A Graph Convolutional Network with Adaptive Graph Generation and Channel Selection for Event Detection

Zhipeng Xie,Yumin Tu
School of Computer Science
Shanghai Key Laboratory of Data Science
Fudan University, Shanghai, China
xiezp@fudan.edu.cn

Abstract

Graph convolutional networks have been successfully applied to the task of event detection. However, existing works rely heavily on a fixed syntactic parse tree structure from an external parser. In addition, the information content extracted for aggregation is determined simply by the (syntactic) edge direction or type but irrespective of what semantics the vertices have, which is somewhat rigid. With this work, we propose a novel graph convolutional method that combines an adaptive graph generation technique and a multi-channel selection strategy. The adaptive graph generation technique enables the gradients to pass through the graph sampling layer by using the ST-Gumbel-Softmax trick. The multi-channel selection strategy allows two adjacent vertices to automatically determine which information channels to get through for information extraction and aggregation. The proposed method achieves the state-of-the-art performance on ACE2005 dataset.

Introduction

Event detection (ED) is a fundamental information extraction task in natural language processing, which aims to detect the event triggers and classify them to corresponding event types from given texts. Each event trigger is a word or phrase that identifies an event mention. For example, in the sentence presented in Figure 1, event detection model should identify "***delivered***" to be a trigger word of "*Life:Be-Born*" event, but not "*Movement:Transport*". In the age of deep learning, sequence-based neural methods have been first developed for event detection (Chen et al. 2015; Nguyen, Cho, and Grishman 2016). These methods empower the ED systems with the ability of generalization to unseen words and automatic extraction of task-specific features (Nguyen, Cho, and Grishman 2016), but are confronted with the difficulty of capturing long-range dependencies.

Most recently, graph convolutional approaches have been applied to event detection task (Nguyen and Grishman 2018; Liu, Luo, and Huang 2018; Yan et al. 2019; Lai, Nguyen, and Nguyen 2020; Cui et al. 2020) and achieved state-of-the-art performance, which rely on dependency parse trees as the graph structures. It has been shown that the syntactic connections between words provide effective constraints

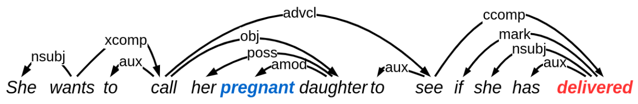

advcl
ccomp
obj
mark
xcomp
poss
nsubj
nsubj
aux
amod
aux
aux
She wants to call her pregnant daughter to see if she has delivered

Figure 1: An example sentence of event type *Life:Be-Born*.

in boosting the performance of event detection (Nguyen and Grishman 2018). However, such a fixed syntactic parse tree structure is unlikely to be the best for the task at hand. For a given trigger candidate, its relevant words are often multi-hop away. For example, for the sentence illustrated in Figure 1, the shortest path between the trigger "delivered" and the relevant word "pregnant" is of length 4: "*delivered–see–call–daughter–pregnant*". It was also reported that more than half of event-related entities need more than one hop to get to the corresponding trigger words (Yan et al. 2019). To capture the multi-hop dependencies, one way is to stack multiple GCN layers (Nguyen and Grishman 2018; Liu, Luo, and Huang 2018; Lai, Nguyen, and Nguyen 2020; Cui et al. 2020), and another way is to extend first-order graph to high-order graphs (Yan et al. 2019). Both ways make it possible that the relevant words can pass useful information through multi-hops to the trigger candidate, but they also introduce more spurious information from irrelevant words, suffering from the so-called "*over-smoothing*" problem (Li, Han, and Wu 2018). Therefore, it is desirable that the graph structure is adaptive such that the relevant words are only one-hop from the trigger candidate, and thus a single graph convolutional layer will be sufficient to retrieve enough information from the one-hop neighborhood for effective event detection. However, in existing GCN-based event detection methods as shown in Figure 2a, graph structures are usually syntactic dependency parse trees generated by an external off-the-shelf dependency parser. The process of graph construction is not differentiable and cannot be optimized specifically for the event detection task by gradient-based optimization techniques. Additionally, the whole system is pipelined and the performance of event detection may suffer from the errors propagated from the external parser.

On the other hand, let us investigate how a GCN network extracts and aggregates information from its neighborhood to produce a global representation for each trigger candidate. In existing GCN-based event detection methods, the information content extracted for aggregation is simply determined by the (syntactic) edge direction or type (Nguyen and Grishman 2018; Liu, Luo, and Huang 2018). Considering that there are usually 3 edge types (*along*, *rev* and *self-loop*), they associate a linear transformation with each edge type, as its information extractor which is called an *information channel* in this paper. Specifically, the information channels are selected simply according to syntactic relationships (instead of semantic ones) for extracting relevant information. Such a semantics-agnostic manner is a little bit rigid.

