Event Argument Identification on Dependency Graphs with Bidirectional LSTMs

Alex Judea and **Michael Strube**
Heidelberg Institute for Theoretical Studies gGmbH
Schloss-Wolfsbrunnenweg 35
69118 Heidelberg, Germany
(alex.judea|michael.strube)@h-its.org

Abstract

In this paper we investigate the performance of event argument identification. We show that the performance is tied to syntactic complexity. Based on this finding, we propose a novel and effective system for event argument identification. Recurrent Neural Networks learn to produce meaningful representations of long and short dependency paths. Convolutional Neural Networks learn to decompose the lexical context of argument candidates. They are combined into a simple system which outperforms a feature-based, state-of-the-art event argument identifier without any manual feature engineering.

1 Introduction

Event extraction is a difficult information extraction task. The 2005 Automatic Content Extraction evaluation (ACE 2005) defines three challenging sub-tasks: *Entity mention detection*, the task of finding mentions of predefined entity types like persons and organizations; *event trigger detection*, the task of finding words, mostly verbs or nominalizations, indicating an event from a set of predefined event types; and *event argument identification*, the identification of entity mentions[1] playing a role in the events, as well as the identification of the roles they play.

When we look at the evaluations in three of the most influential recent event extraction papers (Li et al., 2013, 2014; Chen et al., 2015; Nguyen et al., 2016) we note that argument identification performance is low, ranging from 52.7 to 55.4 F_1. There are multiple reasons for the low performance. First, argument identification suffers from error propagation. Missed or spurious event triggers or entity mentions may lead to missed or spurious event arguments. Second, event structure is complex. Multiple entities can play the same role in the same event. Additionally, one entity can play different roles across events (and thus cause multiple event arguments). Consider the following example.

> A Palestinian **boy** as well as his **brother** and a **sister** were *wounded* late Wednesday by Israeli *gunfire*.

Here, the three entity mentions (in bold) are all `Victims` of the INJURE event triggered by 'wounded' as well as `Targets` of the ATTACK event triggered by 'gunfire'. Such structures can become even more complex when more events and more entities are involved.

The third reason for low argument identification performance is syntactic complexity. Many arguments are syntactically far away from their triggers, making it hard to construct meaningful syntactic features. Section 3 shows that performance is tied to the length of the shortest dependency path connecting trigger and argument.

Error sources one and two were already targeted by systems which jointly infer triggers and their arguments. To the best of our knowledge, no previous work identified syntactic complexity as the third key problem for argument identification performance, and no previous system aimed to decompose syntactic structure in order to learn better classifiers for the task. The contributions of this paper are the following.

1. We observe that syntactic complexity is a crucial factor for argument identification. Argument identification performance highly correlates with dependency path length (Section 3.1).

[1] In this work we make no distinction between 'entity', 'time', and 'place' for the sake of simplicity.

Proceedings of the The 8th International Joint Conference on Natural Language Processing, pages 822–831,
Taipei, Taiwan, November 27 – December 1, 2017

2. We propose to represent dependency paths with bidirectional Long Short-term Memory networks (biLSTMs) in order to account for their sequential and compositional nature. Using LSTMs to learn dependency path representations proved effective in other areas like relation extraction (Xu et al., 2015) and semantic role labeling (Roth and Lapata, 2016). We investigate their use for argument identification.

3. We propose to represent lexical contexts of event arguments with Convolutional Neural Networks. Together with LSTMs, they form an effective and simple argument identifier which beats a state-of-the-art, feature-based system without any manual feature engineering, especially for long dependency paths.

2 Baseline

2.1 Baseline Argument Extraction

Our baseline is a re-implementation of Li et al. (2013). It is a state-of-the-art event extractor that predicts event triggers and arguments jointly. Because of this joint inference, it avoids error propagation and can draw features based on joint event extraction decisions, e.g., how many arguments of a specific type does a specific trigger have?

The system uses a structured perceptron with beam search. It processes a sentence from left to right and token by token. With each position it advances, it constructs new hypotheses containing event trigger and argument assignments. Then, it prunes the hypotheses spaces to the n best alternatives and processes the next token position. After the last token was processed, the hypothesis with highest score is selected as the final prediction. This hypothesis contains trigger and argument assignments for the entire sentence.

2.2 Baseline Argument Extraction Features

Our baseline system uses a rich, hand-engineered feature set. Feature templates can be divided into *local templates* and *global templates*. Local templates characterize single arguments, and they involve only the mentions and triggers of this argument. They capture, e.g., the trigger and entity types, the mention context, and the dependency path between trigger and mention.

Global templates on the other hand characterize multiple arguments, either in terms of shared mentions, or in terms of shared roles. Global templates can be divided into *segment level templates* and *sentence level templates*. Segment level templates capture characteristics of the mentions within one event, e.g., the words between two mentions sharing a role in one event, or the head and modifier of nominal modifications like 'IBM CEO'. Sentence level templates capture characteristics of events sharing mentions, e.g., the roles such a mention fills, or the dependency path connecting the two triggers. The system uses two dozen feature templates, resulting in 150,000 features for argument identification.

