# Graph Convolutional Networks with Argument-Aware Pooling for Event Detection

**Thien Huu Nguyen**
Department of Computer and Information Science
University of Oregon
Eugene, Oregon 97403, USA
thien@cs.uoregon.edu

**Ralph Grishman**
Computer Science Department
New York University
New York, NY 10003 USA
grishman@cs.nyu.edu

## Abstract

The current neural network models for event detection have only considered the sequential representation of sentences. Syntactic representations have not been explored in this area although they provide an effective mechanism to directly link words to their informative context for event detection in the sentences. In this work, we investigate a convolutional neural network based on dependency trees to perform event detection. We propose a novel pooling method that relies on entity mentions to aggregate the convolution vectors. The extensive experiments demonstrate the benefits of the dependency-based convolutional neural networks and the entity mention-based pooling method for event detection. We achieve the state-of-the-art performance on widely used datasets with both perfect and predicted entity mentions.

## Introduction

Event Detection (ED) is an important information extraction task of natural language processing that seeks to recognize instances of specified types of events (event mentions) in text. Each event mention is often presented within a single sentence in which an event trigger is selected to associate with that event mention. Event triggers are generally single verbs or nominalizations that serve as the main words to evoke the corresponding events. The event detection task, more precisely stated, aims to detect event triggers and classify them into specific types of interest. For instance, consider the following sentence with two words "*fired*":
"*The police officer who **fired** into a car full of teenagers was **fired** Tuesday*"
In this example, an ED system should be able to realize that the first occurrence of "*fired*" is an event trigger of type *Attack* while the second "*fired*" takes *End-Position* as its event type. ED is a challenging task, as an expression might evoke different events depending on contexts (illustrated in our previous example with the word "*fired*"), and the same event might be presented in various expressions (e.g, the trigger words "*killed*", "*shot*" or "*beat* for the event type *Attack*).

The current state-of-the-art approach for ED employs deep learning models in which convolutional neural networks (CNN) are the typical architectures (Nguyen and Grishman 2015; Chen et al. 2015; Nguyen and Grishman 2016b). In the basic implementation, CNNs apply the temporal convolution operation over the consecutive $k$-grams[1] in the sentences, attempting to generate the latent structures that are informative for ED (Nguyen and Grishman 2015; Chen et al. 2015). The disadvantage of such consecutive convolution is the inability to capture the non-consecutive $k$-grams that can span words far apart in the sentences. Those non-consecutive $k$-grams are necessary to recognize event triggers in some situations. For example, in the example above, the non-consecutive 3-grams "*officer was fired*" should be considered to correctly identify the event type *End-Position* for the second word "*fired*". The non-consecutive CNN model (NCNN) in (Nguyen and Grishman 2016b) seeks to overcome this problem by operating the temporal convolution over all the non-consecutive $k$-grams in the sentences, leading to the state-of-the-art CNN model for ED.

Unfortunately, due to the consideration of all possible non-consecutive $k$-grams, the non-consecutive CNN architecture might model unnecessary and noisy information, potentially impairing the prediction performance for ED. In particular, NCNN utilizes the max-pooling operation to aggregate the convolution scores over all the non-consecutive $k$-grams. As such $k$ grams might include irrelevant or misleading sequences of words, the max-pooling might incorrectly focus on those $k$-grams and make a wrong final prediction for ED. One example is the non-consecutive 3-gram "*car was fired*" in the example above. In contrast to the correct 3-gram "*officer was fired*", "*car was fired*" suggests the event type *Attack* for the second word "*fired*", causing the confusion or failure of NCNN to predict the true event type of "*End-Position*" in this situation.

One way to circumvent this issue for NCNN is to notice that "*police officer*" is the subject of the second word "*fired*" while "*a car*" does not have much direct connection with the second "*fired*" in this example. Guided by this intuition, in this paper, we propose to perform the convolution operation over the syntactic dependency graphs of the sentences to perform event detection. Syntactic dependency graphs represents sentences as directed trees with head-modifier dependency arcs between related words (Mcdonald and Pereira 2006; Koo, Carreras, and Collins 2008). Each word in such



[1] $k$ is often chosen to be some fixed value.

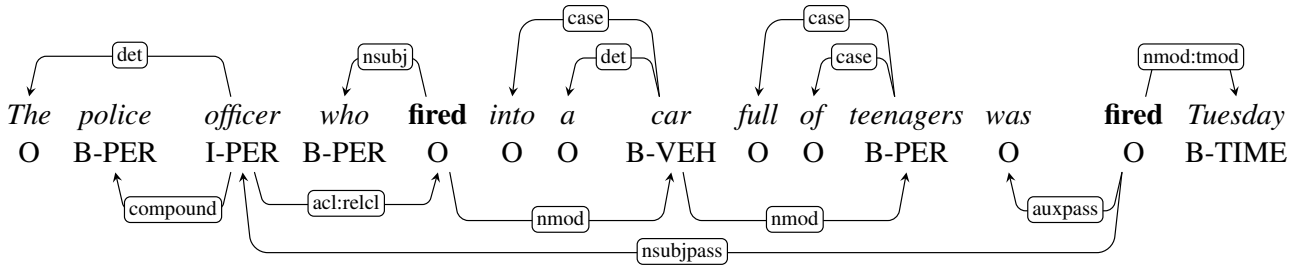


Figure 1: BIO annotation for entity mentions and dependency parse tree using universal dependency relations for the example sentence. The label "B-X" for entity mentions indicates the beginning of an entity mention of type "X" while "I-X" is used for tokens that are inside (but do not start) the range of an entity mention of type "X". The label "O" is reserved for other tokens that do not belong to any entity mentions. In this figure, "PER" and "VEH" stands for *PERSON* and *VEHICLE* respectively.

graphs is surrounded by its direct syntactic governor and dependent words (the neighbors), over which the convolution can focus on the most relevant words for the current word and avoid the modeling of unrelated words/$k$-grams. In the experiments, we demonstrate that these syntactic connections for words provide effective constraints to implement convolution for ED. Note that the governor and dependent words has also been found as useful features for ED in the traditional feature approaches (Ahn 2006; Li, Ji, and Huang 2013). This further helps to justify the convolution over dependency graphs for ED in this paper. The dependency parse tree for the previous example sentence is shown in Figure 1. As we can see from this figure, the dependency tree helps to link the second word "*fired*" directly to the dependent words "*officer*" and "*was*" that altogether constitute an effective evidence to predict the event type for "*fired*" via convolution.

