GTN-ED: Event Detection Using Graph Transformer Networks

Sanghamitra Dutta[*]
Carnegie Mellon University
sanghamd@andrew.cmu.edu

Liang Ma
Dataminr
lma@dataminr.com

Tanay Kumar Saha
Dataminr
tsaha@dataminr.com

Di Lu
Dataminr
dlu@dataminr.com

Joel Tetreault
Dataminr
jtetreault@dataminr.com

Alejandro Jaimes
Dataminr
ajaimes@dataminr.com

Abstract

Recent works show that the graph structure of sentences, generated from dependency parsers, has potential for improving event detection. However, they often only leverage the edges (dependencies) between words, and discard the dependency labels (e.g., nominal-subject), treating the underlying graph edges as homogeneous. In this work, we propose a novel framework for incorporating both dependencies and their labels using a recently proposed technique called Graph Transformer Networks (GTN). We integrate GTNs to leverage dependency relations on two existing homogeneous-graph-based models, and demonstrate an improvement in the F1 score on the ACE dataset.

1 Introduction

Event detection is an important task in natural language processing, which encompasses predicting important incidents in texts, e.g., news, tweets, messages, and manuscripts (Yang and Mitchell, 2016; Nguyen et al., 2016; Feng et al., 2016; Zhang et al., 2020; Du and Cardie, 2020; McClosky et al., 2011; Ji and Grishman, 2008; Liao and Grishman, 2010; Li et al., 2013; Yang et al., 2019). As an example, consider the following sentence: *The plane arrived back to base safely.* Here, the word *arrived* is an event trigger that denotes an event of the type "Movement:Transport," while "*The plane*" and "*base*" are its arguments. Given a sentence, the objective of the event detection task is to predict all such event triggers and their respective types.

Recent works on event detection (Nguyen and Grishman, 2018; Liu et al., 2018; Yan et al., 2019; Balali et al., 2020) employ graph based methods (Graph Convolution Networks (Kipf and Welling, 2017)) using the dependency graph (shown in Fig. 1) generated from syntactic dependency-parsers. These methods are able to capture useful non-local dependencies between words that are

---

S. Dutta was a research intern at Dataminr.

Figure 1: Examples of syntactic dependency parsing.

relevant for event detection. However, in most of these works (with the notable exception of Cui et al. (2020)), the graph is treated as a homogeneous graph, and the dependency labels (i.e., edge-types in the graph) are ignored.

Dependency labels can often better inform whether a word is a trigger or not. Consider the two sentences in Fig. 1. In both the sentences, there is an edge between "*police*" and "*fired*". A model that does not take into account dependency labels will only have access to the information that they are connected. However, in the first sentence, "*fired*" is an event trigger of type "Conflict:Attack," whereas in the second sentence, it is of type "Personnel:End Position." The fact that the edge label between "*police*" and "*fired*" is a nominal-subject or an object relation serves as an indicator of the type of event trigger. Hence, leveraging the dependency labels can help improve the event detection performance.

In this work, we propose a simple method to employ the dependency labels into existing models inspired from a recently proposed technique called Graph Transformer Networks (GTN) (Yun et al., 2019). GTNs enable us to learn a soft selection of edge-types and composite relations (e.g., multi-hop connections, called *meta-paths*) among the words, thus producing heterogeneous adjacency matrices.

We integrate GTNs into two homogeneous-graph-based models (that previously ignored the dependency relations), namely, a simple gated-graph-convolution-based model inspired by Nguyen and Grishman (2018); Liu et al. (2018); Balali et al. (2020), and the near-state-of-the-art MOGANED model (Yan et al., 2019), enabling them to now leverage the dependency relations as well. Our method demonstrates a relative improvement in the
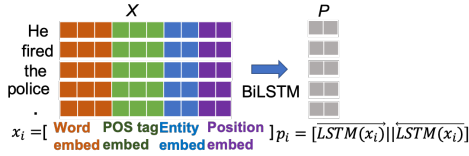
Figure 2: Embedding and BiLSTM Module.
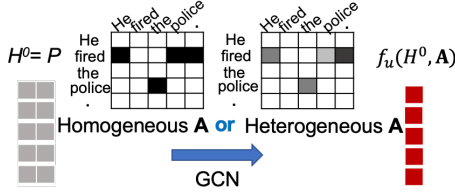
Figure 3: Basic Gated-Graph-Convolution Network.