Argument type	$\text{Support}_{\text{train}}$	F1_{dev}
`Victim`	578	79.0
`Instrument`	256	77.1
`Artifact`	605	75.6
`Attacker`	574	47.3
`Target`	438	42.6
`Giver`	94	32.9

Table 1: Training set support and development set baseline F_1 for the three best and three worst performing argument types.

3 Performance Analysis

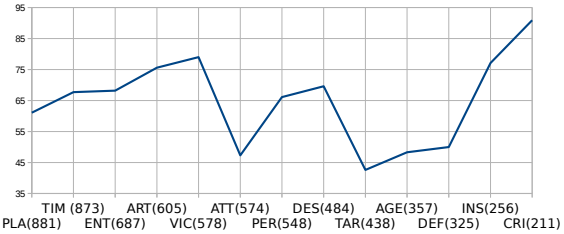
3.1 Analysis of Baseline Performance

We start the analysis of argument identification performance with the observation that despite the low overall performance, some argument types perform reasonably well. Table 1 reports development set precision, recall, and F_1 of our baseline for the three best and the three worst performing argument types in the development set.[2] As we can see, the difference between the best type (`Victim`) and the worst type (`Giver`) is 46.1 F_1 points. What is the reason for this big difference?

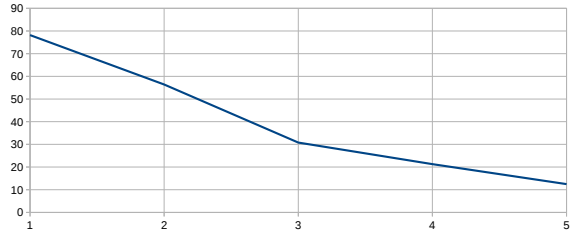
Our first assumption is that the best performing types have more training samples. Indeed, `Victim` has considerably more samples than `Giver`. `Attacker` however has nearly the same amount of training samples but a much lower performance (-31.7 F_1). `Instrument` on the other hand has only half the training samples of `Attacker`, but a better performance (+29.8 F_1).

To further investigate this, Figure 1a plots training set support in decreasing magnitude against development set F_1 for the 12 most frequent argument types. The plot is not conclusive: More

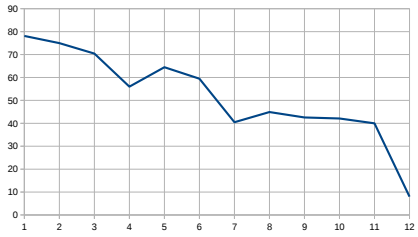
[2] We excluded types with less than 20 samples in the development set.



(a) Training set support in decreasing magnitude plotted against baseline devset F_1.



(b) Increasing dependency path length plotted against baseline devset F_1.



(c) Increasing lexical distance plotted against baseline devset F_1.

Figure 1: Training set support, lexical distance of trigger and argument, and dependency path length plotted against development set performance.

training data does not automatically lead to better performance. The most frequent argument type `Place` with 881 training samples has an F_1 of 61.1, whereas `Victim` with 34% less training samples has an F_1 of 79.0. `Instrument` has about 70% less training samples and an F_1 of 77.1. If the number of training samples is not an important factor for performance, what else could be?

One important factor is semantic variety: Some roles can only be filled by one or two entity types, and most of their mentions *are* role fillers. This is especially true for `Instrument` which can only be filled by vehicles and weapons in ACE 2005; in turn, most weapons are `Instruments`. This is reflected in the good performance of `Instrument` in Table 1 and Figure 1a.

However, most roles can be filled by more than two entity types, and their potential fillers are more frequent than vehicles and weapons: `Entity` for example can be filled by persons, organizations, and geopolitical entities. At the same time, most of the respective mentions are *not* `Entities`. Even if a role can only be filled by one entity type, it may be that most occurrences are not role fillers, making the task to correctly fill those roles harder. Consider `Time` for example, which can only be filled by *time* mentions, yet it has only a mediocre performance of 67.7 F_1 points on the development set. Semantic variety alone cannot explain the big performance differences between argument types.

Another important factor is syntactic complexity: How long and diverse are dependency paths connecting arguments and triggers? To investigate the effect of syntactic complexity, Figure 1b depicts length of dependency paths connecting triggers and arguments in decreasing magnitude against development set F_1. In this plot, we see a much clearer trend: Shorter syntactic distance leads to better performance. Length-1 paths (direct trigger-argument dependency) have an F_1 of 78.2. Length-2 path F_1 drops to 56.4, and to 30.8 for length-3 paths. Length-4 and length-5 paths have an F_1 of 21.3 and 12.5, respectively.