In order to implement the syntactic convolution, we employ the graph convolutional networks (GCNs) (Kearnes et al. 2016; Kipf and Welling 2017; Marcheggiani and Titov 2017) that are studied very recently to use graph structures to form connections between layers of multilayer neural networks. In GCNs, the convolution vector for each node is computed from the representation vectors of the immediate neighbors. GCNs has been mainly applied for the node classification tasks in which the convolution representation vector for a node functions as the only features to classify that node (Kipf and Welling 2017; Marcheggiani and Titov 2017). For event detection, we can also utilize the graph-based convolution vector of the current word (the current node in the dependency graphs) to perform prediction. Unfortunately, as the convolution vector tends to preserve only the most important information of the local context for the current word (i.e, the immediate neighbors in the dependency graph), it might not have the capacity to encode the specific (detailed) information about the entity mentions distributed at different positions in the sentences. An entity mention is a reference to an object or a set of objects in the world, including names, nominals and pronouns such as the entity mentions "*police officer*", "*car*", "*teenagers*" and "*Tuesday*" in the example sentence above[2]. The specific knowledge about entity mentions (e.g, entity types), especially the participants (arguments) of the events, is important as it might provide models with more confidence to make prediction for ED (Nguyen and Grishman 2015; Liu et al. 2017). For instance, the first and the second words "*fired*" in the example sentence might be aware of the entity mentions (arguments) "*car*" and "*officer*" in their syntactic context respectively; however, the types of such entity mentions (i.e, *VEHICLE* for "*car*" and *PERSON* for "*officer*") might not be encapsulated or be less pronounced in the convolution vectors for the two words "*fired*" due to the local attention. These entity types are crucial to accurately predict the event types for the two words "*fired*" in this case.

In this work, we propose to overcome this issue by *operating a pooling over the graph-based convolution vectors of the current word as well as the entity mentions in the sentences*. This aggregates the convolution vectors to generate a single vector representation for event type prediction. The rationale is to explicitly model the information from the entity mentions to improve the performance for ED. We extensively evaluate the proposed pooling method with both manually annotated (perfect) entity mentions and automatically predicted entity mentions to demonstrate its benefit in the experiments.

To summary, our contribution in this work is as follows:

- We are the first to integrate syntax into neural event detection and show that GCNs are effective for ED.
- We propose a novel pooling method based on entity mentions for ED.
- We achieve the state-of-the-art performance on the widely used datasets for ED using the proposed model with GCNs and entity mention-based pooling.

## Model

Event detection can be cast as a multi-class classification problem (Nguyen and Grishman 2015; Chen et al. 2015; Nguyen and Grishman 2016b; Liu et al. 2017). Each word in the document is associated with the sentence containing the word (the context) to form an event trigger candidate or an example in the multi-class classification terms. Our task

[2]For convenience, we also consider time and value expressions as entity mentions in this work.

is to predict the event label for every event trigger candidate in the document. The label can be one of the pre-defined event types (subtypes) in the datasets or *NONE* to indicate a non-trigger candidate. Consequently, we have an equivalent problem of $(L+1)$-class classification for ED where $L$ is the number of pre-defined event types.

Let $w = w_1, w_2, \ldots, w_n$ be a sentence of length $n$ of some event trigger candidate, in which $w_a$ $(1 \leq a \leq n)$ is the current word for trigger prediction ($w_i$ is the $i$-th token in the sentence $\forall 1 \leq i \leq n$). In addition, as we assume the availability of the entity mentions (i.e, the positions and the types) in $w$, we can utilize the BIO annotation scheme to assign the entity type label $e_i$ to each token $w_i$ of $w$ using the non-overlapping heads (the most important tokens) of the entity mentions. This results in the sequence of entity type labels $e_1, e_2, \ldots, e_n$ for $w$, demonstrated in Figure 1 for the example sentence. Note that in such a scheme, $e_i \neq$ O implies that $w_i$ is within the range of an entity mention in $w$.

The graph convolutional networks for ED in this work consists of three modules: (i) the encoding module that represents the input sentence with a matrix for GCN computation, (ii) the convolution module that performs the convolution operation over the dependency graph structure of $w$ for each token in the sentence, and (iii) the pooling module that aggregates the convolution vectors based on the positions of the entity mentions in the sentence to perform ED.

## 1. Encoding

In the encoding module, each token $w_i$ in the input sentence is transformed into a real-valued vector $x_i$ by concatenating the following vectors:

- The word embedding vector of $w_i$: This is a real-valued vector that captures the hidden syntactic and semantic properties of $w_i$ (Bengio et al. 2003). Word embeddings are often pre-trained on some large unlabeled corpus (Mikolov et al. 2013).
- The position embedding vector of $w_i$: In order to indicate that $w_a$ is the current word, we encode the relative distance from $w_i$ to $w_a$ (i.e, $i - a$) as a real-valued vector (called as the position embedding vector) and use this vector as an additional representation of $w_i$. We obtain the position embedding vector by looking up the position embedding table that maps the possible values of the relative positions (i.e, $i - a$) into randomly initialized vectors (Nguyen and Grishman 2016b; Liu et al. 2017).
- The entity type embedding vector of $w_i$: Similar to the position embeddings, we maintain a table of entity type embeddings that maps entity type labels of tokens (i.e, the BIO labels for entity mentions) to real-valued random vectors. We look up this table for the entity type label $e_i$ of $w_i$ to retrieve the corresponding embedding.