Figure 4: (Left) Model I; (Right) Model II.

F1 score on the ACE dataset (Walker et al., 2006) for both models, proving the value of leveraging dependency relations for a graph-based model. While the goal of this paper is not to establish a state-of-the-art (SOTA) method, but rather to show the merit of our approach, we do note that the improvements with our method approach the current SOTA (Cui et al., 2020) (which leverages dependency relations using embeddings instead of GTNs).

To summarize, our main contribution is *a method of enabling existing homogeneous-graph-based models to exploit dependency labels for event detection, inspired from GTNs.* Incorporating GTNs in NLP tasks has received less attention (also see recent related work Veyseh et al. (2020)).

Notations: We denote matrices and vectors in bold, e.g., $\boldsymbol{A}$ (matrix) or $\boldsymbol{a}$ (vector). Note that, $A(u,v)$ denotes the element at index (u,v) in matrix $\boldsymbol{A}$.

2 Proposed Method

In this work, we incorporate GTNs onto two homogeneous-graph-based models: (i) **Model I**: a gated-graph-convolution-based model inspired by Nguyen and Grishman (2018); Liu et al. (2018); Balali et al. (2020); and (ii) **Model II**: MOGANED model (Yan et al., 2019). Both models have a similar initial embedding and BiLSTM module, followed by a graph-based module (where their differences lie), and finally a classification module.

Embedding and BiLSTM Module: Our initial module (shown in Fig. 2) is similar to existing works (e.g., Yan et al. (2019)). Each word of the sentence is represented by a token which consists of the word embedding, the POS tag embedding, the Named-Entity type embedding, and its positional
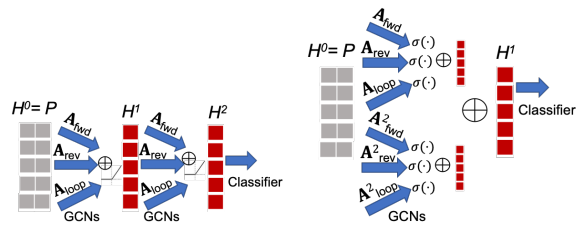
embedding. For a sentence of n words, we denote this sequence of tokens as $X = x_0, x_1, \ldots, x_{n-1}$. Next, we introduce a BiLSTM to encode X into its context $P = p_0, p_1, \ldots, p_{n-1}$ where $p_i = [\overrightarrow{LSTM}(x_i)||\overleftarrow{LSTM}(x_i)]$, and $||$ denotes the concatenation operation. P is then fed to the graph-based module, as discussed next.

Graph-Based Module: We first introduce the basic unit of both Model I and II, i.e., gated-graph-convolution network (see Fig. 3). Let $H^k = h_0^k, h_1^k, \ldots, h_{n-1}^k$ be the input and $H^{k+1} = h_0^{k+1}, h_1^{k+1}, \ldots, h_{n-1}^{k+1}$ be the output of the k-th layer of this module with $H^0 = P$. Given any adjacency matrix $\boldsymbol{A}$ and input H^k, consider the following operation at layer k:

$$f_u(H^k, \boldsymbol{A}) = \sum_{v=0}^{n-1} G_{\boldsymbol{A}}^k(u,v)(\boldsymbol{W}_{\boldsymbol{A}}^k h_v^k + \boldsymbol{b}_{\boldsymbol{A}}^k). \quad (1)$$

Here, $\boldsymbol{W}_{\boldsymbol{A}}^k$ and $\boldsymbol{b}_{\boldsymbol{A}}^k$ are the weight matrix and bias item for the adjacency matrix $\boldsymbol{A}$ at layer k, and $G_{\boldsymbol{A}}^k(u,v)$ is the gated-importance, given by $G_{\boldsymbol{A}}^k(u,v) = A(u,v)\sigma(\boldsymbol{w}_{att,\boldsymbol{A}}^k h_v^k + \epsilon_{att,\boldsymbol{A}}^k)$, where $\sigma(\cdot)$ is an activation function, and $\boldsymbol{w}_{att,\boldsymbol{A}}$ and $\epsilon_{att,\boldsymbol{A}}$ are the attention weight vector and bias item.