This trend is also reflected in the performance of individual argument types. The most frequent type `Place` has a high average path length of 2.2 and a low F_1 of 61.1. `Victim` on the other hand has considerably less training data, but an average path length of 1.5 and an F_1 of 79.0. For the three best performing types, the average path length is 1.7 vs. 2.3 for the three worst performing types.

Dependency path length is related to lexical distance – the longer a dependency path, the more words are usually between trigger and argument. To investigate the effect of lexical distance, Figure 1c depicts the number of words between trigger and argument against development set performance. Here, we see a somewhat less clearer trend: Increasing lexical distance leads to lower performance; however, with a

considerable increase between distances 4 and 5, and a performance plateau between 8 and 11. Word sequences are much more diverse than dependency paths. A dependency path like returning $\xrightarrow{nmod:from}$ summit $\xrightarrow{nmod:in}$ Ireland abstracts from actual word sequences and ignores many words which are less relevant for argument identification, like adjectives and adverbs. This in turn alleviates data sparsity.

Syntactic complexity is a crucial factor for argument identification, both in terms of overall performance as well as and in terms of individual argument type performance. Therefore, it is inevitable to reduce or better handle syntactic complexity. Most systems incorporate dependency paths merely as strings, or rely on direct dependencies of triggers and arguments. They do not decompose or further analyze dependency paths in order to find relevant substructures, or to deal with data sparsity of long paths. In Section 4, we present a simple but efficient system which directly addresses syntactic complexity.

3.2 Dependency Paths

We now illustrate the benefits and difficulties of using dependency paths for argument identification. Our dependency paths are lexicalized; they always start with the trigger word and end with a mention word.[3] We say that a path has length 1 if trigger and argument are directly related, length 2 if the path includes one intermediate dependency, etc.

Often, short dependency paths directly reflect event argument structure:

killed $\xrightarrow{\text{nsubj}}$ father-in-law
killed $\xrightarrow{\text{dobj}}$ him

The trigger ('killed', a DIE event) has two dependencies, 'father-in-law' being the subject and 'him' being the object. Even without looking at more context we can say with confidence that the subject must be the `Agent` of the event and the object must be the `Victim`. Even longer paths may be quite clear:

returning $\xrightarrow{nmod:from}$ summit $\xrightarrow{nmod:in}$ Ireland

Here, 'returning' indicates a TRANSPORT event. The path conveys the information that some entity returns from a summit in Ireland, making 'Ireland' the `Origin` of the event.

Of course, not all dependency paths are as easy to interpret. The following examples show the necessity to decompose dependency paths in order to catch similarities between them.

war $\xleftarrow{dobj}$ fight $\xrightarrow{nsubj}$ U.S.
war $\xleftarrow{nmod:to}$ go $\xrightarrow{nsubj}$ we

These paths are more complex than previous ones because trigger and argument are governed by other words, namely by 'fight' and 'go'. In both cases, 'war' triggers an ATTACK event and the subject is an `Attacker` argument. Humans can easily spot similarities in the two paths. The arguments are in both cases the subjects of the governing verbs: 'U.S.' is the entity fighting a war and 'we' is the entity going to war. However, the left sides of the paths look quite different: In one case, 'war' is the direct object of the governing verb, in the other it is the nominal modifier. Additionally, the governing verbs do not share meaning. In order to catch the similarities, a system needs the ability to decompose the paths and to learn the meaning of sequences of words and dependencies.

4 Approach

4.1 Problem Encoding

We cast argument identification as a classification task. For a trigger-mention pair (t, m) we make one instance for training/testing consisting of (a) event type, mention type, and text genre, (b) the shortest lexicalized dependency path $t \rightarrow m$ and (c) the sentence. Figure 2 depicts the pair ('returning', 'Ireland'). Event type (TRANSPORT), entity type (*loc*) and genre (newswire, not depicted) constitute the first information layer. The second Information layer is the lexicalized dependency path (returning $\xrightarrow{\text{nmod:from}}$ summit $\xrightarrow{\text{nmod:in}}$ Ireland). The third information layer is the sentence.

The three layers correspond to the most valuable information sources for the baseline. However, the baseline draws only simple categorical features from them. Most notably, it relies on having seen dependency paths during training because it cannot decompose them into meaningful subpaths, which is crucial for better identification performance (Section 3.1). The neural network architecture we present in Section 4.2 is able to automatically construct more meaningful features.

[3] For multiword expressions, the path connects trigger and entity mention head.

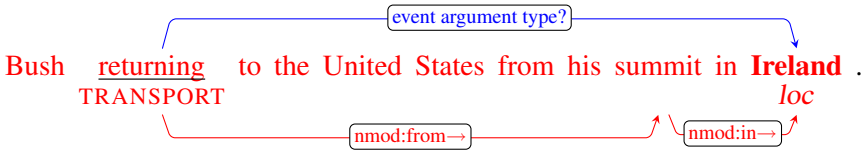


Figure 2: A training/test instance. Depicted in red is given information, depicted in blue is requested information. The trigger is underlined, the entity mention is bold.