As each token $w_i$ is represented by the vector $x_i$ with dimensionality of $d_0$, the input sentence $w$ can be seen as a sequence of vectors $X = x_1, x_2, \ldots, x_n$. $X$ would be used as input for the graph convolution module in the next step.

## 2. Graph Convolution

Let $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ be the dependency parse tree for $w$ with $\mathcal{V}$ and $\mathcal{E}$ as the sets of nodes and edges of $\mathcal{G}$ respectively. $\mathcal{V}$ contains $n$ nodes corresponding to the $n$ tokens $w_1, w_2, \ldots, w_n$ in $w$. For convenience, we also use $w_i$ to denote the $i$-th node in $\mathcal{V}: \mathcal{V} = \{w_1, w_2, \ldots, w_n\}$. Each edge $(w_i, w_j)$ in $\mathcal{E}$ is directed from the head word $w_i$ to the dependent word $w_j$ and has the dependency label $L(w_i, w_j)$. For instance, in the dependency tree of Figure 1, there is a directed edge from the node for the second word $w_i =$ "*fired*" (the head word) to the node for the word $w_j =$ "*officer*" (the dependent word) with the edge label $L(w_i, w_j) = L$("*fired*", "*officer*") = *nsubjpass*.

In order to allow the convolution for each token $w_i$ in $\mathcal{G}$ to involve the word $w_i$ itself as well as its governor word (if any) in the dependency graph, we add the self loops $(w_i, w_i)$ and the inverse edges $(w_j, w_i)$ $((w_i, w_j) \in \mathcal{E})$ into the initial edge sets $\mathcal{E}$, resulting in a new set of edges $\mathcal{E}'$ (Kipf and Welling 2017; Marcheggiani and Titov 2017):

$$\mathcal{E}' = \mathcal{E} \cup \{(w_i, w_i) : 1 \leq i \leq n\} \cup \{(w_j, w_i) : (w_i, w_j) \in \mathcal{E}\}$$

Note that the additional edges of $\mathcal{E}'$ are also directed and labeled. The label for the self loops is a special symbol "*LOOP*" while the label for the inverse edge $(w_j, w_i)$ involves the label of the corresponding original edge $(w_i, w_j)$ followed by an apostrophe to emphasize the opposite direction with respect to the original edge $(w_i, w_j)$ in $\mathcal{G}$:

$$L(w_i, w_i) = \textit{LOOP}\ \forall 1 \leq i \leq n$$
$$L(w_j, w_i) = L'(w_i, w_j)\ \forall (w_i, w_j) \in \mathcal{E}$$

The new edge set $\mathcal{E}'$ along with the node set $\mathcal{V}$ constitute a new graph $\mathcal{G}' = \{\mathcal{V}, \mathcal{E}'\}$ on which the convolution operation can rely. In particular, the graph convolution vector $h_u^{k+1}$ at the $(k+1)$-th layer $(k \geq 0)$ for a node $u \in \mathcal{G}'$ (corresponding to a word $w_i$ in the input sentence $w: u = w_i \in \{w_1, w_2, \ldots, w_n\}$) is computed by:

$$h_u^{k+1} = g\left(\sum_{v \in \mathcal{N}(u)} W_{L(u,v)}^k h_v^k + b_{L(u,v)}^k\right) \quad (1)$$

where $\mathcal{N}(u)$ is the set of neighbors of $u$ in $\mathcal{G}'$: $\mathcal{N}(u) = \{v : (u, v) \in \mathcal{E}'\}$; $W_{L(u,v)}^k \in \mathbb{R}^{d_{k+1} \times d_k}$ and $b_{L(u,v)}^k \in \mathbb{R}^{d_{k+1}}$ are the weight matrix and the bias (respectively) for the edge $(u, v)$ in $\mathcal{G}'$ ($d_k$ is the number of hidden units or the dimensionality of $h_u^k$ in the $k$-th layer); and $g$ is a nonlinear activation function[3]. For convenience, we assume the same number of hidden units for all the graph convolution layers in this work (i.e, $d_1 = d_2 = \ldots = d$). Note that the initial vectors $h_u^0$ are set to the representation vectors obtained from the encoding module:

$$h_u^0 = h_{w_i}^0 = x_i\ \forall u \in \mathcal{V} \quad (2)$$

**Limiting the capacity** The convolution in Equation (1) assumes different weight matrices $W_{L(u,v)}^k$ for different edge labels $L(u, v)$. The capacity of such parameters might be too

[3] $g$ is the rectify function $g(x) = \max(0, x)$ in this paper.

high, given that the datasets for ED often have moderate size with respect to the deep learning perspectives. In order to reduce the capacity, following (Marcheggiani and Titov 2017), we only use three different weight matrices $W^k_{L(u,v)}$ in each layer depending on whether the corresponding edge $(u, v)$ is an original edge in $\mathcal{E}$, a self loop or an added inverse edge in $\mathcal{E}'$:

$$W^k_{L(u,v)} = W^k_{\text{type}(u,v)} \quad (3)$$

where "type$(u, v)$" returns the type of the edge $(u, v)$ (i.e, original edges, self loops and inverse edges).