A dependency parser, e.g., Stanford Core NLP (Manning et al., 2014), generates a directed heterogeneous graph $\mathcal{G}$ for each sentence (recall Fig. 1). Existing works typically do not use the dependency labels (e.g., nominal-subject); they only derive three homogeneous adjacency matrices from $\mathcal{G}$ as follows: (i) $\boldsymbol{A}_{fwd}$ where $A_{fwd}(i,j) = 1$ if there is an edge from node i to j; (ii) $\boldsymbol{A}_{rev}$ where $A_{rev}(i,j) = 1$ if there is an edge from node j to i; and (iii) $\boldsymbol{A}_{loop}$ which is an identity matrix.

For **Model I** (see Fig. 4 (Left)), the output of the k-th layer (input to $k+1$-th layer) is given by $h_u^{k+1} = \text{ReLu}(\sum_{\boldsymbol{A} \in \{\boldsymbol{A}_{fwd}, \boldsymbol{A}_{rev}, \boldsymbol{A}_{loop}\}} f_u(H^k, \boldsymbol{A}))$. The first layer of gated-graph-convolution network captures dependencies between immediate neighbors (1-hop). To capture K-hop dependencies, Model I has K consecutive layers of such gated-
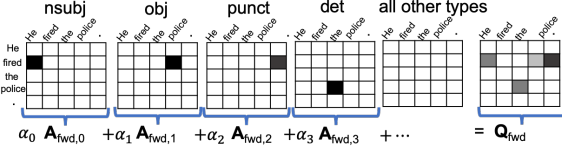
Figure 5: GTN to obtain heterogeneous adjacency matrix of meta-path length 1 (Recall Fig. 1 for the graph).

graph-convolution networks. The output of this graph-based module is then fed to a multi-layer perceptron (MLP) with attention weights for classifying each word into its event-type (or "not an event").

In **Model II**, instead of passing the BiLSTM output P through a *series* of K consecutive gated-graph-convolution layers (to capture K-hop connections), this model separately aggregates the outputs of T *parallel* graph-convolution layers with separate adjacency matrices representing hops of length $1, 2, \ldots, T$ (see Fig. 4 (Right)). Let $H^0 (= P)$ be the input and H^1 be the output of the graph-based module of Model II (which effectively has only one layer, i.e., $k=0$). In Yan et al. (2019), the homogeneous adjacency matrices $\boldsymbol{A}_{fwd}$, $\boldsymbol{A}_{rev}$, and $\boldsymbol{A}_{loop}$ are considered with their corresponding t-hop adjacency matrices $\boldsymbol{A}^t_{fwd}$, $\boldsymbol{A}^t_{rev}$, and $\boldsymbol{A}^t_{loop}$ (multiplied t times) respectively. The output of the graph-based module is given by: $h^1_u = \sum_{t=0}^{T-1} \boldsymbol{w}_{att,t} \boldsymbol{v}_t$ where $\boldsymbol{v}_t = \sum_{\boldsymbol{A} \in \{\boldsymbol{A}_{fwd}, \boldsymbol{A}_{rev}, \boldsymbol{A}_{loop}\}} \sigma(f_u(H^0, \boldsymbol{A}^t))$. Here, $\boldsymbol{w}_{att,t}$ is an attention-weight (further details in (Yan et al., 2019)) and $\sigma(\cdot)$ is the exponential linear unit[1]. Finally, these outputs are passed through an MLP with attention weights for classification.

Remark. The reason for using only three matrices instead of a separate adjacency matrix for each edge-type is that it results in an explosion of parameters for the gated-graph-convolution network, as individual weight matrices have to be learnt for each type of edge (see also Nguyen and Grishman (2018)). In this work, we replace the homogeneous matrices $\boldsymbol{A}_{fwd}$ and $\boldsymbol{A}_{rev}$ with heterogeneous adjacency matrices without a significant overhead in the number of parameters, as discussed next.