4.2 biLSTM/CNN: Architecture

Input to our system (called biLSTM/CNN) are instances as described in Section 4.1. Each instance has three information layers, each layer is processed by a separate component. Figure 3 depicts the system architecture. The figure is split in four (bottom, middle left/right, and top), each part visualizing one component, plus the final classification. We will now describe each part. In the following, $\oplus$ means the concatenation of vectors.

Bias

The bias vector $\mathbf{b}$ provides a representation of the event type, entity type, and genre. The intuition behind the bias vector is that arguments are expressed differently across event types, entity types, and genres. $\mathbf{b}$ is input to the other two components and helps to learn better representations.

More formally, $\mathbf{b}$ is the last layer of a fully-connected three layer neural network whose input is defined as follows: $\mathbf{b}_1 = en \oplus ev \oplus ge$, where en, ev, and ge are representations of the entity type, event type, and genre. They are randomly initialized and will receive standard backpropagation updates during training.

biLSTMs

Representing dependency paths with Long Short-term Memory networks (LSTMs, Hochreiter and Schmidhuber 1997) shows good results in fields like relation extraction (Xu et al., 2015) and semantic role labeling (Roth and Lapata, 2016). We investigate their use for argument identification.

Our LSTMs produce a representation of lexicalized dependency paths, such that similar paths have similar representations. LSTMs automatically learn meaningful patterns in arbitrarily long paths. For example, they learn that the paths attacked$\xrightarrow{\text{nsubj}}$US and attacked$\xrightarrow{\text{nsubj}}$Iraq have similar representations given that both indicate an `Attacker` and only differ in two closely related words. They also learn that attacked$\xrightarrow{\text{dobj}}$Iraq has a different representation because the change from nsubj to dobj often distinguishes between `Target` and `Attacker`.

Input to our LSTM is a lexicalized dependency path $\mathbf{P} = (w_1, d^{1:2}, w_2, \ldots, d^{n-1:n}, w_n)$ where w_1 is the trigger word, w_n is the argument word, and $d^{a:b}$ is the dependency between w_a and w_b. The element at position i in $\mathbf{P}$ is resolved by a vector $\mathbf{v}_i = \mathbf{e}_i \oplus \text{dist}_{\text{trigger}} \oplus \text{dist}_{\text{mention}} \oplus \mathbf{b}$. $\mathbf{e}_i$ is either a pre-trained word embedding or a randomly-initialized dependency embedding, according to the element type at position i. $\text{dist}_{\text{trigger}}$ and $\text{dist}_{\text{mention}}$ refer to the lexical distance of a word to the trigger or the argument word, respectively.[4] Finally, $\mathbf{b}$ is our bias vector as defined above. We keep word embeddings fixed, but dependency embeddings receive backpropagation updates during training.

$\mathbf{P}$ is processed by a bidirectional LSTM (biLSTM). For a path element i, the biLSTM produces two (so-called) hidden states h_i^f (from the forward LSTM) and h_i^b (from the backward LSTM). These vectors contain information about the respective input, i.e., $\mathbf{v}_i$, as well as the hidden states of previously processed elements, i.e. elements to the left of i for h_i^f and elements to the right for h_i^b. We average the hidden states belonging to the same input vector: $\mathbf{a}_i = 1/2(\mathbf{h}_i^f + \mathbf{h}_{n-i+1}^b)$. Instead of averaging one could combine the vectors differently, e.g., by concatenating them. However, none of the other possibilities worked better in our case.

Using biLSTMs has the advantage that each $\mathbf{a}_i$ contains information from the entire sequence; using only a forward LSTM limits the representations at each position to the left context.

The middle-left part of Figure 3 depicts the biLSTM and its final output, the average vectors a.

Convolutional Neural Networks

In contrast to LSTMs, which are designed to capture the meaning of sequences, Convolutional Neural Networks (CNNs) are often used to pro-

[4] Dependencies have the same distance as their governor.

