
<td>72.9</td>
<td>71.4</td>
<td>72.1</td>
<td>81.7</td>
<td>81.6</td>
<td>81.7</td>
<td></td>
</tr>
<tr>
<td>[Ens.] (FitzGerald et al. 2015)</td>
<td>Y</td>
<td>85.0</td>
<td>84.3</td>
<td>84.6</td>
<td>74.9</td>
<td>72.4</td>
<td>76.6</td>
<td>83.3</td>
<td>83.3</td>
<td>83.4</td>
<td></td>
</tr>
<tr>
<td>(He et al. 2018a)</td>
<td>N</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(He et al. 2018b)</td>
<td>N</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(Staub, Ganchev, and Das 2015)</td>
<td>Y</td>
<td>84.6</td>
<td>84.6</td>
<td>84.6</td>
<td>74.8</td>
<td>74.8</td>
<td>74.6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(Toutanova, Haghighi, and Manning 2008)</td>
<td>Y</td>
<td>84.7</td>
<td>84.2</td>
<td>84.3</td>
<td>73.9</td>
<td>72.4</td>
<td>73.1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(Peters et al. 2018)</td>
<td>N</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(He et al. 2018a)</td>
<td>N</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(Strubell et al. 2018)</td>
<td>Y</td>
<td>84.6</td>
<td>84.6</td>
<td>84.6</td>
<td>74.8</td>
<td>74.8</td>
<td>74.6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(Ouchi, Shindo, and Matsumoto 2018)</td>
<td>Y</td>
<td>84.7</td>
<td>83.9</td>
<td>83.5</td>
<td>76.0</td>
<td>70.4</td>
<td>73.3</td>
<td>84.4</td>
<td>81.7</td>
<td>83.0</td>
<td></td>
</tr>
<tr>
<td>(Wang et al. 2019)</td>
<td>N</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(Marcheggiani and Titov 2020)</td>
<td>Y</td>
<td>85.8</td>
<td>85.1</td>
<td>85.4</td>
<td>76.2</td>
<td>74.7</td>
<td>75.5</td>
<td>84.3</td>
<td>84.3</td>
<td>84.4</td>
<td></td>
</tr>
<tr>
<td>[Joint] (Zhou, Li, and Zhao 2020)</td>
<td>N</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Sequence-based</td>
<td>+E</td>
<td>N</td>
<td>87.4</td>
<td>85.6</td>
<td>86.5</td>
<td>80.0</td>
<td>78.1</td>
<td>79.0</td>
<td>84.2</td>
<td>85.6</td>
<td>84.9</td>
</tr>
<tr>
<td>+GCN Syntax Encoder</td>
<td>+E</td>
<td>Y</td>
<td>87.2</td>
<td>86.8</td>
<td>87.0</td>
<td>78.6</td>
<td>80.2</td>
<td>79.4</td>
<td>85.3</td>
<td>85.7</td>
<td>85.5</td>
</tr>
<tr>
<td>+SA-LSTM Syntax Encoder</td>
<td>+E</td>
<td>Y</td>
<td>87.1</td>
<td>86.5</td>
<td>86.8</td>
<td>79.3</td>
<td>78.9</td>
<td>79.1</td>
<td>85.9</td>
<td>84.3</td>
<td>85.1</td>
</tr>
<tr>
<td>Tree-based</td>
<td>+E</td>
<td>Y</td>
<td>87.1</td>
<td>85.9</td>
<td>86.5</td>
<td>78.8</td>
<td>79.2</td>
<td>79.0</td>
<td>85.2</td>
<td>84.2</td>
<td>84.7</td>
</tr>
<tr>
<td>+GCN Syntax Encoder</td>
<td>+E</td>
<td>N</td>
<td>88.8</td>
<td>86.0</td>
<td>87.4</td>
<td>79.9</td>
<td>79.5</td>
<td>79.7</td>
<td>86.6</td>
<td>84.8</td>
<td>85.7</td>
</tr>
<tr>
<td>+SA-LSTM Syntax Encoder</td>
<td>+E</td>
<td>Y</td>
<td>87.7</td>
<td>88.3</td>
<td>88.0</td>
<td>81.1</td>
<td>79.9</td>
<td>80.5</td>
<td>86.9</td>
<td>85.5</td>
<td>86.2</td>
</tr>
<tr>
<td>+Tree-LSTM Syntax Encoder</td>
<td>+E</td>
<td>Y</td>
<td>87.5</td>
<td>87.6</td>
<td>87.6</td>
<td>80.4</td>
<td>79.8</td>
<td>80.1</td>
<td>86.3</td>
<td>85.3</td>
<td>85.8</td>
</tr>
<tr>
<td>Graph-based (2019a)</td>
<td>+E</td>
<td>N</td>
<td>87.0</td>
<td>87.6</td>
<td>87.3</td>
<td>81.0</td>
<td>79.1</td>
<td>80.0</td>
<td>86.0</td>
<td>85.6</td>
<td>85.8</td>
</tr>
<tr>
<td>+Constituent Soft Pruning</td>
<td>+E</td>
<td>Y</td>
<td>88.4</td>
<td>87.4</td>
<td>87.9</td>
<td>80.9</td>
<td>80.3</td>
<td>80.6</td>
<td>85.5</td>
<td>86.9</td>
<td>86.2</td>
</tr>
<tr>
<td>+GCN Syntax Encoder</td>
<td>+E</td>
<td>Y</td>
<td>88.9</td>
<td>88.2</td>
<td>88.6</td>
<td>80.8</td>
<td>81.2</td>
<td>81.0</td>
<td>87.2</td>
<td>86.5</td>
<td>86.7</td>
</tr>
<tr>
<td>+SA-LSTM Syntax Encoder</td>
<td>+E</td>
<td>Y</td>
<td>88.6</td>
<td>87.8</td>
<td>88.2</td>
<td>81.0</td>
<td>81.2</td>
<td>81.1</td>
<td>87.0</td>
<td>85.8</td>
<td>86.4</td>
</tr>
<tr>
<td>+Tree-LSTM Syntax Encoder</td>
<td>+E</td>
<td>Y</td>
<td>86.9</td>
<td>89.1</td>
<td>88.0</td>
<td>81.5</td>
<td>80.3</td>
<td>80.9</td>
<td>86.6</td>
<td>86.0</td>
<td>86.3</td>
</tr>
</tbody>
</table>

CoNLL-2005 in-domain (WSJ) and out-of-domain (Brown) test sets, as well as the CoNLL-2012 test set (OntoNotes). The first block of the table presents results from previous work. These results demonstrate that with the development of neural networks, in particular the emergence of pre-trained language models, SRL achieved a large performance increase of more than 8.0%, and syntax further enhanced the effect of these strong baselines, enabling syntax+pre-trained language models to achieve the state-of-the-art results (Wang et al. 2019). This indicates that the effect of SRL can still be improved as long as the syntax is used properly under current circumstances.