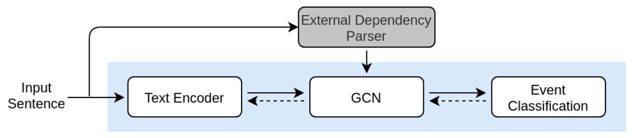
To overcome these problems, this paper proposes AGGED, an event detection method based on adaptive graph generation, which employs a single graph convolutional layer (instead of stacked multiple layers) to aggregate information for event detection. Its modular structure is depicted in Figure 2b. The AGGED method features an adaptive graph generation module and a gated-multi-channel graph convolutional (GMC-GCN) mechanism:

- Firstly, the *adaptive graph generation* module is in charge of building up a probabilistic head-selection matrix, from which a random graph can be sampled. To make the graph sampling process be differentiable, we employ the ST-Gumbel-Softmax trick such that the gradients can backpropagate through it.
- Secondly, the *gated-multi-channel* mechanism is responsible for information extraction and aggregation on the generated graph by two techniques: the *multi-channel selection* allows two adjacent vertices to automatically determine which information channels to get through, and the *information gating* modulates the amount of information to be passed.

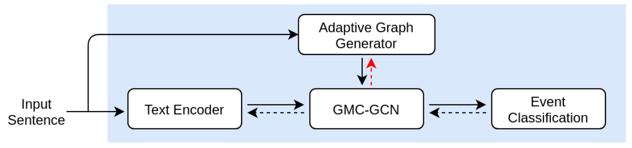
Put colloquially, sampling a graph from probabilistic head-selection matrix is analogous to building up a highway, and multi-channel selection determines the content to be transmitted on the highway, where information gating controls the amount of the transmitted content. To the best of our knowledge, this is the first work that simultaneously learns the graph structure and the parameters of a graph convolutional network for event detection, or even for natural language processing.

Related Work

Due to the fact that syntactic relations can capture the mutual relationship among triggers and related entities, Nguyen & Grishman (2018) and Liu *et al.* (2018) are the first to apply GCN on dependency trees where syntactic relations are embodied as directed arcs. In terms of graph structures, (Yan et al. 2019) extended first-order syntactic graph to high-order ones and used a graph attention mechanism to calculate the multi-order representations, in order to overcome the over-smoothing problem in stacking multiple graph convolutional layers. In terms of information flow on the graph, (Cui et al. 2020) used additional dependency label information, under the consideration that dependency labels can be important for indicating triggers; (Lai, Nguyen, and Nguyen 2020) designed a gate mechanism based on the trigger candidate to filter the noise and let only relevant information to pass through.



(a) Traditional GCN-based ED methods



(b) The proposed AGGED method

Figure 2: Modular structures of GCN-based ED methods, where the solid arrows denote the forward pass of information, and the dashed arrows denotes the backpropagation of gradients.

With the help of dependency parse trees, these GCN-based methods have achieved state-of-the-art performance on the event detection task. However, the parse trees are fixed and static, which can not be adjusted with respect to the downstream learning task. The problem we are facing is how to adaptively learn the graph structure together with the downstream GCN network in an end-to-end manner.

Little attention has been paid to this problem. To the best of our knowledge, there are only two exceptions. (Franceschi et al. 2019; Kazi et al. 2020). Franceschi *et al.* (2019) made the first attempt to simultaneously learn the graph structure and a GCN network for semi-supervised classification. Kazi *et al.* (2020) introduced a differentiable graph module as a learnable predictor of edge probabilities and applied it to several tasks such as disease prediction and gender classification.

Method

The overall AGGED architecture is illustrated in Figure 3. Let $S = [w_1, w_2, ..., w_n]$ be an input sentence of length n. We use $\mathbf{w}_i$ and $\mathbf{p}_i$ to denote the word embedding and the POS-tag embedding of the i-th token w_i, respectively. Following previous work (Chen et al. 2018), the word embeddings are pretrained by Skip-Gram algorithm on the NYT Corpus. We obtain the part-of-speech tags by using the Stanford CoreNLP (Manning et al. 2014). The POS-tag embedding matrix is randomly initialized. Thereupon, in the input layer, each token w_i is represented as a vector $\mathbf{x}_i = [\mathbf{w}_i; \mathbf{p}_i]$ by concatenating its word and POS-tag embeddings.

Adaptive Graph Generation Module

Given an input sentence S, a dependency graph is a directed graph where vertices are the word tokens in S, and a directed edge $w_j \rightarrow w_i$ (or (w_i, w_j)) going from the head word w_j to the dependent word w_i indicates there is some (semantic or syntactic or both) dependency relation from w_i to w_j.