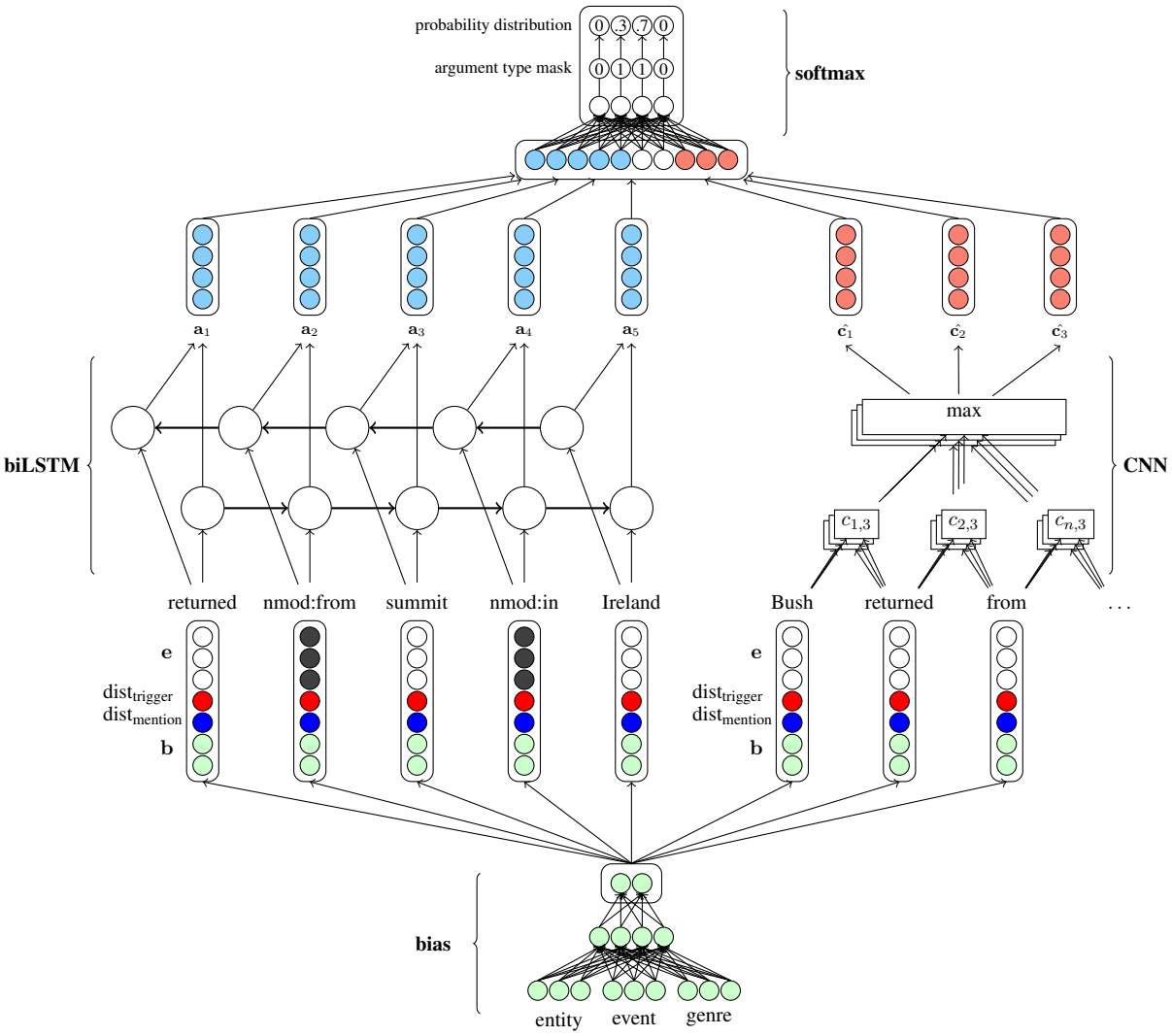


Figure 3: biLSTM/CNN architecture. Process flow is depicted from bottom to top. Embeddings $\mathbf{e}$ depicted in white are fixed, every other node receives backpropagation updates during training. Section 4.2 describes each component in detail.

duce bag-of-words-like representations. They were successfully applied to many NLP problems (Kim, 2014; Johnson and Zhang, 2015, inter alia).

Input to our CNN is a tokenized sentence where each word at position i is replaced by a vector $\mathbf{x}_i$ which is almost identical to the definition of $\mathbf{v}_i$ above. The only difference is that $\mathbf{x}_i$ contains only word embeddings.

CNNs apply filters (also called kernels) to a fixed number of consecutive input vectors. Filters are then moved by a certain offset (also called a stride) and re-applied. In our case, one filter produces one feature for one position i: $c_i = \sigma(\mathbf{W} \cdot \mathbf{x}_{i:i+h-1} + s)$ where σ is a non-linearity (tanh in our case), $\mathbf{W}$ is a weight matrix, $s \in \mathbb{R}$ is a bias, and $\mathbf{x}_{i:i+h-1}$ is a concatenation of vectors $[\mathbf{x}_i \oplus \mathbf{x}_{i+1} \oplus \ldots \oplus \mathbf{x}_{i+h-1}]$. h is the filter width. $\mathbf{W}$ and s receive backpropagation updates during training, word embeddings are fixed.

We use CNNs to represent a sentence. The sentence is important because the lexical contexts of trigger and mention convey valuable information for argument identification. This means that in our case a filter of width h produces features for the entire sentence, $c = [c1, c2, \ldots, c_{n-h+1}]$. We apply max-pooling afterwards, i.e., the final output of one CNN filter is given by $\hat{c} = \max(c)$. Our CNN uses 20 filters for filter widths 2,3,4. The middle-right part of Figure 3 exemplifies a CNN with 3 filters and filter width 2.