Additionally, comparing the results of Strubell et al. (2018) and He et al. (2018a), it can be found that the feature extraction ability of self-attention is stronger than that of RNN, and the self-attention baseline obviously outperforms RNN-based one, but when syntactic information or a pre-trained language model is used to enhance performance, the performance margin becomes smaller. Therefore, we can speculate...
that self-attention implicitly and partially functions as the syntax information, as do pre-trained language models.

By comparing our full model to state-of-the-art SRL systems, we show that our model genuinely benefits from incorporating syntactic information and other modeling factorization. Although our use of constituent syntax requires the composition and decomposition processes, which contrasts the simple and intuitive dependency syntax, we achieved consistent improvements compared to dependency SRL on all three baselines: sequence-based, tree-based and graph-based. This shows that these syntax encoders are general for syntax choice and can encode syntax effectively.

Constituent syntax information is usually used in span SRL, while the dependency tree is adopted for the argument pruning algorithm. Additionally, in the sequence-based and tree-based factorizations of span SRL, argument spans are linearized with multiple B-, I-, and O- tags, which alleviates some label imbalance problems and hence lessens the need for argument pruning. Constituent syntax mainly provides the boundary information of span for the model to guide the model to predict the correct argument span when it is used for pruning. In graph-based models, because the argument span exists alone, the boundary set obtained by the constituent tree can be used to prune candidate arguments. As shown in the results, constituent-based soft pruning can still improve performance on the graph-based baseline, but the improvement is smaller than that of syntax encoders, indicating that the syntax encoder can extract more information than just span boundaries.

We report the experimental results on the CoNLL-2005 and -2012 data sets without pre-identified predicates in Table 5. Overall, our syntax-enhanced model using ELMo
achieved the best $F_1$ scores on the CoNLL-2005 in-domain and CoNLL-2012 test sets. In comparison with the three baselines, our syntax utilization approaches consistently yielded better $F_1$ scores regardless of the factorization. Although the performance difference is small when using the constituent soft pruning on the graph-based model, the improvement seems natural because the constituent syntax for SRL has more to be explored than just boundary information.

5.5 Dependency vs. Span

It is very hard to say which style of semantic formal representation, dependency or span, would be more convenient for machine learning as they adopt incomparable evaluation metrics. Recent research (Peng et al. 2018) has proposed to learn semantic parsers from multiple data sets in FrameNet style semantics, while our goal is to compare the quality of different models in span and dependency SRL for Propbank style semantics. Following Johansson and Nugues (2008a), we choose to directly compare their performance in terms of dependency-style metric through transformation. Using the head-finding algorithm in Johansson and Nugues (2008a) which used gold-standard syntax, we may determine a set of head nodes for each span. This process will output an upper bound performance measure about the span conversion due to the use of gold syntax.

Based on our syntax-agnostic graph-based baseline, we do not train new models for the conversion and the resultant comparison. Instead, we use the span-style CoNLL-2005 test set and the dependency-style CoNLL-2009 test set (WSJ and Brown), considering these two test sets share the same text content. As the former only contains verbal predicate-argument structures, for the latter, we discard all nominal predicate-argument related results and predicate disambiguation results during performance statistics. CoNLL-2005 and CoNLL-2009 have inconsistent tokenization standards, but there exists a straightforward conversion that does not skew results. Because CoNLL-2009 tokenizes words into more fine-grained segments, in order to avoid the need to modify the dependency tree annotations, we re-tokenize the data of CoNLL-2005 according to the tokenization standard adopted in CoNLL-2009. For the constituent trees and span SRL annotations, this further tokenization will not affect the boundary of the spans, so the constituent tree and span SRL annotations are not affected. The detailed conversion process can be found in our paper (Li et al. 2019b).

Table 6 shows the comparison. On a more strict setting basis, the results from our same model for span and dependency SRL verify the same conclusion of Johansson and Nugues (2008a), namely, dependency form is more favorable in machine learning.

<table>
<thead>
<tr>
<th></th>
<th>Dep $F_1$</th>
<th>Span-converted $F_1$</th>
<th>$\Delta F_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WSJ</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J &amp; N</td>
<td>85.93</td>
<td>84.32</td>
<td>1.61</td>
</tr>
<tr>
<td>Our system</td>
<td>90.41</td>
<td>89.20</td>
<td>1.21</td>
</tr>
<tr>
<td><strong>WSJ+Brown</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J &amp; N</td>
<td>84.29</td>
<td>83.45</td>
<td>0.84</td>
</tr>
<tr>
<td>Our system</td>
<td>88.91</td>
<td>88.23</td>
<td>0.68</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Span $F_1$</th>
<th>Dep-converted $F_1$</th>
<th>$\Delta F_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WSJ</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Our system</td>
<td>87.70</td>
<td>87.23</td>
<td>0.47</td>
</tr>
<tr>
<td><strong>WSJ+Brown</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Our system</td>
<td>86.03</td>
<td>85.62</td>
<td>0.41</td>
</tr>
</tbody>
</table>
for SRL even compared to the conversion upper bound of the span form. While we used golden dependency trees for the conversion from span SRL to dependency SRL, some argument spans and constituent spans differ, meaning an argument span can have multiple head words. This means the conversion is not fully accurate and thus produces errors. To prevent these errors from impacting our evaluation, we also added the conversion from dependency SRL to span SRL. From the comparison, we found that although there are errors in both directions of conversion, the difference between the result of dependency-converted SRL and original span SRL system is less than that of span-converted and original dependency SRL system. With this observation, we can roughly conclude that dependency is a more computationally friendly form (Johansson and Nugues 2008b). This suggests that if we can only perform one type of SRL training but need two types of SRL outputs, we can prioritize selecting the training dependency SRL and then getting the span SRL based on the conversion to achieve the best overall performance.