Obtaining Heterogeneous Adjacency Matrices With GTN: Consider a directed heterogeneous graph $\mathcal{G}$ with each edge belonging to one of L types. This graph can be represented using a set of L adjacency matrices $\{\boldsymbol{A}_{fwd,0}, \boldsymbol{A}_{fwd,1}, \ldots, \boldsymbol{A}_{fwd,L-1}\}$,

each corresponding to a different edge-type (dependency label). $A_{fwd,l}(i, j) = 1$ if there is a directed edge from node i to j of type l. A GTN obtains a heterogeneous adjacency matrix by learning a convex combination $\boldsymbol{Q}_{fwd} = \sum_{l=0}^{L-1} \alpha_l \boldsymbol{A}_{fwd,l}$ (see Fig. 5) where $\boldsymbol{\alpha} = \text{softmax}(\boldsymbol{w})$ and $\boldsymbol{w}$ is a weight vector that the model learns. The matrix $\boldsymbol{Q}_{fwd}$ is a heterogeneous adjacency matrix with an "appropriately weighted" edge between any two nodes that have an edge in any of the L original matrices.

For **Model I**, we first generate a set of L adjacency matrices (for L edge-types) corresponding to the directed forward edges, and another set of L adjacency matrices corresponding to the reverse edges. Next, we learn heterogeneous adjacency matrices, i.e., $\boldsymbol{A}_{fwd} = \boldsymbol{Q}_{fwd}$ and $\boldsymbol{A}_{rev} = \boldsymbol{Q}_{rev}$. Our technique enables baseline Model I to leverage dependency relations by learning only $2L$ more scalar parameters which is significantly less than learning individual weight matrices for L edge-types.

For **Model II**, we not only aim to learn heterogeneous adjacency matrices to replace the homogeneous $\boldsymbol{A}_{fwd}$ and $\boldsymbol{A}_{rev}$, but also learn heterogeneous adjacency matrices that have an "appropriately weighted" edge between every two nodes that are t-hops apart in the original graph $\mathcal{G}$ (called a meta-path of length t) so as to replace $\boldsymbol{A}^t_{fwd}$ and $\boldsymbol{A}^t_{rev}$. Specifically, for the case of $t = 2$, GTN first learns two convex combinations $\boldsymbol{Q}_{fwd,0}$ and $\boldsymbol{Q}_{fwd,1}$ (each corresponds to meta-paths of length 1), and then computes the product $\boldsymbol{Q}_{fwd,0}\boldsymbol{Q}_{fwd,1}$. Similarly, one can compute a product of t such adjacency matrices to learn meta-paths of length t.

We replace all t-hop adjacency matrices with heterogeneous adjacency matrices of meta-path length t, learnt through GTNs, e.g., $\boldsymbol{A}^t_{fwd}$ is replaced by $\boldsymbol{Q}_{fwd,0}\boldsymbol{Q}_{fwd,1} \ldots \boldsymbol{Q}_{fwd,t-1}$, where each $\boldsymbol{Q}_{fwd,i}$ is a convex combination of L-adjacency matrices corresponding to the directed forward edges. Similar heterogeneous adjacency matrices of meta-path length t are learnt for the reverse edges as well to replace $\boldsymbol{A}^t_{rev}$. This modification enables the baseline Model II to leverage the dependency relations, by only learning $2Lt$ more scalar parameters for each t, which is practicable.

3   Results

Dataset and Evaluation Metrics: We use the benchmark ACE2005 English dataset (Walker et al., 2006) with the same data split as in prior works (where the sentences from 529, 30, and 40 docu-

---

[1]The gated-importance $G^k_{\boldsymbol{A}}(u, v)$ has subtle differences between Model I and II.

K	P	R	F1	K	P	R	F1
1	72.0	75.5	73.7	1	72.9	**76.4**	**74.6**
2	70.7	75.7	73.1	2	72.1	75.9	74.0
3	72.8	70.7	71.7	3	**73.8**	73.6	73.7

Table 1: Performance of gated-graph-convolution-based models (**Model I**) for varying number of consecutive convolution layers (K): (Left) Baseline models with no GTNs; (Right) Proposed models with GTNs.

Method	P	R	F1
Baseline (no GTNs)	79.5	72.3	75.7
Proposed (with GTNs)	**80.9**	**73.2**	**76.8**

Table 2: Performance of MOGANED (**Model II**).

ments are used as the training, validation, and test set). We use the Stanford CoreNLP toolkit (Manning et al., 2014) for sentence splitting, tokenizing, POS-tagging and dependency parsing. We use word embeddings trained over the New York Times corpus with Skip-gram algorithm following existing works (Yan et al., 2019). We evaluate the Precision (P), Recall (R) and F1 score.