Final Classification

Finally, the averaged biLSTM hidden vectors for the dependency path and the max values for all CNN filters applied to the entire sentence serve as input to a softmax layer which produces a probability distribution over all argument types. We pick the class with the highest probability as our final result. However, choosing between *all* classes is unnecessary because not all combinations of event type, entity type and argument type are possible. For example, the argument type `Vehicle` can only be assigned to TRANSPORT events, and only mentions with entity type *veh* can be possible fillers. We modify softmax to assign zero probability to classes which are disallowed:

$$y_i = \frac{m_i e^{x_i}}{\sum_j m_j e^{x_j}}$$

The above equation gives the probability for a particular argument type, y_i, where x is the input vector to softmax, and m is a binary vector indicating allowed types. Note that $y_i > 0$ only if the respective argument type is allowed.

The top part of Figure 3 visualizes the softmax component. The input vector is first reduced to 29 dimensions (28 argument types and one 'null' type), multiplied with the restriction mask, and forwarded to our modified softmax.

Parameter Averaging

Inspired by the Averaged Perceptron (Freund and Shapire, 1999; Collins, 2002) we do not use the learned parameters directly for prediction. Instead, in each epoch we keep a moving average of the parameters:

$$\theta^V = \alpha\theta^T + (1-\alpha)\theta^{V-1}$$

Here, θ^T is the current weight vector after training epoch T, θ^{V-1} is the averaged weight vector before the new update, and α is the fraction of how much θ^T influences θ^V. We set $\alpha = 0.1$. θ^V is then used during testing. Note that this procedure does not change the training in any way.

With the formulation above, older weight vectors have less influence on θ^V than more recent vectors. After every training epoch, we evaluate θ^V on the development set and keep the version with highest F_1 for the final evaluation.

Parameter averaging leads to better generalization of our system. Furthermore, performance fluctuation for different training runs is reduced.

5 Experiments

5.1 Data, Evaluation Metrics and Parameters

We evaluate on ACE 2005 and use the same data split as most previous approaches (Ji and Grishman, 2008; Nguyen et al., 2016, inter alia). We follow standard evaluation procedures: An event argument is correct, if its span and role match a reference argument (Ji and Grishman, 2008). Section 1 gives an overview over the annotations provided in ACE 2005.

Because we want to measure argument identification performance by itself, we must ensure that compared systems use the same entity mention and trigger predictions; the best way to ensure this is to set both to gold. Using gold entity mentions is a common setting in event extraction (Li et al., 2013; Chen et al., 2015; Nguyen et al., 2016, inter alia). Since we have evaluation numbers only for biLSTM/CNN and our baseline in this setting, *there is no direct comparability* with previous work other than Li et al. (2013).

Both systems were trained on the same training set, and hyperparameters were optimized on the same development set. We trained both systems using `enhanced++` dependencies (Schuster and Manning, 2016).

We optimize hyper parameters for biLSTM/CNN using Random Search (Bergstra and Bengio, 2012). We use a batch size of 450, 20 CNN filters, 150 LSTM dimensions, and 130 bias dimensions. In order to deal with class imbalance, we set the weight of non-null training samples to 2; this value is used to scale the loss accordingly. We used Keras (Chollet et al., 2015) version 2.0.2 with the TensorFlow backend as the learning framework. Training the network on an NVIDIA P40 GPU takes about 20 seconds per epoch.

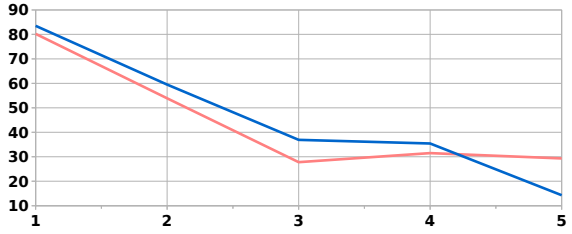
5.2 Experiments and Results

We report the results of four main experiments, namely measuring argument identification performance in general, grouped by argument type, grouped by dependency path length, and using only a dependency path biLSTM. We report precision, recall, and F_1.

Training neural networks is usually a non-deterministic process. In order to increase reliability, training was performed five times and the evaluation on the test set was averaged across five evaluation runs, one run per trained model.

		Baseline			biLSTM/CNN				
		P	R	F_1	P	R	F_1	Support	ΔF_1
1	**Micro**	**67.7**	**58.7**	**62.9**	**63.1**	**68.3**	**65.5**†	**916**	**2.6**±0.5
2	dep-path biLSTM	-	-	-	64.9	64.0	64.4	916	1.5
3	`Time`	69.9	70.9	70.4	70.9	80.1	75.2	134	4.8
4	`Entity`	63.7	56.7	60.0	57.7	65.2	61.2	127	1.2
5	`Place`	64.0	41.7	50.5	52.1	48.0	49.9	115	-0.6
6	`Person`	74.6	61.7	67.6	69.1	78.3	73.4	81	5.8
7	`Artifact`	78.5	71.8	75.0	70.3	77.2	73.5	71	-1.5
8	`Destination`	63.4	66.7	65.0	65.6	80.0	72.1	39	7.1
9	`Crime`	84.4	100.0	91.6	82.5	99.5	90.2	38	-1.4
10	`Attacker`	60.7	47.2	53.1	52.4	66.6	58.6	36	5.5
11	`Defendant`	70.0	63.6	66.7	67.6	75.2	71.1	33	4.4
12	`Agent`	64.7	34.3	44.9	55.9	40.6	46.8	32	1.9