5.6 Syntax Role under Different Pre-trained Language Models

Language modeling, as an unsupervised natural language training technique, can produce a pre-trained language model by training generally on a large amount of text before further training on a more specific one out of a variety of natural language processing tasks. The downstream tasks then use the obtained pre-trained models for further enhancement. After the introduction of pre-trained language models, they quickly and greatly dominated the performance of downstream tasks. Typical language models are ELMo (Peters et al. 2018), BERT (Devlin et al. 2019), RoBERTa (Liu et al. 2019), XLNet (Yang et al. 2019), and ALBERT (Lan et al. 2020), and so on. Therefore, in order to further explore the role of syntax in SRL on the strong basis of pre-trained language models, we evaluated the performance of these typical pre-trained language models on the best-performing combination of the graph-based baseline and GCN syntax encoder. The experimental results are shown in Table 7.

<table>
<thead>
<tr>
<th>System</th>
<th>SYN</th>
<th>CoNLL09 WSJ</th>
<th>CoNLL09 Brown</th>
<th>CoNLL05 WSJ</th>
<th>CoNLL05 Brown</th>
<th>CoNLL12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>N</td>
<td>87.8</td>
<td>79.2</td>
<td>84.5</td>
<td>74.8</td>
<td>83.5</td>
</tr>
<tr>
<td></td>
<td>Y</td>
<td>89.2 (+1.4)</td>
<td>80.1 (+0.9)</td>
<td>85.6 (+1.1)</td>
<td>76.2 (+1.4)</td>
<td>84.7 (+1.2)</td>
</tr>
<tr>
<td>ELMo</td>
<td>N</td>
<td>90.4 (+2.6)</td>
<td>81.5 (+2.3)</td>
<td>87.7 (+3.2)</td>
<td>80.5 (+5.7)</td>
<td>86.0 (+2.5)</td>
</tr>
<tr>
<td></td>
<td>Y</td>
<td>91.1 (+0.7)</td>
<td>82.1 (+0.6)</td>
<td>88.6 (+0.9)</td>
<td>81.0 (+0.5)</td>
<td>86.7 (+0.7)</td>
</tr>
<tr>
<td>BERT</td>
<td>N</td>
<td>91.4 (+3.6)</td>
<td>82.8 (+3.6)</td>
<td>89.0 (+4.5)</td>
<td>82.3 (+7.5)</td>
<td>87.4 (+3.9)</td>
</tr>
<tr>
<td></td>
<td>Y</td>
<td>91.8 (+0.4)</td>
<td>83.2 (+0.4)</td>
<td>89.6 (+0.6)</td>
<td>82.8 (+0.5)</td>
<td>87.9 (+0.5)</td>
</tr>
<tr>
<td>RoBERTa</td>
<td>N</td>
<td>91.4 (+3.6)</td>
<td>83.1 (+3.9)</td>
<td>89.3 (+4.8)</td>
<td>82.7 (+7.9)</td>
<td>87.9 (+4.4)</td>
</tr>
<tr>
<td></td>
<td>Y</td>
<td>91.7 (+0.3)</td>
<td>83.2 (+0.1)</td>
<td>89.7 (+0.4)</td>
<td>83.4 (+0.7)</td>
<td>88.0 (+0.1)</td>
</tr>
<tr>
<td>XLNet</td>
<td>N</td>
<td>91.5 (+3.7)</td>
<td>84.1 (+4.9)</td>
<td>89.8 (+5.3)</td>
<td>85.2 (+10.4)</td>
<td>88.2 (+7.7)</td>
</tr>
<tr>
<td></td>
<td>Y</td>
<td>91.6 (+0.1)</td>
<td>84.2 (+0.1)</td>
<td>89.8 (+0.0)</td>
<td>85.4 (+0.2)</td>
<td>88.3 (+0.1)</td>
</tr>
<tr>
<td>ALBERT</td>
<td>N</td>
<td>91.6 (+3.8)</td>
<td>84.0 (+4.8)</td>
<td>90.0 (+5.5)</td>
<td>84.9 (+10.1)</td>
<td>88.5 (+7.5)</td>
</tr>
<tr>
<td></td>
<td>Y</td>
<td>91.6 (+0.0)</td>
<td>84.3 (+0.3)</td>
<td>90.1 (+0.1)</td>
<td>85.1 (+0.2)</td>
<td>88.7 (+0.2)</td>
</tr>
</tbody>
</table>

Table 7

SRL results with different pre-trained language models on CoNLL-2005, CoNLL-2009, and CoNLL-2012 test sets. The results listed in the table are evaluated while given pre-identified predicates. The baseline is our proposed graph-based model because of its good performance. The “SYN” column indicates whether syntax information is employed, and the GCN syntax encoder is adopted for all models which are enhanced with syntax information. In this table, “+” in brackets represent relative improvements using syntax information compared to the syntax-agnostic model, while the “↑” indicates an absolute improvement using a pre-trained language model compared to the pure baseline model without any syntax information or pre-trained language model enhancement.
From the results in the table, all systems using the pre-trained language model have been greatly improved compared with the baseline models, especially in the out-of-domain test set. This indicates that since the pre-trained language model is trained on a very large scale corpus, the performance decrease caused by the domain inconsistency between the training and the test data is reduced, and the generalization ability for the SRL model is enhanced.

For the role of syntax, we found that the improvement from syntax enhancement in the baseline is greater than the improvement that systems gained from using the pre-trained language models. When the ability of pre-trained language models is strengthened, the performance growth brought by syntax gradually declines, and in some very strong language models, syntax even brings no further improvements for SRL. We suspect the reason is that the pre-trained language model can learn some syntactic information implicitly through unsupervised language modeling training (Hewitt and Manning 2019; Clark et al. 2019), thus superimposing explicit syntactic information will not bring as much improvement as it will on the baseline model. We also found that the implicit features for pre-trained language models do not fully maximize syntactic information, as providing explicit syntactic information can still lead to improvements, though this depends on the quality of syntactic information and the ability to extract explicit syntactic information. When explicit syntax is accurate enough and the syntax feature encoder is strong enough, the syntax information can further enhance the accuracy of SRL; however, if these conditions are not satisfied, then syntax is no longer the first option for promoting SRL performance in the neural network era. Besides, due to the uncontrollability of the implicit features of the pre-trained models, syntax as an auxiliary tool for SRL will exist for a long time, and neural SRL models can see even higher accuracy gains by leveraging syntactic information rather than ignoring it until neural networks’ black box is fully revealed.