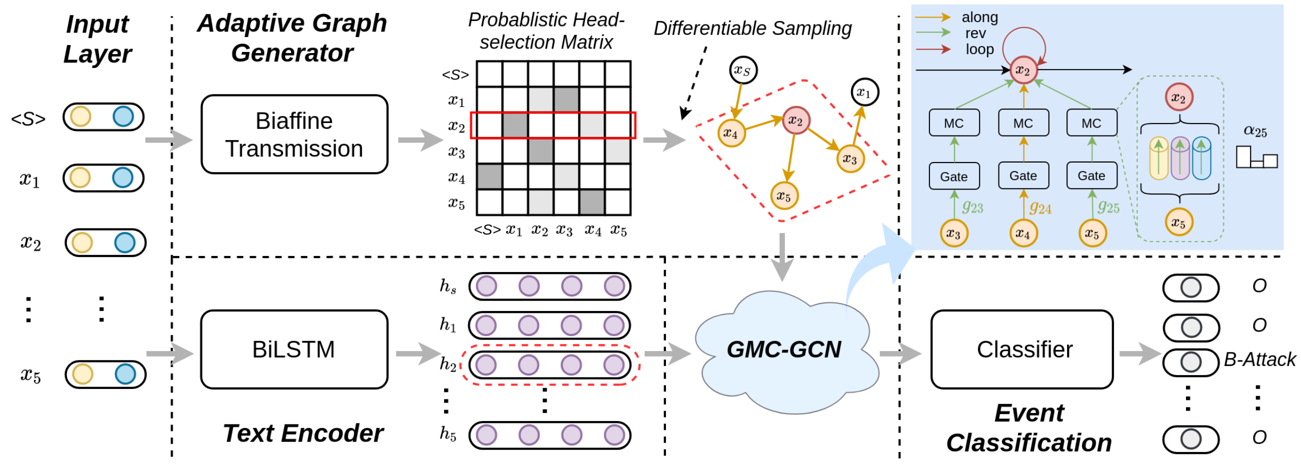


Figure 3: Model architecture. After the input layer, the initial vector representation of input sentence is fed into two modules: one is the adaptive graph generator that outputs a discrete graph structure based on Deep Biaffine Attention and Gumbel-Softmax sampling; the other is the text encoder implemented by a BiLSTM network that generates node embeddings (or equivalently, the contextualized embeddings of words). In turn, both the graph structure and the node embeddings flow into the proposed GMC-GCN module for aggregating information from one-hop neighborhoods. Finally, the updated representations are used for event classification as sequence labeling.

To limit the complexity, this paper is confined to 1-indegree graphs where each vertex has one and only one incoming edge.

The graph generation module is to learn a dependency graph with regards to the event detection task (that is, a task-specific graph structure) in the gradient-based optimization framework. To do so, the module consists of two components:

- The first one learns a probabilistic head-selection (PHS) matrix $\mathbf{P}$, in which each row $\mathbf{p}_i$ corresponds to a word w_i and denotes the probability distribution for selecting its head;
- The second component employs the so-called "ST-Gumbel-Softmax" trick to sample a graph from the PHS matrix, where the $\arg\max$ is used in the forward pass and the gradients are approximated by the normal Gumbel Softmax in the backward pass.

Deep biaffine attention for PHS matrix. As shown in Figure 3, the PHS matrix is obtained by using the neural network architecture of the Deep Biaffine Attention (DBA) parser proposed by (Dozat and Manning 2017). After receiving the input sequence of vectors $[\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]$, a stacked BiLSTM first converts it to the context representation $[\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_n]$.

Next, two multi-layer perceptrons(MLPs) are used to obtain the representations of word w_i being the head or the dependent in any dependency relation:

$$\mathbf{h}_i^{\text{head}} = \mathbf{MLP}_{\text{arc}}^{\text{head}}(\mathbf{r}_i) \tag{1}$$

$$\mathbf{h}_i^{\text{dep}} = \mathbf{MLP}_{\text{arc}}^{\text{dep}}(\mathbf{r}_i) \tag{2}$$

Given a dependent word w_i, the scores of each possible dependency relation from w_i to any possible head are calculated as:

$$\mathbf{s}_i = \left(\mathbf{H}^{\text{head}}\right)^\top \mathbf{U}_{\text{arc}} \mathbf{h}_i^{\text{dep}} + \left(\mathbf{H}^{\text{head}}\right)^\top \mathbf{v}_{\text{arc}} \tag{3}$$

where $\mathbf{H}^{\text{head}} = [\mathbf{h}_i^{\text{head}}]_{1 \le i \le n}$ is the head representations of all the words, $\mathbf{U}_{\text{arc}}$ is a parameter matrix and $\mathbf{v}_{\text{arc}}$ is a parameter vector.

Finally, we obtained the probabilistic head selection matrix $\mathbf{P}$, with each element $p_{i,j}$ to be:

$$p_{i,j} = \frac{\exp(s_{i,j})}{\sum_{k=1}^{n} \mathbb{I}_{[k \neq i]} \exp(s_{i,k})} \tag{4}$$

where $\mathbb{I}_{[k \neq i]} \in \{0, 1\}$ is an indicator function that equals 1 only if $k \neq i$ and $p_{i,j}$ represents the probability of word w_j being the head of word w_i.

ST-Gumbel-Softmax trick for sparse-graph sampling. *Deterministic Sparse Graphs.* Based on the probabilistic head selection matrix, an arc can be generated by choosing the highest scoring head for each dependent node, as done at the training time of graph-based dependency parsing neural models (Dozat and Manning 2017). Or instead, the Maximum Spanning Tree (MST) algorithm can be applied to generate a well-formed parse tree structure, as done at the testing time. However, both the graph-generation methods prevent gradients from passing through. As to the task of event detection, all the existing graph convolution based methods (Nguyen and Grishman 2018; Liu, Luo, and Huang 2018) rely on such a fixed parse tree structure from dependency parsers.