**Weighting the edges** The second word "*fired*" in the example sentence of Figure 1 has three immediate neighbors in the dependency graph: "*officer*", "*was*" and "*Tuesday*". While "*officer*" and "*was*" are crucial to determine the event type of *End-Position* for "*fired*", "*Tuesday*" do not contribute much information in this case. It is thus not appropriate to weight the neighbors uniformly in the graph convolution for ED. Consequently, for the $k$-th layer, we compute a weight $s^k_{(u,v)}$ for each neighboring edge $(u, v)$ of a node $u \in \mathcal{V}$ to quantify its importance for ED in GCNs (Marcheggiani and Titov 2017):

$$s^k_{(u,v)} = \sigma(h^k_v \bar{W}^k_{\text{type}(u,v)} + \bar{b}^k_{L(u,v)}) \quad (4)$$

where $\bar{W}^k_{\text{type}(u,v)} \in \mathbb{R}^{d_k}$ and $\bar{b}^k_{L(u,v)} \in \mathbb{R}$ are weight matrix and the bias (respectively); and $\sigma$ is a nonlinear activation function[4].

The edge weights in Equation (4) and the weight matrices in Equation (3) transform the convolution operation in Equation (1) into:

$$h^{k+1}_u = g\left(\sum_{v \in \mathcal{N}(u)} s^k_{(u,v)}(W^k_{\text{type}(u,v)} h^k_v + b^k_{L(u,v)})\right) \quad (5)$$

Note that edge weighting also helps to alleviate the effect of the potentially wrong syntactic edges that are automatically predicted by imperfect syntactic parsers.

**Abstracting the initial representation with LSTM** The graph convolution induces a hidden representation for the local graph context of each node (word) in $\mathcal{V}$ (i.e, the word itself, the governor and the dependents), functioning as features for ED. The hidden representations of a single layer of GCNs can only capture the information for the immediate neighbors while those of multiple layers (e.g, $K$ layers) can incorporate nodes (words) that are at most $K$ hops aways in the dependency tree. In other words, the context coverage of the representation vectors for the nodes is restricted by the number of convolution layers, causing the inability of the representation vectors to encode the dependencies between words far away from each other in the dependency graph. Increasing the number of convolution layers might help to mitigate this problem, but it might fail to capture the word dependencies with shorter distances due to the redundant modeling of context. It is thus preferable to have a mechanism to adaptively accumulate the context rather than fixing the context coverage with $K$ layers in the current formulation of GCNs. In this work, we employ a bidirectional long-short term memory network (BiLSTM) (Hochreiter and Schmidhuber 1997) to first abstract the initial representation vectors $x_i$ whose outputs are later consumed by GCNs for ED.

Specifically, we run a forward LSTM and a backward LSTM over the representation vector sequence $(x_1, x_2, \ldots, x_n)$ to generate the forward and backward hidden vector sequences (i.e, $(\overrightarrow{r_1}, \overrightarrow{r_2}, \ldots, \overrightarrow{r_n})$ and $(\overleftarrow{r_1}, \overleftarrow{r_2}, \ldots, \overleftarrow{r_n})$ respectively). We then concatenate the hidden vectors at the corresponding positions to obtain the abstract representation vector sequence $(r_1, r_2, \ldots, r_n)$ where $r_i = [\overrightarrow{r_i}, \overleftarrow{r_i}]$. The new representation vectors $r_1, r_2, \ldots, r_n$ would then replace the initial vector sequences $x_1, x_2, \ldots, x_n$ in Equation (2) for further computation of GCNs in Equations (4) and (5):

$$h^0_u = h^0_{w_i} = r_i \; \forall u \in \mathcal{V} \quad (6)$$

The convolution of GCNs over these new representation vectors would allow the adaptive integration of long-range dependencies of words with fewer convolution layers in GCNs (Marcheggiani and Titov 2017).

## 3. Pooling

The GCN model with $K$ convolution layers produces the sequence of convolution representation vectors $h^K_{w_1}, h^K_{w_2}, \ldots, h^K_{w_n}$. The role of the pooling module is to aggregate such convolution vectors to generate a single vector representation $v^{ED}$, that would be fed into a standard feed-forward neural network with softmax in the end to perform classification for ED.

There are several methods to aggregate the convolution vectors for ED in the literature. In this section, we first review these methods to emphasize the entity mention-based pooling in this paper.

- Anchor Pooling (*ANCHOR*): In this case, $v^{ED}$ is set to the convolution vector of the current word: $v^{ED} = h^K_{w_a}$. This method is used in (Nguyen, Cho, and Grishman 2016a) and most work on GCNs so far (Kipf and Welling 2017; Marcheggiani and Titov 2017).
- Overall Pooling (*OVERALL*): $v^{ED}$ is computed by taking the element-wise max over the entire convolution vector sequence $h^K_{w_1}, h^K_{w_2}, \ldots, h^K_{w_n}$: $v^{ED} =$ *max_element-wise*$(h^K_{w_1}, h^K_{w_2}, \ldots, h^K_{w_n})$. This methods is employed in (Nguyen and Grishman 2015; 2016b).
- Dynamic Pooling (*DYNAMIC*) (Chen et al. 2015; 2017): The convolution vector sequence is divided into two parts based on the position of the current word (i.e, $(h^K_{w_1}, h^K_{w_2}, \ldots, h^K_{w_a})$ and $(h^K_{w_{a+1}}, h^K_{w_{a+2}}, \ldots, h^K_{w_n})$). These two subsequences are then aggregated by an element-wise max operation whose outputs are concatenated to generate $v^{ED}$:
  $v^{ED} = [$*max_element-wise*$(h^K_{w_1}, h^K_{w_2}, \ldots, h^K_{w_a})$, *max_element-wise*$(h^K_{w_{a+1}}, h^K_{w_{a+2}}, \ldots, h^K_{w_n})]$

The common limitation of these methods is the failure to explicitly model the convolution representation vectors for