5.7 Syntactic Contribution

Syntactic information plays an informative role in semantic role labeling; however, few studies have been done to quantitatively evaluate the contribution of syntax to SRL. In dependency SRL, we observe that most of the neural SRL systems compared above used the syntactic parser of Bjørkelund et al. (2010) for syntactic inputs instead of the one from the CoNLL-2009 shared task, which adopted a much weaker syntactic parser. In particular, Marcheggiani and Titov (2017) adopted an external syntactic parser with even higher parsing accuracy. Contrarily, our SRL model is based on the automatically predicted parse with moderate performance provided by the CoNLL-2009 shared task, but we still manage to outperform their models. In span SRL, He et al. (2017) injected syntax as a decoding constraint without having to retrain the model and compared the auto and gold syntax information for SRL. Strubell et al. (2018) presented a neural network model in which syntax is incorporated by training one attention head to attend to syntactic parents for each token and demonstrated that the SRL models benefit from injecting state-of-the-art predicted parses. Because different types of syntax and syntactic parsers are used in different works, the results are not directly comparable. Thus, this motivates us to explore how much syntax contributes to dependency-based SRL in the deep learning framework and how to effectively evaluate the relative performance of syntax-based SRL. To this end, we conduct experiments for empirical analysis with different syntactic inputs.

Syntactic Input. In dependency SRL, four types of syntactic input are used to explore the role of syntax:
1. The automatically predicted parse provided by CoNLL-2009 shared task.
2. The parsing results of the CoNLL-2009 data by state-of-the-art syntactic parser, the Biaffine Parser (Dozat and Manning 2017).
3. Corresponding results from another parser, the BIST Parser (Kiperwasser and Goldberg 2016), which is also adopted by Marcheggiani and Titov (2017).
4. The gold syntax available from the official data set.

Besides, to obtain flexible syntactic inputs for research, we design a faulty syntactic tree generator (referred to as STG hereafter) that is able to produce random errors in the output dependency tree like a real parser does. To simplify implementation, we construct a new syntactic tree based on the gold standard parse tree. Given an input error probability distribution estimated from a true parser output, our algorithm presented in Algorithm 2 stochastically modifies the syntactic heads of nodes on the premise of a valid tree. Notably, the “random error” in the STG parse tree is just a simulation of the actual error of a real parser. It does not fully reflect the actual error of the parsers but is used to roughly explore the effect of the relative accuracy of the syntax on the SRL model. In span SRL, because the data set only provides a golden constituent syntax, we compared the auto syntax from the Choe&Charniak Parser (Choe and Charniak 2016), Kitaev&Klein Parser (Kitaev and Klein 2018), and the gold syntax from the data set for SRL models.

**Algorithm 2** Faulty syntactic tree generator.

**Input:** A gold standard syntactic tree $\text{GT}$, the specific error probability $p$

**Output:** The new generative syntactic tree $\text{NT}$

1. $N$ denotes the number of nodes in $\text{GT}$
2. for each node $n \in \text{GT}$ do
3.  $r = \text{random}(0, 1)$, a random number
4.  if $r < p$ then
5.    $h = \text{random}(0, N)$, a random integer
6.    find the syntactic head $n_h$ of $n$ in $\text{GT}$
7.    modify $n_h = h$, and get a new tree $\text{NT}$
8.  if $\text{NT}$ is a valid tree then
9.    break
10. else
11.    goto step 5
12. end if
13. end if
14. end for
15. return the new generative tree $\text{NT}$

**Evaluation Measure.** For the SRL task, the primary evaluation measure is the semantic labeled $F_1$ score; however, this score is influenced by the quality of syntactic input to some extent, leading to an unfaithful reflection of the competence of a syntax-based SRL system. Namely, this is not the outcome of a true and fair quantitative comparison for these types of SRL models. To normalize the semantic score relative to syntactic
Table 8

Dependency SRL results on the CoNLL 2009 English WSJ test set, in terms of labeled attachment score for syntactic dependencies (LAS), semantic precision (P), semantic recall (R), semantic labeled F$_1$ score (Sem-F$_1$), and the ratio Sem-F$_1$/LAS. A superscript *** indicates LAS results from our personal communication with the authors.

<table>
<thead>
<tr>
<th>System</th>
<th>PLM</th>
<th>LAS</th>
<th>P</th>
<th>R</th>
<th>Sem-F$_1$</th>
<th>Sem-F$_1$/LAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Zhao et al. 2009b)</td>
<td></td>
<td>86.0</td>
<td>-</td>
<td>-</td>
<td>85.4</td>
<td>99.30</td>
</tr>
<tr>
<td>(Zhao et al. 2009a)</td>
<td></td>
<td>89.2</td>
<td>-</td>
<td>-</td>
<td>86.2</td>
<td>96.64</td>
</tr>
<tr>
<td>(Björkelund et al. 2010)</td>
<td></td>
<td>89.8</td>
<td>87.1</td>
<td>84.5</td>
<td>85.8</td>
<td>95.55</td>
</tr>
<tr>
<td>(Lei et al. 2015)</td>
<td></td>
<td>90.4</td>
<td>-</td>
<td>-</td>
<td>86.4</td>
<td>95.80</td>
</tr>
<tr>
<td>(FitzGerald et al. 2015)</td>
<td></td>
<td>90.4</td>
<td>-</td>
<td>-</td>
<td>86.7</td>
<td>95.90</td>
</tr>
<tr>
<td>(Roth and Lapata 2016)</td>
<td></td>
<td>89.8</td>
<td>88.1</td>
<td>85.3</td>
<td>86.7</td>
<td>96.5</td>
</tr>
<tr>
<td>(Marcheggiani and Titov 2017)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>90.34*</td>
<td>97.41</td>
</tr>
<tr>
<td>Sequence-based + K-order hard pruning</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CoNLL09 predicted</td>
<td>+E</td>
<td>86.0</td>
<td>89.7</td>
<td>89.3</td>
<td>89.5</td>
<td>104.07</td>
</tr>
<tr>
<td>STG Auto syntax</td>
<td>+E</td>
<td>90.0</td>
<td>90.5</td>
<td>89.3</td>
<td>89.9</td>
<td>99.89</td>
</tr>
<tr>
<td>Gold syntax</td>
<td>+E</td>
<td>100.0</td>
<td>91.0</td>
<td>89.7</td>
<td>90.3</td>
<td>90.30</td>
</tr>
<tr>
<td>Sequence-based + Syntax</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CoNLL09 predicted</td>
<td>+E</td>
<td>86.0</td>
<td>90.5</td>
<td>88.5</td>
<td>89.5</td>
<td>104.07</td>
</tr>
<tr>
<td>Biaffine parser</td>
<td>+E</td>
<td>90.22</td>
<td>90.3</td>
<td>89.3</td>
<td>89.8</td>
<td>99.53</td>
</tr>
<tr>
<td>BIST Parser</td>
<td>+E</td>
<td>90.05</td>
<td>90.3</td>
<td>89.1</td>
<td>89.7</td>
<td>99.61</td>
</tr>
<tr>
<td>Gold syntax</td>
<td>+E</td>
<td>100.0</td>
<td>91.0</td>
<td>90.0</td>
<td>90.5</td>
<td>90.50</td>
</tr>
</tbody>
</table>