Fully-Connected Probabilistic Graphs. The probabilistic head selection matrix can be thought of as a fully-connected weighted graph structure, which can be directly passed to the downstream task. However, this fully-connected-graph based method has several shortcomings compared with parse-graph-sampling method (Franceschi et al. 2019). Firstly, the evaluation of sparse GCN on sampled graphs is much faster than the evaluation of dense GCN on fully-connected graphs. Secondly, the downstream event detection module is usually nonlinear, which makes the two methods have different results. Thirdly, graph-sampling method is more intuitive to interpret than a dense probabilistic matrix.

Stochastic Sparse-Graph. This paper adopts the approach of sparse-graph-sampling and employs the ST-Gumbel-Softmax trick (Jang, Gu, and Poole 2017) to reparameterize the discrete choice of a $(n-1)$-way categorical variable. For each word w_i, we randomly sample a head according to $\mathbf{p}_i$, and its one-hot representation is:

$$\mathbf{a}_i = \text{one-hot}\left(\underset{k \neq i}{\arg\max}[\log p_{i,k} + g_{i,k}]\right) \tag{5}$$

where $g_{i,k}$ ($1 \leq k \leq n$ and $k \neq i$) is i.i.d sampled from $\text{Gumbel}(0,1)$ [1]. Let ϕ stand for the set of all the parameters in the deep biaffine module for calculating PHS matrices. In the backward pass, by using the softmax function as a continuous, differential approximation to the $\arg\max$ in Equation 5, the ST-Gumbel-Softmax trick approximates $\mathbf{a}_i$ with $\hat{\mathbf{a}}_i = [\hat{a}_{i,1}, \ldots, \hat{a}_{i,n}]$, and thus obtains a continuous approximation of the gradients $\nabla_\phi \mathbf{a}_i \approx \nabla_\phi \hat{\mathbf{a}}_i$:

$$\hat{a}_{i,k} = \frac{\exp((\log p_{i,k} + g_{i,k})/\tau)}{\sum_{k'=1}^{n} \mathbb{I}_{[k' \neq i]} \exp((\log p_{i,k'} + g_{i,k'})/\tau)} \tag{6}$$

where τ is the temperature parameter.

During testing time, each dependent word w_i find its head as $w_j : j = \arg\max_{k \neq i} p_{i,k}$. That is, the word w_j with the highest probability is deterministically chosen as the head.

Gated-Multi-Channel GCN

A graph convolutional network (GCN) (Kipf and Welling 2017) usually takes two inputs: a graph structure $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $\mathcal{V}$ as the vertex set and $\mathcal{E}$ as the edge set, and a feature matrix $\mathbf{H}$ of the vertex set, where the ith column $\mathbf{h}_i$ of $\mathbf{H}$ is the vector representation of the ith vertex $v_i \in \mathcal{V}$. The feature matrix $\mathbf{H}$ is usually obtained from the initial embedding representations $\mathbf{x}_i$ ($1 \leq i \leq n$) by using a stacked-BiLSTM network. In turn, the GCN constructs a new representation $\mathbf{h}_i^{\text{conv}}$ for each vertex v_i by extacting and aggregating information from its adjacent vertices (*i.e.*, by allowing information to flow among vertices through edges).

The subsection is devoted to a so-called *gated-multi-channel* (GMC) mechanism to control the information flow among vertices. Before delving into the details of GMC, let us begin with some background knowledge of vanilla graph convolutional network for event detection.

Vanilla GCN for event detection. For event detection, a vanilla GCN uses dependency parse tree as the graph structure $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. To allow the convolution for each token w_i in $\mathcal{G}$ to involve the word w_i itself as well as its head word, the self-loops (w_i, w_i) and the inverse edges (w_j, w_i) for each $(w_i, w_j) \in \mathcal{E}$ are also added into the initial edge set $\mathcal{E}$, resulting in an extended edge set $\mathcal{E}^*$. Previous GCN-based ED methods (Nguyen and Grishman 2018; Liu, Luo, and Huang 2018; Yan et al. 2019) operate on the extended graph $\mathcal{G}^* = (\mathcal{V}, \mathcal{E}^*)$. Each edge $(w_i, w_j) \in \mathcal{E}^*$ is assigned an edge type as follows:

$$\gamma_{i,j} = \gamma(w_i, w_j) = \begin{cases} 0 & \text{if } i = j \text{ (self-loop)} \\ 1 & \text{if } (w_i, w_j) \in \mathcal{E} \text{ (along)} \\ 2 & \text{if } (w_j, w_i) \in \mathcal{E} \text{ (rev)} \end{cases} \tag{7}$$

The information aggregation equation that computes the graph convolution vector $\mathbf{h}_i^{\text{conv}}$ for a vertex w_i is:

$$\mathbf{h}_i^{\text{conv}} = f\left(\sum_{(w_i,w_j)\in\mathcal{E}^*} \text{IC}_{\gamma_{i,j}}(\mathbf{h}_j)\right) \tag{8}$$

where the activation function $f(\cdot)$ is ReLU, and each *information channel* IC_c $(0 \leq c \leq 2)$, characterized by a matrix $\mathbf{M}_c$ and a vector $\mathbf{b}_c$, defines what information content should be extracted from a vertex and be passed:

$$\text{IC}_c(\mathbf{h}_j) = \mathbf{W}_c \mathbf{h}_j + \mathbf{b}_c \tag{9}$$

Clearly, exiting methods (Nguyen and Grishman 2018; Liu, Luo, and Huang 2018) make use of three information channels: one for *along* edges, one for *rev* edges, and the other for *loop* edges. Given a vertex w_i and one of its neighbors w_j in $\mathcal{G}^*$, the information extracted and aggregated is determined solely by the edge type of (w_i, w_j), which is agnostic of the semantic of w_i and is somewhat rigid.