[4]The sigmoid function in this case.

the entity mentions in the sentence. Such representation vectors are helpful as they encode specific information for the entity mentions that might help to improve the ED performance. In particular, *ANCHOR* ignores the representation vectors for the entity mentions while *OVERALL* and *DYNAMIC* consider both the entity mentions' representations and the others uniformly in $v^{ED}$, potentially rejecting the representation vectors of the entity mentions if the representation vectors from the other words in the sentence accidentally receive higher values. In this paper, we propose to exclusively rely on the representation vectors of the entity mentions to perform the pooling operation for ED. To be more specific, the representation vector $v^{ED}$ in this entity mention-based pooling (called *ENTITY*) is computed by:

$$v^{ED} = \textit{max\_element-wise}(\{h^K_{w_a}\} \cup \{ h^K_{w_i} : 1 \leq i \leq n, e_i \neq \text{O}\}) \quad (7)$$

To summarize, the proposed model for ED in this paper works in the following order:
1. Initial encoding with word embeddings, position embeddings and entity type embeddings
2. Abstracting the initial encoding with bidirectional LSTM
3. Performing convolution over the dependency trees using the BiLSTM representation (Equation 5)
4. Pooling over the convolution vector based on the positions of the entity mentions (Equation 7)
5. Feed-forward neural networks with softmax for prediction

### Training

In order to train the networks, following the previous work on ED (Nguyen and Grishman 2015; Chen et al. 2015; Nguyen and Grishman 2016b; Liu et al. 2017), we minimize the negative log-likelihood on the training dataset using stochastic gradient descent with shuffled mini-batches and the AdaDelta update rule. The gradients are computed via back-propagation while dropout is employed to avoid overfitting. We also rescale the weights whose $l_2$-norms exceed a predefined threshold.

## Experiments

### 1. Datasets and Settings

We evaluate the networks in this paper using the widely used datasets for ED, i.e, the ACE 2005 dataset and the TAC KBP 2015 dataset. We employ the ACE 2005 dataset in the setting with golden (perfect) annotation for entity mentions as do the prior work (Nguyen and Grishman 2015; 2016b; Liu et al. 2017). TAC KBP 2015, on the other hand, is exploited to test the networks for the setting with predicted entity mentions (i.e, the annotation for entity mentions in the sentences is provided by some automatic entity mention detector(Li et al. 2014b)). Although the predicted entity mentions might involve some errors, it is a more realistic setting as we usually do not have the golden entity mentions for the datasets in practice.

The ACE 2005 dataset annotate 33 event subtypes that, along with, the *NONE* class, function as the pre-defined label set for a 34-class classification problem for this dataset. In order to ensure a compatible comparison with the previous work on this dataset (Nguyen and Grishman 2015; Chen et al. 2015; Nguyen and Grishman 2016b; Chen et al. 2017; Liu et al. 2017), we use the same data split with 40 newswire articles for the test set, 30 other documents for the development set and 529 remaining documents for the training set.

The TAC KBP 2015 dataset is the official evaluation data from the Event Nugget Detection Evaluation of the 2015 Text Analysis Conference (TAC). It has 38 event subtypes, thus requiring a 39-class classification problem with the "*NONE*" class for ED. We use the official data split provided by the 2015 Event Nugget Detection, including 360 documents for the training dataset and 202 documents for the test dataset.

### 2. Parameters, Resources and Settings

The parameters are tuned on the development data of the ACE 2005 dataset. The selected values for the parameters include the mini-batch size = 50, the pre-defined threshold for the $l_2$ norms = 3, the dropout rate = 0.5, the dimensionality of the position embeddings and the entity type embeddings = 50 and the number of hidden units for the convolution layers $d = 300$. We employ the pre-trained word embeddings with 300 dimensions from (Mikolov et al. 2013) to initialize the word embeddings. These parameters and resources are used for both datasets in this paper.

In order to parse the sentences in the datasets, we employ the Stanford Syntactic Parser with the universal dependency relations. Following the previous work (Nguyen and Grishman 2015; 2016b; Liu et al. 2017), we utilize a fixed length $n = 31$ of sentences in the experiments[5]. This implies that we need to pad the shorter sentences with a special character or trim the longer sentences to fit the fixed length of $n$. While the syntactic edges in the dependency trees for the short sentences can be preserved, we remove the syntactic edges that are linked to at least one trimmed word for the longer sentences.

### 3. Evaluating Network Architectures

This section evaluates different model architectures to demonstrate the effectiveness of GCNs and BiLSTM for GCNs. In particular, we compare the proposed model with its corresponding versions where the GCNs layers or the BiLSTM layers are excluded. For the versions with GCN layers, we incrementally increase the number of graph-based convolution layers (ie. $K$) until the performance drops. Table 1 reports the performance of the models (*Precision* (P), *Recall* (R), and *F-measure* (F1)) on the development portion of the ACE 2005 dataset. Note that the experiments in this section use the proposed pooling mechanism (i.e, entity mention-based pooling *ENTITY*).

There are three blocks in this table. The first block corresponds to the full proposed models (i.e, BiLSTM + GCNs); the second block amounts to the proposed model excluding the BiLSTM layers (i.e, GCNs (no BiLSTM)); and the third block shows the performance of the proposed model when the GCN layers are not included (i.e., only using BiLSTM

[5]This is also the best value on the development data in our case.

| Model | P | R | F1 |
|---|---|---|---|
| BiLSTM + GCNs ($K = 1$) | 81.1 | 63.8 | 71.4 |
| BiLSTM + GCNs ($K = 2$) | 79.8 | 65.2 | **71.8** |
| BiLSTM + GCNs ($K = 3$) | 79.6 | 58.0 | 67.1 |
| GCNs (no BiLSTM) ($K = 1$) | 81.5 | 62.2 | 70.6 |
| GCNs (no BiLSTM) ($K = 2$) | 75.2 | 67.6 | **71.2** |
| GCNs (no BiLSTM) ($K = 3$) | 79.4 | 63.0 | 70.3 |
| BiLSTM | 78.9 | 63.7 | **70.5** |

Table 1: Model performance on the ACE 2005 development dataset for the *ENTITY* pooling method.

layers). Importantly, we optimize the number of BiLSTM layers in this experiment[6] (tuned on the ACE 2005 development dataset) to measure the actual contribution of GCNs for ED in the presence of BiLSTM more accurately.