The benefits of this measure are two-fold: It quantitatively evaluates syntactic contribution to SRL and impartially estimates the true performance of SRL independent of the performance of the input syntactic parser. In addition, we further extended this evaluation metric for span SRL and used the ratio of Sem-F$_1$ and constituent syntax F$_1$ score (Syn-F$_1$) to measure the pure contribution of the proposed SRL model to clearly show the source of SRL performance improvement from the model’s contribution rather than the improvement due to syntax accuracy. It is worth noting that the ratio of Sem-F1/LAS or Sem-F1/Syn-F1 can only be used as a rough estimate under regular circumstances; that is, it can be used in SRL systems that use sufficiently accurate syntax parses, but if the system uses low-quality syntactic trees with extremely low LAS or Syn-F1, there will be a numerical explosion that renders the value becomes meaningless.

Table 8 reports the dependency SRL performance of existing models in terms of Sem-F$_1$/LAS ratio on the CoNLL-2009 English test set. Interestingly, even though our system has significantly lower scores than others by 3.8% LAS in syntactic components, we obtain the highest results on both Sem-F$_1$ and Sem-F$_1$/LAS ratio. These results show that our SRL component is relatively much stronger. Moreover, the ratio comparison in Table 8 also shows that since the CoNLL-2009 shared task, most SRL works actually benefit from the enhanced syntactic component rather than the improved SRL component itself. No post-CoNLL SRL systems, neither traditional nor neural types, exceeded the top systems of the CoNLL-2009 shared task (Zhao et al. 2009b [SRL-only track using the provided predicted syntax] and Zhao et al. 2009a [Joint track using self-

---

7 The idea of the ratio score in Surdeanu et al. (2008) actually was from an author of this paper, Hai Zhao; this was indicated in the acknowledgment part of Surdeanu et al. (2008).

8 Note that several SRL systems that do not provide syntactic information are not listed in the table.
developed parser]. We believe that this work, for the first time, reports both a higher Sem-F$_1$ and a higher Sem-F$_1$/LAS ratio since the CoNLL-2009 shared task. We also presented the comprehensive results of our sequence-based SRL model with a syntax encoder instead of pruning on the aforementioned syntactic inputs of different quality and compare these with previous SRL models. A number of observations can be made from these results. First, the model with the GCN syntax encoder gives quite stable SRL performance no matter the syntactic input quality, which varies in a broad range, and it obtains overall higher scores compared to the previous states-of-the-art. Second, it is interesting to note that the Sem-F$_1$/LAS score of our model becomes relatively smaller as the syntactic input becomes better. Not surprisingly though, these results show that our SRL component is relatively even stronger. Third, when we adopt a syntactic parser with higher parsing accuracy, our SRL system achieves a better performance. Notably, our model yields a Sem-F$_1$ of 90.53% taking gold syntax as input. This suggests that high-quality syntactic parse may indeed enhance SRL, which is consistent with the conclusion in He et al. (2017).

We further evaluated the syntactic contribution in span SRL, as shown in Table 9. For Wang et al. (2019) and our results, when syntax and pre-trained language model were kept the same, our model obtained better Sem-F$_1$, which is also reflected in the Sem-F$_1$/Syn-F$_1$ ratio, indicating that our model has stronger syntactic utilization ability. In addition, by comparing the results of whether a pre-trained language model is used, it can be found that those using the pre-trained language model have a higher ratio of Sem-F$_1$/Syn-F$_1$, which shows that the features offered by the pre-trained language model potentially increase the syntactic information, resulting in a higher ratio of syntax contribution.

Additionally, to show how SRL performance varies with syntax accuracy, we also test our sequence-based dependency SRL model with k-order hard pruning in first and tenth orders using different erroneous syntactic inputs generated from STG and evaluate their performance using the Sem-F$_1$/LAS ratio. Figure 12 shows Sem-F$_1$ scores at different qualities of syntactic parse inputs on the English test set, whose LAS varies from 85% to 100%, compared to previous states-of-the-art (Marcheggiani and Titov 2017). Our tenth-order pruning model gives quite stable SRL performance no matter the syntactic input quality, which varies in a broad range, while our first-order pruning model yields overall lower results (1-5% F$_1$ drop), owing to missing too many true