Gated-Multi-Channel GCN. Once a graph structure $\mathcal{G}$ is generated (or sampled) by the adaptive graph generation module described in Section , the proposed GMC-GCN works on it as follows.

In GMC-GCN, there are $(C+1)$ information channels, $\{\text{IC}_c : 0 \leq c \leq C\}$. As defined in Equation 9, each information channel can be thought of as an information extractor. For a *self-loop* edge (w_i, w_i), we extract the information $\text{IC}_0(\mathbf{h}_i)$ for aggregation by the information channel IC_0, which is the same as in the vanilla GCN. For an *along* or *rev* edge (w_i, w_j), the information to be extracted from w_j and aggregated to w_i is calculated by a multi-channel selection mechanism based on the following biaffine attention:

$$\boldsymbol{\alpha}_{i,j} = \text{softmax}\left(\mathbf{h}_i^\top \mathbf{U}_s^{\gamma_{i,j}} \mathbf{h}_j + (\mathbf{h}_i \oplus \mathbf{h}_j)^\top \mathbf{M}_s^{\gamma_{i,j}} + \mathbf{b}_s^{\gamma_{i,j}}\right) \tag{10}$$

where $\boldsymbol{\alpha}_{i,j} \in \mathbb{R}^C$ represents the attention distribution over the C information channels, $\mathbf{U}_s^1, \mathbf{U}_s^2 \in \mathbb{R}^{d\times C\times d}$, $\mathbf{M}_s^1, \mathbf{M}_s^2 \in \mathbb{R}^{(2d)\times C}$ and $\mathbf{b}_s^1, \mathbf{b}_s^2 \in \mathbb{R}^C$ are parameters. Clearly, the channel selection depends not only on the edge type, but also the semantics of two vertices. In this way, each vertex will aggregate the information contents from its neighboring vertices, which are extracted by automatically selecting the information channels:

$$\mathbf{h}_i^{\text{conv}} = f\left(\text{IC}_0(\mathbf{h}_i) + \sum_{\substack{(w_i,w_j)\in\mathcal{E}^* \\ j\neq i}} \sum_{c=1}^{C} \alpha_{i,j,c} \cdot \text{IC}_c(\mathbf{h}_j)\right) \tag{11}$$

[1] $g_{i,k} = -\log(-\log(u_{i,k}))$ and $u_{i,k} \sim \text{Uniform}(0,1)$

The graph generation module tries to adaptively capture task-specific dependency relations among tokens and thus forms their neighborhoods, and the multi-channel mechanism empowers our model the ability to adaptively choose the appropriate information channels to extract the information content to get aggregated. Last but not the least, we also employ a gating mechanism to modulate the amount of information content to be passed and aggregated, for two following reasons. Firstly, different neighbors should have different influences on a given vertex, and it is unfair to assign them the same weight. Secondly, such a gating mechanism is also helpful to correct possible mistakes made in the graph generation phase.

$$g_{i,j} = \sigma\left(\left(\mathbf{h}_i \oplus \mathbf{h}_j\right)^\top \mathbf{v}_g + b_g\right) \tag{12}$$

where σ is the sigmoid function, $\mathbf{v}_g$ is a vector and b_g is a scalar.

Combined with the information gate and multi-channel mechanism, information aggregation equation (11) can be rewritten as follows:

$$\mathbf{h}_i^{\text{conv}} = f\left(\text{IC}_0(\mathbf{h}_i) + \sum_{\substack{(w_i,w_j)\in\mathcal{E}^* \\ j\neq i}} g_{i,j} \sum_{c=1}^{C} \alpha_{i,j,c} \cdot \text{IC}_c(\mathbf{h}_j)\right) \tag{13}$$

Please note that the multi-channel selection mechanism is applied only to the *along* and *rev* edges.

Event Classification

We treat event detection as a classification task using the traditional BIO scheme. The aggregated representation $\mathbf{h}_i^{\text{conv}}$ of each token w_i in S is fed into a fully-connected layer with the softmax activation function to calculate its probability distribution $\mathbf{y}_i$ on all the label types:

$$\mathbf{y}_i = \text{softmax}\left(\mathbf{W}_o \mathbf{h}_i^{\text{conv}} + \mathbf{b}_o\right) \tag{14}$$

where $\mathbf{y}_i \in \mathbb{R}^{2N_e+1}$, and N_e denotes the number of event types.