Model Settings: For Model I, the number of consecutive layers of gated-graph-convolution networks (K) is varied from 1 to 3. For Model II, we use the code[2] with same hyper parameter settings.

Performance: For both Models I and II, GTNs demonstrate an improvement of about 1 point F1 score (see Tables 1 and 2). The 76.8 F1 score for Model II with GTNs is also quite close to the SOTA performance of 77.6 for this task (Cui et al., 2020).

Examining Specific Predictions For Insights: To explain the role of GTNs, we examined all the predictions on the validation set using the baseline Model II (no GTNs) and the proposed Model II (with GTNs). We include some specific instances here that we found interesting and insightful.

We observe that using GTNs makes the predictions more "precise," by reducing the number of false-positive event trigger detections. For instance, *He's now national director of Win Without War, and* ***former*** *Congressman Bob Dornan, Republican of California.* Here, "*former*" is the only event trigger (type Personnel:End-Position), as is correctly identified by our model. However, the baseline model also falsely identifies "*War*," as an event trigger of type Conflict:Attack. Another example is: *In a monstrous conflict of interest, [...].* Here, the baseline falsely identifies "*conflict*," as a trigger of

type Conflict:Attack. Our model is able to identify "*War*," and "*conflict*" as non-triggers based on their context in the sentence, while the baseline seems to be over-emphasizing on their literal meaning.

In some cases, the baseline model also leads to misclassification. For instance, *The Apache troop* ***opened*** *its tank guns,[...].* Here, "*opened*," is an event trigger of type Conflict:Attack, as is correctly identified by our model; however, the baseline misclassifies it as type Movement:Transport.

Another interesting example is: *[...] Beatriz* ***walked*** *into the USCF Offices in New Windsor and immediately* ***fired*** *17 staff members.* Here, "*walked*" is an event trigger of type Movement:Transport, and "*fired*" is of type Personnel:End-Position. The baseline model misclassifies "*fired*" as Conflict:Attack, while using GTNs help classify it correctly. However, using GTNs can sometimes miss certain event triggers while attempting to be more precise, e.g., "*walked*" is missed when using GTNs while the baseline model identifies it correctly.

Lastly, there are examples where both the baseline and proposed models make the same errors. E.g., *I* ***visited*** *all their families.* or, *I would have* ***shot*** *the insurgent too.* Here, both models misclassify "*visited*," (type Contact:Meet) as Movement:Transport, and "*shot*," (type Life:Die) as Conflict:Attack. As future work, we are examining alternate techniques that better inform the context of the event trigger in such sentences. Another interesting example is: "*It is legal, and* ***it*** *is done.*" Both models miss "*it*," (type Transaction:Transfer-Money). For this example (and some other similar examples of anaphora resolution), we believe that it might be quite non-intuitive to classify the event trigger from the sentence alone, and dependencies among sentences from the same article might need to be leveraged to better inform the context, as we will examine in future work.

4 Conclusion

We developed a novel method of enabling existing event extraction models to leverage dependency relations without a significant rise in the number of parameters to be learnt. Our method relies on GTN, and demonstrates an improvement in F1 score over two strong baseline models that do not leverage dependency relations. The benefits of using GTN in an NLP task suggests that other NLP tasks could be improved in the future.

---

[2] https://github.com/ll0iecas/MOGANED

References

Ali Balali, Masoud Asadpour, Ricardo Campos, and Adam Jatowt. 2020. Joint event extraction along shortest dependency paths using graph convolutional networks. *arXiv preprint arXiv:2003.08615*.

Shiyao Cui, Bowen Yu, Tingwen Liu, Zhenyu Zhang, Xuebin Wang, and Jinqiao Shi. 2020. Event detection with relation-aware graph convolutional neural networks. *arXiv preprint arXiv:2002.10757*.

Xinya Du and Claire Cardie. 2020. Event extraction by answering (almost) natural questions. *arXiv preprint arXiv:2004.13625*.

Xiaocheng Feng, Lifu Huang, Duyu Tang, Heng Ji, Bing Qin, and Ting Liu. 2016. A language-independent neural network for event detection. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 66–71.