The table indicates that both the proposed model and the proposed model without BiLSTM (i.e, blocks 1 and 2 respectively) achieve the best performance when the number of GCN layers is 2. The best performance of the former (i.e, the full proposed model "BiLSTM + GCNs ($K = 2$)" with F1 score of 71.8%) is better than the best performance of the latter (i.e, the full proposed excluding BiLSTM "GCNs (no BiLSTM) ($K = 2$)" with F1 score of 71.2%). Consequently, BiLSTM captures some useful dependencies for ED that are not encoded in GCNs. Thus, BiLSTM is complementary to GCNs for ED and the utilization of BiLSTM with GCNs would further improve the performance for GCNs. However, as BiLSTM only adds 0.6% (i.e, from 71.2% to 71.8%) into the performance of GCNs, most of the necessary information for ED has been captured by GCNs themselves. More importantly, comparing the proposed model (i.e, BiLSTM + GCNs ($K = 2$) in block 1 of the table) with the BiLSTM model in block 3, we see that GCNs significantly improve the performance of BiLSTM (i.e, from 70.5% to 71.8%), thus demonstrating the effectiveness of GCNs for ED.

In the following experiments, we would always use the best network architecture for the proposed model discovered in this section, i.e, BiLSTM + GCNs ($K = 2$).

## 4. Evaluating Pooling Mechanisms

In order to show the benefit of the entity mention-based pooling method (*ENTITY*) for GCNs, we compare it with the other pooling methods for ED in the literature (i.e, *ANCHOR* (Nguyen, Cho, and Grishman 2016a; Marcheggiani and Titov 2017), *OVERALL* (Nguyen and Grishman 2015; 2016b), *DYNAMIC* (Chen et al. 2015; 2017) as discussed in the section about pooling above). Specifically, we repeat the model selection procedure in Table 1 of the previous section to select the best network architecture for each pooling method of comparison in {*ANCHOR*, *OVERALL*, *DYNAMIC*} (using the ACE 2005 development dataset). For each pooling method, the selection includes the model with both BiLSTM and GCNs (BiLSTM + GCNs), the model with just GCNs (GCNs (no BiLSTM)) and the model with just BiLSTM (BiLSTM). We also optimize the number of GCN layers and the number of BiLSTM layers for each model as do the previous section. This procedure ensures that each pooling method has its best network architecture to facilitate a fair comparison. The best network architecture for each pooling method and their corresponding performance on the ACE 2005 test set are shown in Table 2.

| Pooling | Best Architecture | F1 |
|---|---|---|
| *ENTITY* | BiLSTM + GCNs ($K = 2$) | **73.1** |
| *ANCHOR* | BiLSTM + GCNs ($K = 3$) | 71.4 |
| *OVERALL* | GCNs (no BiLSTM) ($K = 1$) | 70.8 |
| *DYNAMIC* | GCNs (no BiLSTM) ($K = 2$) | 68.5 |

Table 2: ED performance for pooling mechanisms.

As we can see from the table, the best architectures for *ENTITY* and *ANCHOR* have BiLSTM layers while this is not the case for *OVERALL* and *DYNAMIC* whose best architectures only include GCN layers. We attribute this phenomenon to the fact that both *OVERALL* and *DYNAMIC* aggregate the convolution vectors of every word in the sentences, potentially encapsulating useful long-range dependencies of word in the sentences for ED. This makes BiLSTM redundant as BiLSTM also attempts to capture such long-range dependencies in this case. This is in contrast to *ENTITY* and *ANCHOR* that only aggregate the convolution vectors at some specific positions in the sentences (i.e, the entity mentions and the current word) and lack the capacity to model the long-range dependencies of words. This necessitates BiLSTM to incorporate the long-range dependencies for *ENTITY* and *ANCHOR*. Finally, we see that *ENTITY* significantly outperforms all the other pooling methods ($p < 0.05$) with large margins (i.e, 1.7% better than the second best method of *ANCHOR* in terms of F1 score), demonstrating the effectiveness of the entity mention-based pooling (*ENTITY*) for ED with GCN models.

## 5. Comparing to the State of the art

This section compares the proposed model (i.e, BiLSTM + GCNs ($K = 2$) with *ENTITY* pooling) (called *GCN-ED*) with the state-of-the-art ED systems on the ACE 2005 dataset in Table 3. These systems include:

*1) Perceptron*: the structured perceptron model for joint beam search with both local and global hand-desgined features in (Li, Ji, and Huang 2013)

*2) Cross-Entity*: the cross-entity model (Hong et al. 2011)

*3) PSL*: the probabilistic soft logic model to capture the event-event correlation (Liu et al. 2016a)

*4) Framenet*: the model that leverages the annotated corpus of FrameNet to improve ED (Liu et al. 2016b)

*5) CNN*: the CNN model (Nguyen and Grishman 2015)

*6) DM-CNN*: the dynamic multi-pooling CNN model (Chen et al. 2015)

*7) DM-CNN+*: the dynamic multi-pooling CNN model augmented with automatic labeled data (Chen et al. 2017)

*8) B-RNN*: the bidirectional recurrent neural network model (Nguyen, Cho, and Grishman 2016a)

*9) NCNN*: the nonconsecutive CNN model (Nguyen and Grishman 2016b)

*10) ATT*: the attention-based model (Liu et al. 2017)

[6]The optimal number of BiLSTM layers is 2.