Training

Model optimization phase. On ACE2005 dataset, we train the proposed event detection model to minimize the cross-entropy loss function:

$$L = -\sum_{S\in\mathcal{D}} \sum_{i=1}^{n_S} \log y_{i,t_i} \tag{15}$$

where $\mathcal{D}$ is the train dataset, n_S is the length of sentence S and t_i denotes the gold label for the ith word in S.

Burn-in phase. We find that a good initial PHM matrix can be helpful to obtain better performance. For this reason, before the model optimization phase above, we use a "burn-in" phase for the Deep Biaffine Attention module, where it is trained as a part of the DBA dependency parser developed in (Dozat and Manning 2017), on the Universal Dependencies English Web Treebank v2.6 [2].

[2]https://github.com/UniversalDependencies/UD_English-EWT

Experimental Results

Experiment Settings

Dataset, resources and evaluation metrics. We perform experiments on ACE2005 dataset, which is composed of 599 documents and includes 33 predefined event types. For comparison, we follow the data split of previous works (Li, Ji, and Huang 2013; Liu, Luo, and Huang 2018), where 529 documents (14956 sentences) are used as the train set, 30 documents (851 sentences) as the development set, and 40 documents (668 sentences) as the test set.

The pretrained word embedding library of dimension 100 is from (Chen et al. 2018)[3], which was trained by using Skip-Gram algorithm on the NYT corpus. To "burn-in" the Deep Biaffine Attention module, we use the Universal Dependencies English Web Treebank v2.6 to train the dependency parser.

Finally, we use micro-averaged *Precision* (P), *Recall* (R) and *F1 score* (F1) as the evaluation metrics, the same as previous works. All the reported experimental results are averaged on 10 runs.

Hyperparameter setting. For the Graph Generation Module, we basically follow the config of original paper (Dozat and Manning 2017). For other parameters of the model, we tune them according to the development set. And the final setting is as follows:

As for the model training, we use Adam algorithm (Kingma and Ba 2015) as the optimizer and set the learning rate to 10^{-3}. For the DBA network, the learning rate is set to 2×10^{-4} in the model optimization phase. The duration of the "burn-in" phase is set to 4 epochs. The batch size is set to 30. Because of the overfitting problem, we also use the early stop strategy.

The dimension of word embeddings is 100, and the dimension of POS-tag embeddings is 50. The numbers of hidden states of BiLSTM layer and GMC-GCN are set to 300. The temperature parameter of the *ST-Gumbel-Softmax* trick is set to 1.0. The number of information channels is set to 3, which is tuned on the development set.

Overall Performance

We compare our method with the following state-of-the-arts:

- **GCN-ED**. (Nguyen and Grishman 2018) used GCN first to learn context representation for event detection and propose argument-aware pooling.
- **JMEE**. (Liu, Luo, and Huang 2018) applied highway connection and self-attention pooling to improve context information aggregation.
- **MOGANED**. (Yan et al. 2019) adopted multi-order graphs to integrate information rather than just 1-order graph.
- **EE-GCN**. (Cui et al. 2020) took dependency label information into consideration during graph convolution.
- **DMBERT**. (Wang et al. 2019) used adversarial training to expand more data from unstructured text.

[3]https://github.com/yubochen/NBTNGMA4ED/blob/master/100.utf8

Model	P	R	F1
GCN-ED (Nguyen and Grishman 2018)	77.9	68.8	73.1
JMEE (Liu, Luo, and Huang 2018)	76.3	71.3	73.7
MOGANED (Yan et al. 2019)	**79.5**	72.3	75.7
EE-GCN (Cui et al. 2020)	76.7	78.6	77.6
DMBERT (Wang et al. 2019)	77.9	72.5	75.1
SS-VQ-VAE (Huang and Ji 2020)	75.7	77.8	76.7
GatedGCN (Lai, Nguyen, and Nguyen 2020)	78.8	76.3	77.6
EKD (Tong et al. 2020)	79.1	78.0	78.6
Our AGGED Method	77.8	**82.2**	**79.9**

Table 1: Performance comparison with existing event detection methods.

Model	P	R	F1
Full AGGED Model	77.8	82.2	79.9
–MC	77.9	81.0	79.4
–IG	77.5	80.0	78.7
–MC & IG	77.8	80.3	79.0
–AG	76.8	79.9	78.3

Table 2: Ablation study on ACE test set.

- **SS-VQ-VAE**. (Huang and Ji 2020) employed VAE to learn the latent event type representation.
- **GatedGCN**. (Lai, Nguyen, and Nguyen 2020) combined the gate, diversity and structure consistency to integrate syntactic information.
- **EKD** (Tong et al. 2020) introduced open-domain trigger knowledge to improve event detection.

Table 1 lists the results. We find that our method achieves substantial gain on the recall, and maintains a satisfactory precision value competitive to the state-of-the-arts. It can be seen that our method achieves a new state-of-the-art result on both Recall and F1 score, with 3.6% and 1.3% improvement respectively. The MOGANED method achieves the best precision result that is 1.9% higher than the precision of ours, but its recall is nearly 10% lower than our method. Such a high recall may be credited to the effective information aggregation mechanism of our method. On the one hand, the adaptive graph generation module can generate task-specific graph structure which eases the information pass between the trigger candidate and its relevant words. On the other hand, gated-multi-channel mechanism modulates the information content and its amount to be passed.