Heng Ji and Ralph Grishman. 2008. Refining event extraction through cross-document inference. In *Proceedings of ACL-08: Hlt*, pages 254–262.

Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR*.

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)*, pages 73–82.

Shasha Liao and Ralph Grishman. 2010. Using document level cross-event inference to improve event extraction. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 789–797.

Xiao Liu, Zhunchen Luo, and Heyan Huang. 2018. Jointly multiple events extraction via attention-based graph information aggregation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1247–1256, Brussels, Belgium. Association for Computational Linguistics.

Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David Mc-Closky. 2014. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60.

David McClosky, Mihai Surdeanu, and Christopher D Manning. 2011. Event extraction as dependency parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 1626–1635.

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*, pages 300–309.

Thien Huu Nguyen and Ralph Grishman. 2018. Graph convolutional networks with argument-aware pooling for event detection. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, pages 5900–5907.

Amir Pouran Ben Veyseh, Tuan Ngo Nguyen, and Thien Huu Nguyen. 2020. Graph transformer networks with syntactic and semantic structures for event argument extraction. *arXiv preprint arXiv:2010.13391*.

Christopher Walker, Stephanie Strassel, Julie Medero, and Kazuaki Maeda. 2006. Ace 2005 multilingual training corpus. *Linguistic Data Consortium, Philadelphia*, 57:45.

Haoran Yan, Xiaolong Jin, Xiangbin Meng, Jiafeng Guo, and Xueqi Cheng. 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 (EMNLP-IJCNLP)*, pages 5770–5774.

Bishan Yang and Tom Mitchell. 2016. Joint extraction of events and entities within a document context. *arXiv preprint arXiv:1609.03632*.

Sen Yang, Dawei Feng, Linbo Qiao, Zhigang Kan, and Dongsheng Li. 2019. Exploring pre-trained language models for event extraction and generation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5284–5294.

Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. 2019. Graph transformer networks. In *Advances in Neural Information Processing Systems*, pages 11983–11993.

Yunyan Zhang, Guangluan Xu, Yang Wang, Daoyu Lin, Feng Li, Chenglong Wu, Jingyuan Zhang, and Tinglei Huang. 2020. A question answering-based framework for one-step event argument extraction. *IEEE Access*, 8:65420–65431.

A   More Details on the MOGANED model

There are some subtle differences in the graph-attention mechanisms of Model I and II. In particular, for Model II, the gated-importance $G_A^0(u, v)$ in equation (1) is redefined as follows: $G_A^0(u, v)=\text{softmax}(E(u, v))$, where $E(u, v)=A(u, v)\gamma(\boldsymbol{W}_{c,\boldsymbol{A}}[\boldsymbol{W}_{att,\boldsymbol{A}}h_u^0||\boldsymbol{W}_{att,\boldsymbol{A}}h_v^0])$, γ is LeakyReLU (with negative input slope α), and $\boldsymbol{W}_{c,\boldsymbol{A}}$ and $\boldsymbol{W}_{att,\boldsymbol{A}}$ are weight matrices. Further details are provided in Yan et al. (2019).

B   Data Preprocessing

We use the same data split as several existing works (Nguyen et al., 2016; Nguyen and Grishman, 2018; Liu et al., 2018; Balali et al., 2020; Yan et al., 2019; Cui et al., 2020; Ji and Grishman, 2008; Liao and Grishman, 2010; Li et al., 2013), where the sentences from 529, 30, and 40 documents are used as the training, validation, and test set. For preprocessing, we directly used the following code[3] which uses the Stanford Core NLP toolkit (Manning et al., 2014).

C   Hyper Parameter Setting

For both the models, we select 100 as the dimension of the word embeddings, and 50 as the dimension of all the other embeddings, i.e., POS-tag embedding, Named-Entity-type embedding, and positional embedding. Following prior work, we restrict the length of each sentence to be 50 (truncating long sentences if necessary). We select the hidden units of the BiLSTM network as 100. We choose a batch size of 10, and Adam with initial learning rate of 0.0002. We select the dimension of the graph representation to be 150. When using GTNs, the number of edge-types (L) is 35, which is determined by the number of unique types of dependency relations, e.g., nsubj, case, etc., as obtained from the dependency parser.

---

[3] https://github.com/nlpcl-lab/ace2005-preprocessing