### Table 9

<table>
<thead>
<tr>
<th>System</th>
<th>PLM</th>
<th>Syn-F$_1$</th>
<th>P</th>
<th>R</th>
<th>Sem-F$_1$</th>
<th>Sem-F$_1$/Syn-F$_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(He et al. 2017)</td>
<td>Choe&amp;Charniak Parser</td>
<td>93.8</td>
<td>–</td>
<td>–</td>
<td>84.8</td>
<td>90.41</td>
</tr>
<tr>
<td></td>
<td>Choe&amp;Charniak Parser +E</td>
<td>93.8</td>
<td>–</td>
<td>–</td>
<td>85.6</td>
<td>91.26</td>
</tr>
<tr>
<td></td>
<td>Gold syntax</td>
<td>100.0</td>
<td>–</td>
<td>–</td>
<td>87.0</td>
<td>87.00</td>
</tr>
<tr>
<td>(Wang et al. 2019)</td>
<td>Kitaev&amp;Klein Parser</td>
<td>95.4</td>
<td>–</td>
<td>–</td>
<td>88.2</td>
<td>92.45</td>
</tr>
<tr>
<td></td>
<td>Kitaev&amp;Klein Parser +E</td>
<td>95.4</td>
<td>–</td>
<td>–</td>
<td>89.1</td>
<td>92.20</td>
</tr>
<tr>
<td></td>
<td>Gold syntax</td>
<td>100.0</td>
<td>–</td>
<td>–</td>
<td>92.2</td>
<td>92.20</td>
</tr>
<tr>
<td>(Marcheggiani and Titov 2020)</td>
<td>Kitaev&amp;Klein Parser</td>
<td>95.4</td>
<td>85.4</td>
<td>88.1</td>
<td>92.6</td>
<td>92.60</td>
</tr>
<tr>
<td>Sequence-based + Syntax GCN encoder</td>
<td>Choe&amp;Charniak Parser</td>
<td>93.8</td>
<td>86.4</td>
<td>84.8</td>
<td>85.6</td>
<td>91.26</td>
</tr>
<tr>
<td></td>
<td>Choe&amp;Charniak Parser +E</td>
<td>93.8</td>
<td>87.2</td>
<td>86.8</td>
<td>87.0</td>
<td>92.75</td>
</tr>
<tr>
<td></td>
<td>Kitaev&amp;Klein Parser +E</td>
<td>95.4</td>
<td>88.3</td>
<td>89.1</td>
<td>88.5</td>
<td>92.77</td>
</tr>
<tr>
<td></td>
<td>Gold syntax</td>
<td>100.0</td>
<td>93.2</td>
<td>92.4</td>
<td>92.6</td>
<td>92.60</td>
</tr>
</tbody>
</table>
arguments. These results show that high-quality syntactic parses may indeed enhance dependency SRL. Furthermore, they indicate that our model with syntactic input as accurate as Marcheggiani and Titov (2017), namely, 90% LAS, will give a Sem-F1 exceeding 90%.

5.8 Improvement Source for Syntactic Pruning

In this paper, we explore two benefits of using syntax information in SRL. The first is the proximal distribution of arguments in relation to predicates in dependency trees, which allows us to prune unlikely argument candidates. We hypothesize that the pruning process is so effective because of the imbalanced distribution of semantic role labels, which is compounded by the fact that non-arguments are usually used as special arguments in modeling. Pruning primarily eliminates obvious non-arguments and hence reduces this label imbalance to a certain degree. To verify this hypothesis, we compare the F1 scores for argument labeling using our tree-based models with and without the SynRule Soft Pruning. We chose the typical argument labels: A0, A1, A2, and AM-* (such as AM-TMP, AM-LOC) to count their respective frequencies and F1 according to nominal and verbal predicates.

The quantitative results are shown in Table 10. From the label frequency, we found that the distribution of role labels was uneven. Due to the introduction of the pruning approach, the gain on the low-frequency label is higher than that on high-frequency ones, which suggests that the pruning method reduced the imbalance in the distribution of role labels, which is consistent with our hypothesis.

5.9 Improvement Source of Syntactic Feature

For the second motivation in improving SRL in this paper, we assume that because there are mirrored arcs\(^9\) in the dependency tree and the semantic dependency graph,

\(^9\) When a dependency edge \(i \rightarrow j\) exists in both the syntactic tree and the semantic graph, the semantic dependency arc \(i \rightarrow j\) is referred to as a mirror arc.
Table 10
The frequency and P / R / F₁ scores for typical argument role labels on the CoNLL-2009 English test set. Baseline refers to the Tree-based model, and +Pruning refers to the SynRule Soft Pruning method. Due to the width constraint, we omit the Precision (P) and Recall (R) scores of the nominal pred.

<table>
<thead>
<tr>
<th>Role</th>
<th>FREQ</th>
<th>Baseline</th>
<th>+Pruning</th>
<th>FREQ</th>
<th>Baseline</th>
<th>+Pruning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>P</td>
<td>R</td>
<td>F</td>
<td></td>
<td>P</td>
</tr>
<tr>
<td>A0</td>
<td>15%</td>
<td>93.1</td>
<td>91.9</td>
<td>92.5</td>
<td>93.2</td>
<td>92.3</td>
</tr>
<tr>
<td>A1</td>
<td>21%</td>
<td>93.9</td>
<td>93.1</td>
<td>93.5</td>
<td>93.6</td>
<td>93.4</td>
</tr>
<tr>
<td>A2</td>
<td>5%</td>
<td>84.3</td>
<td>81.8</td>
<td>83.0</td>
<td>85.1</td>
<td>83.1</td>
</tr>
<tr>
<td>AM-*</td>
<td>16%</td>
<td>82.2</td>
<td>80.2</td>
<td>81.2</td>
<td>82.4</td>
<td>80.6</td>
</tr>
</tbody>
</table>

Table 11
The accuracy on mirror arcs in the dependency trees and semantic dependency graphs predicted using the CoNLL-2009 English test set.

<table>
<thead>
<tr>
<th>Sequence-based</th>
<th>Tree-based</th>
<th>Graph-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>88.4%</td>
<td>88.7%</td>
</tr>
<tr>
<td>+Syntax GCN encoder</td>
<td>89.7%</td>
<td>89.9%</td>
</tr>
</tbody>
</table>

The use of syntax would increase the accuracy of role labeling compared to that of the baseline model. In order to verify this hypothesis, on the CoNLL-2009 English test set, we computed additional statistics on the predictions of three baselines and these models with the GCN Syntax encoder. In these statistics, we compute the ratio of the number of correctly predicted mirrored arcs to the total number of mirror arcs and exclude arc direction to focus solely on how syntax aids in the predictions on mirror arcs.

The statistical results are presented in Table 11. Comparing the correct ratio of the mirror arcs in the prediction between the baseline and the baseline with an added syntactic encoder, we found that the syntactic information consistently improved these mirrored semantic dependency arcs on the three baselines, which fits our hypothesis. Because these mirrored structures are greatly enhanced in a number of syntactic enhancement models, we can conclude that the GCN encoder effectively encodes syntactic information, and that this encoded syntactic feature is useful for improving the mirrored semantic structures. It is also worth noting that our exploration of the source of the improvement of syntax for SRL does not mean that the syntax has only this effect. Actually, there may be many other factors, which deserve more exploration in the future.