Ablation Study

To check the impacts of the adaptive graph generation module and the gated-multi-channel mechanism, we make the following ablation study by removing each component from the full method. Table 2 presents the results of 4 situations:

- –MC: multi channels are removed before information propagation, which means information just flows along the specific direction. The rigid restriction results in a 0.5% drop of F1 score, demonstrating the rationality of the multi channels.

	P	R	F1
without burn-in	75.7	80.3	77.9
with burn-in	77.8	82.2	79.9

Table 3: Performance of our method with or without the burn-in phase for the adaptive graph generation module

- –IG: We remove the information gate. In this scenario, for each trigger candidate, all its adjacent nodes can propagate their information to it without control. The possible noise in the previous stages may also interfere with the information extraction and aggregation. As a result, F1 score drops by 1.2%.
- –IG & MC: We remove both the multi-channel selection and the information gate. An interesting observation is that the F1 score is 0.3% higher than removing only the information gate. We conjecture that the multi-channel selection is highly dependent on the information gate, for the information gating can effectively control the additional complexity introduced by the multi-channel mechanism. Without information gates, the multi-channel can not work well alone,
- –AG: We remove the graph generation module, and simply use the dependency parse tree as the static graph structure. It is observed that the F1 score drops by 1.6%, which manifests that the task-specific dependency graph plays a most important role in the success of our method.

Finally, in all these situations, the performance gets lower than the full model, which suggests that both the adaptive graph generation module and the gated-multi-channel mechanism play indispensable roles in our event detection method.

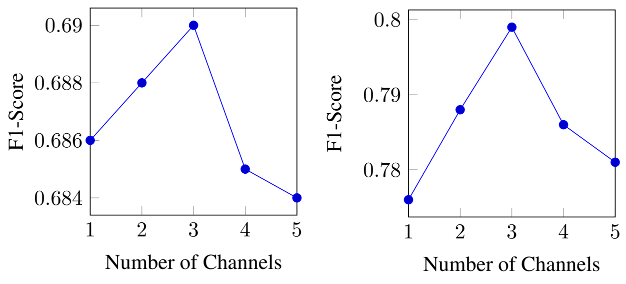
Effect of "Burn-in" Phase

As a fundamental module, adaptive graph generator works on the basis of the probabilistic head-selection matrix that is the output of a Deep Biaffine Attention network. To initialize the DBA network, there are two alternatives: one is to randomly initialize its parameters, and the other is to burn-in the DBA network as part of a dependency parser induction task. We test the performance of our method in these two situations, and present the results in Table 3. Without the burn-in phase, it is observed that F1 score drops about 2.0%. The possible reason is that the burn-in phase sets the adaptive graph generation module to a good initial point, which makes the subsequent task-specific adjustment much easier.

We also examine the induced graph structures from the adaptive graph generation module. Without burn-in, the words in a sentence tend to link to the end of the sentence, because the randomly initialized probabilistic head-selection matrix makes each word more likely interact with the most informative node(*e.g.* the end of the sentence). It explains the performance drop when randomly initializing the DBA module without burn-in.

	Sentence	Prediction	Ground-Truth
Dependency Parser	Mirjana Markovic, the power behind the scenes during Milosevic's 13-year reign, is *accused of illegally* **providing** their grandson's *nanny* with a state-owned luxury apartment in Belgrade in *2000*.	✗ None	Transfer-Ownership
Adaptive Generator	Mirjana Markovic, the power behind the scenes during Milosevic's 13-year reign, is accused *of illegally* **providing** their grandson's *nanny* with a state-owned luxury *apartment* in *Belgrade* in *2000*.	✓ Correct	

Table 4: A case study of the graph structures from the dependency parser and the adaptive graph generator, where the trigger word "*providing*" is marked in bold font, and its adjacent words in the graph structures are italicized and underlined.



(a) On validation data

(b) On test data

Figure 4: Event dection performance with respect to the number of information channels

Effect of the Number of Information Channels

In this section, we study how the number of information channels affect the event detection performance. The F1 scores on the validation and the test data are shown in Figure 4. We can see that our method achieves the best performance on both the validation and the test datasets when the number of information channels is set to 3. As the number of channels increases, the performance first increases and then begins to drop. It can be explained as follows: when the number of channels is less than 3, the channels are fewer than necessary for capturing relevant information; but after the number of channels exceeds 3, there is a risk of overfitting so that the generalization performance of the model gets hurt.

Case Study

To give an intuitive understanding of the adaptive dependency graph generated by our method, a simple case study is shown in Table 4. In the example sentence, there is a mention of the event type *Transfer-Ownership*, which is indicated by the trigger word "*providing*"(marked in red color). This event is also accompanied with five relevant arguments: "*Mirjana Markovic*", "*nanny*", "*apartment*", "*Belgrade*" and "*2000*", which play the roles of "*Seller*", "*Beneficiary*", "*Artifact*", "*Place*" and "*Time-within*" respectively.