*11) ATT+*: the attention-based model augmented with annotated data in Framenet (Liu et al. 2017)
*12) CNN-RNN*: the ensemble model of CNN and LSTM in (Feng et al. 2016)

| Method | P | R | F1 |
|---|---|---|---|
| *Perceptron* | 73.7 | 62.3 | 67.5 |
| *Cross-Entity* † | 72.9 | 64.3 | 68.3 |
| *PSL* † | 75.3 | 64.4 | 69.4 |
| *Framenet* ‡ | 77.6 | 65.2 | 70.7 |
| *CNN* | 71.8 | 66.4 | 69.0 |
| *DM-CNN* | 75.6 | 63.6 | 69.1 |
| *DM-CNN+* ‡ | 75.7 | 66.0 | 70.5 |
| *B-RNN* | 66.0 | 73.0 | 69.3 |
| *NCNN* | NA | | 71.3 |
| *ATT* | 78.0 | 66.3 | 71.7 |
| *ATT+* ‡ | 76.8 | 67.5 | 71.9 |
| *CNN-RNN* | 84.6 | 64.9 | **73.4** |
| *GCN-ED* | 77.9 | 68.8 | **73.1** |

Table 3: Comparison to the state of the art. †beyond the sentence level. ‡using additional data.

From the table, we see that *GCN-ED* is a single model, but it still performs comparably with the ensemble model *CNN-RNN* in (Feng et al. 2016), and significantly outperforms all the other compared models. In particular, *GCN-ED* is 1.2% better than *ATT+* although *GCN-ED* does not utilize the annotated data from Framenet as *ATT+* does. Besides, although *GCN-ED* only uses the sentence-level information, it is still greatly better than the methods that employ the document-level information (i.e, *Cross-Entity* and *PSL*) with large margins (an improvement of about 4.8% on the F1 score). Finally, among the single convolution-based models (i.e, *CNN*, *DM-CNN*, *NCNN* and *GCN-ED*), *GCN-ED* is superior to the others (an improvement of 1.9% on F1 score with respect to the best reported convolutional model *NCNN*). This is significant with $p < 0.05$ and demonstrates the benefits of the proposed model for ED.

## 6. Investigating the effect of predicted entity mentions

The previous sections have demonstrated the effectiveness of the the proposed model in which the pooling mechanism *ENTITY* plays an important role. The operation of *ENTITY* requires entity mentions that are obtained from the manual annotation (perfect entity mentions) in the previous experiments. It remains to test if the proposed model in general and the pooling method *ENTITY* in particular can still perform well when the entity mentions are predicted by an automatic system. The TAC KBP 2015 is used for the experiments in this section. We first utilize the RPI Joint Information Extraction System (Li, Ji, and Huang 2013; Li et al. 2014b) to label this dataset for entity mentions, and then employ the predicted entity mentions as inputs for the models. In order to ensure consistency, we train the models with the best network architectures for each pooling mechanism in Table 2 on the training portion and report the performance on the test portion of the TAC KBP 2015 dataset. We also use the same hyper-parameters and resources as those of the experiments for the ACE 2005 dataset (in Table 2) to achieve compatibility. Table 4 shows the results. We also include the performance of the best system in the Event Nugget Detection Evaluation of the 2015 Text Analysis Conference for reference (Mitamura, Liu, and Hovy 2015).

| Model | P | R | F1 |
|---|---|---|---|
| *TAC TOP* | NA | | 58.4 |
| *ENTITY* | 70.3 | 50.6 | **58.8** |
| *ANCHOR* | 67.3 | 50.8 | 57.9 |
| *OVERALL* | 71.7 | 48.2 | 57.6 |
| *DYNAMIC* | 69.7 | 48.2 | 57.0 |

Table 4: Performance on the TAC KBP 2015 dataset with predicted entity mentions for the models in Table 2.

The most important observation from the table is that *ENTITY* is still significantly better than the other pooling methods ($p < 0.05$), confirming the effectiveness of entity mentions to specify pooling positions for GCNs in ED even when the entity mentions are predicted. In addition, the proposed model *GCN-ED* (corresponding to the row of *ENTITY* in the table) outperforms the best reported system in the 2015 TAC evaluation, further demonstrating the advantages of *GCN-ED* for ED.

## Related Work

Event detection has attracted much research effort in the last decade. The early and successful approach for ED has involved the feature-based methods that hand-design feature sets for different statistical models for ED (Ahn 2006; Ji and Grishman 2008; Hong et al. 2011; Li, Ji, and Huang 2013; Venugopal et al. 2014; Li et al. 2015a).

The last couple of years witness the success of the neural network models for ED. The typical models employs CNNs (Nguyen and Grishman 2015; Chen et al. 2015; Nguyen et al. 2016c), recurrent neural networks (Nguyen, Cho, and Grishman 2016a; Jagannatha and Yu 2016) and attention-based networks (Liu et al. 2017). However, none of these works consider syntax for neural ED as we do in this work.

Syntactic information has also been employed in neural network models for various natural language processing tasks, including sentiment analysis (Socher et al. 2013; Mou et al. 2015), dependency parsing (Le and Zuidema 2014; Dyer et al. 2015), relation extraction (Li et al. 2015b; Nguyen and Grishman 2016d), machine translation (Eriguchi, Tsuruoka, and Cho 2017) etc. However, this is the first work to integrate syntactic information in the neural network models for event detection.