Table 2: Test set precision, recall, and F_1 for the baseline and biLSTM/CNN, ordered by frequency. Reported are argument types with more than 30 instances. **Mirco** reports micro-averaged numbers, averaged across 5 training and testing rounds, 'dep-path biLSTM' reports numbers using only a dependency path biLSTM, the other rows report numbers per argument type. 'Support' reports the number of instances for the respective type. 'ΔF_1' reports the difference in F_1 between biLSTM/CNN and the baseline, as well as the standard deviation of biLSTM/CNN F_1. † means statistically significant for every training and testing round at the $p < 0.05$ level.



(a) Test set F_1 performance plotted against dependency path length. The red curve depicts baseline F_1, the blue curve depicts biLSTM/CNN F_1.

	Baseline	biLSTM/CNN		
Length	F_1	F_1	Support	ΔF_1
1	80.2	83.5	432	3.3
2	53.9	59.5	248	5.6
3	27.8	36.9	123	9.1
4	31.5	35.4	59	3.9
5	29.3	14.3	26	-15.0

(b) Test set F_1 by dependency path length for the baseline and biLSTM/CNN. 'Support' reports the number of instances, 'Δ F_1' reports the difference in F_1 between biLSTM/CNN and the baseline.

Figure 4: Test set F_1 by dependency path length for the baseline and biLSTM/CNN.

Table 2 reports micro-averaged evaluation numbers for the baseline and biLSTM/CNN (Line 1), a variant which uses only a dependency path biLSTM (without context CNNs, Line 2) as well as numbers per argument type (Lines 3-12). The column 'Support' gives the number of instances for the respective evaluation. Finally, the column 'Δ F_1' reports the difference in F_1 between biLSTM/CNN and the baseline, positive numbers meaning better biLSTM/CNN performance, as well as the standard deviation of the biLSTM/CNN F_1 score.

As we can see in Table 2, Line 1, biLSTM/CNN has a lower precision and a considerably higher recall than the baseline, resulting in an increase of 2.6 points in micro-averaged F_1 (with a standard deviation of 0.5 F_1 points). This is statistically significant at the $p < 0.05$ level.[5] Note that biLSTM/CNN does not use any manually engineered features, whereas the baseline uses two dozen feature templates, resulting in 150,000 features. Furthermore, biLSTM/CNN is a simple trigger-argument-pair classifier, whereas the baseline jointly predicts all arguments of all triggers in a sentence.

When we compare the performance using dependency paths alone (Line 2), recall drops by 4.3 points, while increasing precision slightly by 1.8 points, resulting in a decrease of 1.1 F_1 points. The main advantage of the CNN is that it makes the lexical context outside of the shortest dependency path available to the system, which reflects itself in the increased recall.

When we look at individual argument types, we note that biLSTM/CNN improves performance for all but three types. `Destination` has the highest performance improvement (7.1 F_1 points), `Artifact` the highest loss (-1.5 F_1 points).

[5] We measured significance using approximate randomization (Noreen, 1989). Each of the 5 models we trained performed significantly better than the baseline.

Time as the most frequent type in the test data has a high improvement of 4.8 F_1 points.

Figure 4 reports micro-averaged numbers for the baseline and biLSTM/CNN per dependency path length. Figure 4a is a visualization of Table 4b. In total, 888 arguments (out of the 916 in the test set) were connected to their triggers by dependency paths of length 5 or less. biLSTM/CNN performs considerably better for lengths 1-4, especially for paths of length 2 (+5.6 F_1) and 4 (+9.1 F_1). Length-1 paths, which are nearly as frequent as all other path lenghts together, have an increased performance of 3.3 F_1 points. Only length-5 paths (with a test set support of 26) lose 15 F_1 points, mainly because biLSTM/CNN produced some false positives for this class.

Most of biLSTM/CNN's errors are either false positives (wrongly classified as an argument) or false negatives (missed an argument). When it confuses argument types, it usually confuses opposing types like Seller and Buyer.

6 Related Work

To the best of our knowledge, no other paper is targeting event argument identification directly. For this reason, we first summarize 'neural' event extractors, for which argument identification is one necessary step. Then, we report work on representing dependency paths with neural networks.

Chen et al. (2015) use a pipelined event extractor based on CNNs. The input is a sentence, and the output is, in phase one, all triggers in the sentence and, in phase two, a classification of the argument type of a trigger-argument candidate pair. The second phase is similar to the setting we analyze in this paper.