The parse tree generated by the Stanford CoreNLP dependency parser is shown in the row of "Dependency Parser", where the words with direct syntactic relationship with the trigger "*providing*" are marked in blue color. We find that only two of the five relevant arguments, "*nanny*" and "*2000*", have a direct syntactic relationship with the trigger, and thus are one-hop away. With this dependency parse tree as the graph structure, the word "*providing*" is wrongly predicted as "*None*", possibly because the single graph convolutional layer fails to capture sufficient relevant information for make the correct prediction.

With our adaptive graph generation module, as shown in the row of "Adaptive Generator", it is observed that four of the five arguments (except "*Mirjana Markovic*") have direct links to the trigger word "*providing*". It means that relevant information from the four arguments can be aggregated into the final representation of the word "*providing*" by a single graph convolutional layer, which leads to the correct prediction "*Transfer-Ownership*" made by our method. In addition, it is worth noting that the word "*providing*" does not appear as a trigger in the train set, but our method does still detect it correctly in the test set.

Conclusions

To summarize, this paper makes an attempt to simultaneously learn the graph structure and the downstream graph convolutional network in an end-to-end manner for the event detection task. The proposed AGGED method features two novel components: (1) an adaptive graph generation module that enables the gradients of errors from the downstream task to backpropagate through, and (2) a gated-multi-channel GCN module that couples the multi-channel selection and the information gating mechanisms together for information extraction and aggregation on the graph. It achieves the SOTA performance on ACE2005 dataset. Future works may include:

- More advanced network structure for generating graphs of controlled sparsity, such as employing reinforcement learning to obtained task-specific paths; and
- Application of the proposed adaptively-generated networks to other NLP tasks, such as entity and relation recognition.

Acknowledgments

This work is supported by National Key Research and Development Program of China (No.2018YFB1005100) and National Natural Science Foundation of China (No.62076072). We are grateful to the anonymous reviewers for their valuable comments.

References

Chen, Y.; Xu, L.; Liu, K.; Zeng, D.; and Zhao, J. 2015. Event Extraction via Dynamic Multi-Pooling Convolutional Neural Networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, Volume 1: Long Papers*, 167–176.

Chen, Y.; Yang, H.; Liu, K.; Zhao, J.; and Jia, Y. 2018. Collective Event Detection via a Hierarchical and Bias Tagging Networks with Gated Multi-level Attention Mechanisms. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 1267–1276.

Cui, S.; Yu, B.; Liu, T.; Zhang, Z.; Wang, X.; and Shi, J. 2020. Edge-Enhanced Graph Convolution Networks for Event Detection with Syntactic Relation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, 2329–2339.

Dozat, T.; and Manning, C. D. 2017. Deep Biaffine Attention for Neural Dependency Parsing. In *Proceedings of the 5th International Conference on Learning Representations*.

Franceschi, L.; Niepert, M.; Pontil, M.; and He, X. 2019. Learning Discrete Structures for Graph Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning*, 1972–1982.

Huang, L.; and Ji, H. 2020. Semi-supervised New Event Type Induction and Event Detection. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, 718–724.

Jang, E.; Gu, S.; and Poole, B. 2017. Categorical Reparameterization with Gumbel-Softmax. In *Proceedings of the 5th International Conference on Learning Representations*.

Kazi, A.; Cosmo, L.; Navab, N.; and Bronstein, M. M. 2020. Differentiable Graph Module (DGM) Graph Convolutional Networks. *CoRR*, abs/2002.04999.

Kingma, D. P.; and Ba, J. 2015. Adam: A Method for Stochastic Optimization. In *Proceedings of the 3rd International Conference on Learning Representations*.

Kipf, T. N.; and Welling, M. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the 5th International Conference on Learning Representations*.

Lai, V. D.; Nguyen, T. N.; and Nguyen, T. H. 2020. Event Detection: Gate Diversity and Syntactic Importance Scores for Graph Convolution Neural Networks. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, 5405–5411.

Li, Q.; Han, Z.; and Wu, X. 2018. Deeper Insights Into Graph Convolutional Networks for Semi-Supervised Learning. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, 3538–3545.

Li, Q.; Ji, H.; and Huang, L. 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*, 73–82.

Liu, X.; Luo, Z.; and Huang, H. 2018. Jointly Multiple Events Extraction via Attention-based Graph Information Aggregation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 1247–1256.

Manning, C. D.; Surdeanu, M.; Bauer, J.; Finkel, J.; and Mcclosky, D. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*.

Nguyen, T. H.; Cho, K.; and Grishman, R. 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*, 300–309.

Nguyen, T. H.; and Grishman, R. 2018. Graph Convolutional Networks With Argument-Aware Pooling for Event Detection. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, 5900–5907.

Tong, M.; Xu, B.; Wang, S.; Cao, Y.; Hou, L.; Li, J.; and Xie, J. 2020. Improving Event Detection via Open-domain Trigger Knowledge. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 5887–5897.

Wang, X.; Han, X.; Liu, Z.; Sun, M.; and Li, P. 2019. Adversarial Training for Weakly Supervised Event Detection. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 998–1008.

Yan, H.; Jin, X.; Meng, X.; Guo, J.; and Cheng, X. 2019. Event Detection with Multi-Order Graph Convolution and Aggregated Attention. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, 5765–5769.