5.10 Effects of Better Syntax on SRL when using Pre-trained Language Models

The results shown in the previous experiments show that with the addition of strong pre-trained language models, the use of syntax information loses its dominance; however, there are two factors that affect the final performance enhancement of syntax on SRL: one is the quality of the syntax parse, and the other is the method of integration. The previous experiments mainly discussed effective integration and corresponding effects but did not pay enough attention to the quality of the syntax. Therefore, in this section, we study how varying the quality of syntactic inputs impacts SRL systems with pre-trained language models. In addition, we use the HPSG Parser, which was trained
Table 12
The effect of different quality syntactic parse trees on the graph-based SRL model with the enhancement of pre-trained language models. The results are evaluated on the CoNLL-2009 English test set. Notably, the Biaffine Parser was trained by us. Due to the difference in implementation, the parsing performance listed is lower than that of Dozat and Manning (2017).

<table>
<thead>
<tr>
<th>Parser</th>
<th>w/ ELMo</th>
<th>w/ BERT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LAS</td>
<td>P</td>
</tr>
<tr>
<td>N/A</td>
<td>–</td>
<td>89.6</td>
</tr>
<tr>
<td>CoNLL09 predicted</td>
<td>86.00</td>
<td>91.0</td>
</tr>
<tr>
<td>Biaffine Parser</td>
<td>90.22</td>
<td>90.5</td>
</tr>
<tr>
<td>HPSG Parser</td>
<td>94.34</td>
<td>91.7</td>
</tr>
</tbody>
</table>

on the golden syntactic annotations of CoNLL-2009, to provide better quality real parse trees in this experiment. The HPSG parser is a state-of-the-art parser proposed by Zhou and Zhao (2019) and was based on a head-driven phrase structure grammar (HPSG) and the XLNet pre-trained language model. For this experiment, the base model is the graph-based model with the syntax GCN encoder, and the evaluation results are listed in Table 12.

The results show that on one hand, the pre-trained language model greatly improves the performance of SRL, but on the other hand, even with the pre-trained language model, a stronger syntactic parser can still bring performance gains. Using poor quality syntax, however, is unlikely to improve performance and can even have a negative impact on a strong enough SRL baseline.

5.11 Role of Other Syntactic Information

In addition to the syntax tree, part-of-speech tags and lemmas are also viewed as a kind of syntactic knowledge, and they can affect SRL. Since these are typically only used as additional features to enhance representation (which is not the main focus of this work), we conduct a simple experimental exploration on the role of POS tags and lemmas in this section. We picked the sequence-based model\(^\text{10}\) with a syntax GCN encoder, conducted experiments on the CoNLL-2009 English data set, and used two pre-trained language models, ELMo and BERT, to investigate their roles under different conditions.

The experimental results are shown in Table 13. From the comparison of the experimental results, we found that with the help of ELMo, both POS tags and lemmas show enhancement effects. POS tags had a greater improvement, which may be because lemmatization produces a more shallow syntactic information. When switching the pre-trained language model from ELMo to the stronger BERT, we found that lemmas almost provide no additional improvement, and the usefulness of POS tags is also very limited. This shows that stronger pre-trained language models may cover shallow syntactic information, and stacking such information will therefore not have additional enhancement effects.

\(^\text{10}\) The reason for choosing sequence-based model instead of graph-based model as in the previous experiment is that POS and Lemma have less obvious influence on graph-based models.
Table 13
Performance comparison of POS tag and lemma on the CoNLL-2009 English test set. Full Model refers to the sequence-based model with syntax GCN encoder, which leverages POS tag, lemma, syntax tree.

<table>
<thead>
<tr>
<th>System</th>
<th>w/ ELMo</th>
<th>w/ BERT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P</td>
<td>R</td>
</tr>
<tr>
<td>Full Model</td>
<td>90.3</td>
<td>89.3</td>
</tr>
<tr>
<td>-POS</td>
<td>89.8</td>
<td>89.2</td>
</tr>
<tr>
<td>-Lemma</td>
<td>90.1</td>
<td>89.1</td>
</tr>
</tbody>
</table>

6. Related Work

The origins of semantic role labeling can date back several decades to when Fillmore (1968) first theorized the existence of deep semantic relations between the predicate and other sentential constituents. Over several years, various linguistic formalisms and their related predicate-argument structure inventories have extended Fillmore’s seminal intuition (Dowty 1991; Levin 1993). Gildea and Jurafsky (2000) pioneered the task of semantic role labeling as a shallow semantic parsing. In dependency SRL, most traditional SRL models rely heavily on feature templates (Pradhan et al. 2005; Zhao, Chen, and Kit 2009; Björkelund, Hafdell, and Nugues 2009). Among them, Pradhan et al. (2005) combined features derived from different syntactic parses based on SVM classifier, while Zhao, Chen, and Kit (2009) and Zhao, Zhang, and Kit (2013) presented an integrative approach for dependency SRL via a greedy feature selection algorithm. Later, Collobert et al. (2011) proposed a convolutional neural network model that induced word embeddings rather than relied on handcrafted features, which was a breakthrough for the SRL task.

Foland and Martin (2015) presented a dependency semantic role labeler using convolutional and time-domain neural networks, while FitzGerald et al. (2015) exploited neural networks to jointly embed arguments and semantic roles, akin to the work (Lei et al. 2015) that induced a compact feature representation by applying a tensor-based approach. Recently, researchers have considered multiple ways to effectively integrate syntax into SRL learning. Roth and Lapata (2016) introduced dependency path embedding to model syntactic information and exhibited notable success. Marcheggiani and Titov (2017) leveraged the graph convolutional network to incorporate syntax into neural models. Differently, Marcheggiani, Frolov, and Titov (2017) proposed a syntax-agnostic for dependency SRL that used effective word representation, which for the first time achieved performance comparable to state-of-the-art syntax-aware SRL models.

Most neural SRL works, however, often pay less attention to the impact of input syntactic quality on SRL performance since they usually only use a fixed syntactic quality input. This work is thus more than proposing a high performance SRL model through reviewing the highlights of previous models; it also presents an effective syntax tree-based method for argument pruning. Our work is also closely related to Punyakanok, Roth, and Yih (2008). Under traditional methods, Punyakanok, Roth, and Yih (2008) investigated the significance of syntax to SRL systems and showed syntactic information was most crucial in the pruning stage. There are two important differences between Punyakanok, Roth, and Yih (2008) and our work. First, in our paper, we summarize the current dependency and span SRL and consider them under multiple baseline models.
and syntax integration approaches to reduce deviations resulting from model structure and syntax integration approach. Second, the development of pre-trained language models has dramatically changed the basis of SRL models, which motivated us to revisit the role of syntax based on new situations.