## Conclusion

We propose a novel neural network model for event detection that is based on graph convolutional networks over dependency trees and entity mention-guided pooling. We extensively compare the proposed models with various baselines and settings, including both perfect entity mention

setting and predicted entity mention setting. The proposed model achieves the state-of-the-art performance on two widely used datasets for ED, i.e, ACE 2005 and TAC KBP 2015. In the future, we expect to investigate the joint models for event extraction (i.e, both event detection and argument prediction) that employ the syntactic structures. We also plan to apply the GCN models to other information extraction tasks such as relation extraction, entity linking etc.

## References


Ahn, D. 2006. The stages of event extraction. In *Proceedings of the Workshop on Annotating and Reasoning about Time and Events*.

Bengio, Y.; Ducharme, R.; Vincent, P.; and Jauvin, C. 2003. A neural probabilistic language model. In *Journal of Machine Learning Research 3*.

Chen, Y.; Xu, L.; Liu, K.; Zeng, D.; and Zhao, J. 2015. Event extraction via dynamic multi-pooling convolutional neural networks. In *ACL-IJCNLP*.

Chen, Y.; Liu, S.; Zhang, X.; Liu, K.; and Zhao, J. 2017. Automatically labeled data generation for large scale event extraction. In *ACL*.

Dyer, C.; Ballesteros, M.; Ling, W.; Matthews, A.; and Smith, N. A. 2015. Transition-based dependency parsing with stack long short-term memory. In *ACL*.

Eriguchi, A.; Tsuruoka, Y.; and Cho, K. 2017. Learning to parse and translate improves neural machine translation. In *arXiv preprint arXiv:1702.03525*.

Feng, X.; Huang, L.; Tang, D.; Ji, H.; Qin, B.; and Liu, T. 2016. A language-independent neural network for event detection. In *ACL*.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. In *Neural Computation*.

Hong, Y.; Zhang, J.; Ma, B.; Yao, J.; Zhou, G.; and Zhu, Q. 2011. Using cross-entity inference to improve event extraction. In *ACL*.

Jagannatha, A. N., and Yu, H. 2016. Bidirectional rnn for medical event detection in electronic health records. In *NAACL*.

Ji, H., and Grishman, R. 2008. Refining event extraction through cross-document inference. In *ACL*.

Kearnes, S.; McCloskey, K.; Berndl, M.; Pande, V.; and Riley, P. 2016. Molecular graph convolutions: Moving beyond fingerprints. In *Journal of computer-aided molecular design 30(8)*.

Kipf, T. N., and Welling, M. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.

Koo, T.; Carreras, X.; and Collins, M. 2008. Simple semi-supervised dependency parsing. In *ACL*.

Le, P., and Zuidema, W. 2014. The inside-outside recursive neural network model for dependency parsing. In *EMNLP*.

Li, Q.; Ji, H.; Hong, Y.; and Li, S. 2014b. Constructing information networks using one single model. In *EMNLP*.

Li, X.; Nguyen, T. H.; Cao, K.; and Grishman, R. 2015a. Improving event detection with abstract meaning representation. In *Proceedings of ACL-IJCNLP Workshop on Computing News Storylines (CNewS)*.

Li, J.; Luong, M.-T.; Jurafsky, D.; and Hovy, E. 2015b. When are tree structures necessary for deep learning of representations? In *arXiv preprint arXiv:1503.00185*.

Li, Q.; Ji, H.; and Huang, L. 2013. Joint event extraction via structured prediction with global features. In *ACL*.

Liu, S.; Liu, K.; He, S.; and Zhao, J. 2016a. A probabilistic soft logic based approach to exploiting latent and global information in event classification. In *AAAI*.

Liu, S.; Chen, Y.; He, S.; Liu, K.; and Zhao, J. 2016b. Leveraging framenet to improve automatic event detection. In *ACL*.

Liu, S.; Chen, Y.; Liu, K.; and Zhao, J. 2017. Exploiting argument information to improve event detection via supervised attention mechanisms. In *ACL*.

Marcheggiani, D., and Titov, I. 2017. Encoding sentences with graph convolutional networks for semantic role labeling. In *EMNLP*.

Mcdonald, R., and Pereira, F. 2006. Online learning of approximate dependency parsing algorithms. In *EACL*.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS*.

Mitamura, T.; Liu, Z.; and Hovy, E. 2015. Overview of tac kbp 2015 event nugget track. In *TAC*.

Mou, L.; Peng, H.; Li, G.; Xu, Y.; Zhang, L.; and Jin, Z. 2015. Discriminative neural sentence modeling by tree-based convolution. In *EMNLP*.

Nguyen, T. H., and Grishman, R. 2015. Event detection and domain adaptation with convolutional neural networks. In *ACL-IJCNLP*.

Nguyen, T. H., and Grishman, R. 2016b. Modeling skip-grams for event detection with convolutional neural networks. In *EMNLP*.

Nguyen, T. H., and Grishman, R. 2016d. Combining neural networks and log-linear models to improve relation extraction. In *Proceedings of IJCAI Workshop on Deep Learning for Artificial Intelligence (DLAI)*.

Nguyen, T. H.; Fu, L.; Cho, K.; and Grishman, R. 2016c. A two-stage approach for extending event detection to new types via neural networks. In *Proceedings of the 1st ACL Workshop on Representation Learning for NLP (RepL4NLP)*.

Nguyen, T. H.; Cho, K.; and Grishman, R. 2016a. Joint event extraction via recurrent neural networks. In *NAACL*.

Socher, R.; Perelygin, A.; Wu, J.; Chuang, J.; Manning, C. D.; Ng, A.; and Potts, C. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*.

Venugopal, D.; Chen, C.; Gogate, V.; and Ng, V. 2014. Relieving the computational bottleneck: Joint inference for event extraction with high-dimensional features. In *EMNLP*.