Nguyen et al. (2016) propose a joint and hybrid approach using a Gated Recurrent Unit network (Cho et al., 2014), a variant of an LSTM. Input to their network is the word embedding matrix of a sentence. In contrast to Chen et al. (2015), they predict triggers and arguments jointly. They concatenate a one-hot representation of dependencies to each word embedding. In contrast to this paper, they do not attempt to directly operate on syntactic structure. Instead, the GRU goes over the sentence and passes its hidden states on to higher levels of the network, which dynamically output triggers and arguments for the entire sentence. Their approach is hybrid because they additionally use the features from Li et al. (2013) We did not choose this system as our baseline because it is considerably more complex and would heavily rely on the same features as our baseline in the setting we investigate in this paper.

Xu et al. (2015) use LSTMs for relation extraction. Similar to our work, they use LSTMs to compute a representation of the shortest dependency path connecting two related entities. While the general learning scheme is similar to our biLSTM component, they represent dependency paths differently. Instead of one lexicalized path, they construct four different representations, using only dependency labels, only words, only part-of-speech tags, and only WordNet categories. This results in four LSTMs whose output is concatenated.

Roth and Lapata (2016) use dependency path-encoding LSTMs for semantic role labeling (SRL). Event extraction and SRL are similar in terms of structures: SRL also involves finding 'triggers' and 'arguments'. They differ however in the nature of these structures. For example, potential arguments in SRL may be assigned to every verb in a sentence, while in event extraction, potential arguments must be assigned only to event triggers. Similar to our work, Roth and Lapata (2016) use an LSTM which processes a dependency path connecting predicate and argument. Their dependency paths mix words, part-of-speech tags and dependency relations.

7 Conclusions

In this paper, we show that argument identification performance is tied to the length of dependency paths connecting triggers and arguments. We propose a novel and efficient neural network that targets syntactic complexity. Without manual feature engineering, it learns to produce meaningful representations of dependency paths and to extract relevant lexical context of arguments. We show that our system outperforms a state-of-the-art baseline which uses manually engineered features and predicts arguments jointly.

Acknowledgments

This work has been funded by the Klaus Tschira Foundation, Heidelberg, Germany. The first author has been supported by a HITS PhD scholarship. We thank the anonymous reviewers for their helpful comments.

References

James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305.

Yubo Chen, Liheng Xu, Kang Liu, Daojian Zeng, and Jun Zhao. 2015. Event extraction via dynamic multi-pooling convolutional neural networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers),* Beijing, China, 26–31 July 2015, pages 167–176.

Kyunghyun Cho, Bart van Merriënboer Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing,* Doha, Qatar, 25–29 October 2014, pages 1724–1734.

François Chollet et al. 2015. Keras. `https://github.com/fchollet/keras`.

Michael Collins. 2002. Discriminative training methods for Hidden Markov Models: Theory and experiments with perceptron algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing,* Philadelphia, Penn., 6–7 July 2002, pages 1–8.

Yoav Freund and Robert Shapire. 1999. A short introduction to boosting. *Journal of the Japanese Society for Artificial Intelligence*, 14:771–780.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Heng Ji and Ralph Grishman. 2008. Refining event extraction through cross-document inference. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies,* Columbus, Ohio, 15–20 June 2008, pages 254–262.

Rie Johnson and Tong Zhang. 2015. Effective use of word order for text categorization with Convolutional Neural Networks. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies,* Denver, Col., 31 May – 5 June 2015, pages 103–112.

Yoon Kim. 2014. Convolutional Neural Networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing,* Doha, Qatar, 25–29 October 2014, pages 1746–1751.

Qi Li, Heng Ji, Yu Heng, and Sujian Li. 2014. Constructing information networks using one single model. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing,* Doha, Qatar, 25–29 October 2014, pages 1846–1851.

Qi Li, Heng Ji, and Liang Huang. 2013. Joint event extraction via structured prediction with global features. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers),* Sofia, Bulgaria, 4–9 August 2013, pages 73–82.

Thien Huu Nguyen, Kyunghyun Cho, and Ralph Grishman. 2016. Joint event extraction via recurrent neural networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies,* San Diego, Cal., 12–17 June 2016, pages 300–309.

Eric W. Noreen. 1989. *Computer Intensive Methods for Hypothesis Testing: An Introduction*. Wiley, New York, N.Y.

Michael Roth and Mirella Lapata. 2016. Neural semantic role labeling with dependency path embeddings. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers),* Berlin, Germany, 7–12 August 2016, pages 1192–1202.

Sebastian Schuster and Christopher D. Manning. 2016. Enhanced english universal dependencies: An improved representation for natural language understanding tasks. In *Proceedings of the 10th International Conference on Language Resources and Evaluation,* Portorož, Slovenia, 23–28 May 2016.

Yan Xu, Lili Mou, Ge Li, Yunchuan Chen, Hao Peng, and Zhi Jin. 2015. Classifying relations via Long Short Term Memory Networks along shortest dependency paths. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing,* Lisbon, Portugal, 17–21 September 2015, pages 1785–1794.