As researchers have constantly explored new approaches to further improve SRL, the exploitation of syntactic features has emerged as a natural option. Kasai et al. (2019) extracted the supertags from dependency parses as additional feature to enhance the SRL model. Cai and Lapata (2019b) presented a system that jointly trained with two syntactic auxiliary tasks: predicting the dependency label of a word and predicting there exists an arc linking said word to the predicate. They also suggested that syntax could help SRL because a significant portion of the predicate-argument relations in the semantic dependency graph mirror the arcs that appear in the syntactic dependency graph, and there is often a deterministic mapping between syntactic and semantic roles. Compared with our research, these works only focus on using syntax with a specific model to improve SRL, whereas instead we performed more systematic and comprehensive research.

In the other span SRL research lines, Moschitti, Pighin, and Basili (2008) applied tree kernels as encoders to extract constituency tree features for SRL, while Naradowsky, Riedel, and Smith (2012) used graphical models to model the tree structures. Socher et al. (2013) and Tai, Socher, and Manning (2015) proposed recursive neural networks for incorporating syntax information in SRL that recursively encoded constituency trees to constituent representations. He et al. (2017) presented an extensive error analysis with deep learning models for span SRL and included a discussion of how constituent syntactic parsers could be used to improve SRL performance. With the recent advent of self-attention, syntax can be used not only for pruning or encoding to provide auxiliary features, but also for guiding the structural learning of models. Strubell et al. (2018) modified the dependency tree structure to train one attention head to attend to syntactic parents for each token. In addition, compared with the application of dependency syntax, the difficult step is mapping the constituent features back to the word level. Wang et al. (2019) extended the syntax linearization approaches of Gómez-Rodríguez and Vilares (2018) and incorporated this information as a word-level feature in a SRL model; Marcheggiani and Titov (2020) introduced a novel neural architecture, SpanGCN, for encoding constituency syntax at the word level.

7. Conclusion

This paper explores the role of syntax for the semantic role labeling task. We presented a systematic survey based on our recent works on SRL and a recently popular pre-trained language modeling. Through experiments on both the dependency and span formalisms, and the sequence-based, tree-based and graph-based modeling approaches, we conclude that although the effects of syntax on SRL seem like a never-ending topic of research, with the help of current pre-trained language models, the syntax improvement provided to SRL model performance seems to be gradually reaching its upper limit. Beyond presenting approaches that lead to improved SRL performances, we performed a detailed and fair experimental comparison between span and dependency SRL formalisms to show which is more fit for machine learning. In addition, we have studied a variety of methods of syntax integration and have shown that there is unaccclimation for the hard pruning in the deep learning model which is very popular in the pre-NN era.
For the exact role of syntax in serving the state-of-the-art SRL models, we have a cautious conclusion: Syntax may still provide downstream task enhancement in the time of deep learning (even with the time of powerful pre-trained language models); however, its returns are diminishing. As we can see from our experimental results, syntactic clues contribute less and less when used with more and more powerful baseline SRL models, and we saw that for our best settings, absolute performance improvement may be just around 0.5%. This is a vast difference from pre-NN times, when the inclusion of syntax could bring an absolute performance improvement of as high as 5% to 10%. Syntax, however, may have alternative methods of integration that do not consist of simply including a pre-trained syntax parser. Some works have demonstrated that the current deep learning models in NLP naturally incorporate syntax using neural models’ powerful representation learning. Actually, our recent work (Zhou, Li, and Zhao 2020) showed that syntactic parsers and semantic parsers can help each other, which suggests that in general, neural NLP models, including SRL models, may comprehensively and latently learn linguistic knowledge that covers both syntax and semantics. This may effectively explain why neural SRL models do not rely on syntactic clues as much as their non-NN SRL counterparts. When using even more powerful pre-trained language models for enhancement, we indeed saw even less SRL performance improvement, which is also explainable considering that pre-trained language models have been known to contain syntax in specific hidden layers, as shown in Hewitt and Manning (2019) and Clark et al. (2019). When pre-trained language models offered a strong syntactic signal, including syntax information from a third-party pre-trained syntactic parser will certainly lead to weaker improvement, as the added syntax information is somewhat redundant.

Overall, we summarize that there is no certainty that specific syntactic information may surely benefit specific deep SRL models, when the latter may more or less automatically capture syntax from its own task-specific learning. Syntax may still be helpful for deep models in a general way, however, the significance which makes such a help depends on the way how to integrate the syntax, but does not the type of models or syntax.

Syntax is an amazing discovery from theoretical linguistics, as it is a complete linguistic concept, but it is not really an observable linguistic phenomena. As Chomsky (1965) addressed, though it was believed that there exists a universal grammar in human brain according to his and later inspired research, as to our best knowledge, there is not clear enough evidence from cognitive or electrophysiological outcomes to confirm the existence of such syntax or grammar represented by some kind of human brain structure or cognitive mechanism. While lexical (word) linguistics can be identified using writing control (boundaries between words), and semantics can be noted by mapping words and their corresponding entities in the real world, syntax cannot be experienced like either of the above way. It is well known that native speakers do not really speak following some perceptive “syntax”, yet linguists believe syntax is always there. We speculate that if this is the way humans use syntax when processing language, then in the future, this may also be how neural NLP models adopt syntax.

Acknowledgments
We thank Kevin Parnow (parnow@sjtu.edu.cn), from the Department of Computer Science and Engineering, Shanghai Jiao Tong University for his kind help in proofreading when we were working on this paper. Many thanks to the editor and anonymous reviewers for their valuable suggestions and comments.

References
Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and


Intelligence, EAAI 2021, Virtual Event, pages 12803–12811, AAAI Press.
Kasai, Jungo, Dan Friedman, Robert Frank, Dragomir Radev, and Owen Rambow.


Marcheggiani, Diego, Anton Frolov, and Ivan Titov. 2017. A simple and accurate syntax-agnostic neural model for


Punyakanok, Vasin, Dan Roth, and Wen-tau Yih. 2005. The necessity of syntactic parsing for semantic role labeling. In


