creative commons
COMMONS DEED

저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:

저작자표시. 귀하는 원저작자를 표시하여야 합니다.

비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.

변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

dCollection

문학석사 학위논문

Event Argument Extraction with Trigger-Relevant Message Passing on Dependency Graphs

의존 구문 그래프에서의 사건 술어 관련 메시지 전달을 통한 사건 논항 추출

2021 년 8 월

서울대학교 대학원

언어학과

손 보 경

Event Argument Extraction with Trigger-Relevant Message Passing on Dependency Graphs

지도교수 신 효 필

이 논문을 문학석사 학위논문으로 제출함

2021 년 6 월

서울대학교 대학원

언어학과

손 보 경

손보경의 문학석사 학위논문을 인준함

2021 년 8 월

위 원 장 남 승 호

부위원장 신 효 필

위 원 김 문 형

Abstract

Event Argument Extraction with Trigger-Relevant Message Passing on Dependency Graphs

Bokyung Son

Department of Linguistics

The Graduate School

Seoul National University

Event Argument Extraction (EAE) is a task of extracting event arguments and identifying their roles in predefined events present in unstructured natural language texts. In this paper, I improve on previous works that exploit dependency trees as a source of syntactic information for EAE. While previous works discarded dependency label information and treated dependency trees as homogeneous graphs, I introduce dependency labels to the model as typed edges. I present an interpretable model that captures interactions between words in a text with attention flow on dependency graphs. Noting that the interactions are determined by the event type and the trigger word semantics, the word representations are generated in a trigger-relevant fashion. At the core of the model lies Trigger-Relevant Message Passing (`TRMP`), a graph neural network that defines path traversal on the dependency graph as attention propagation

and intermediates node representation updates with the propagated attention. The presented model shows state-of-the-art performance for the role classification subtask, demonstrating the benefits of making full use of dependency labels.

Keywords: Event Argument Extraction, Event Extraction, Dependency Parsing, Attention Flow, Message Passing, Graph Neural Network, Deep Learning

Student Number: 2018-23942

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Event Extraction

Extracting structured information from unstructured texts is a crucial task that serves for various natural language understanding applications such as information retrieval, summarization, knowledge base construction, timeline extraction, and online monitoring systems. Event Extraction (EE), a closed information extraction task, particularly refers to extracting event information according to predefined event schemas.

The most influential program for EE is Automatic Context Extraction (ACE) [8]. It defines a set of terms for the EE task.

Event An event is a specific occurrence involving participants, frequently described as change of status. Both actions and their resultative states can be events. ACE has 8 event types, further classified into 33 subtypes.

Event trigger The word that most clearly expresses an event. Triggers can appear in the following forms:

- Main verb: "The explosion *killed* 7 and *injured* 20."

- Main verb in the form of an adjective/past-participle: "17 sailors were *killed*."
- Modifier adjective/participle: "He said he had no information about any *dead* or *injured* members of the submarine crew."
- Noun/pronoun: "The *attack* killed 7 and injured 20."

Event argument A participant or an attribute with a specific role in an event, indicating time (when), location (where), source/actor (who), target(whom), or instruments (how). ACE tags 35 argument roles.

Entity mention A superset of event arguments. These include named entities and nominals that denote objects (*e.g.* "White House spokesman", "the terrorists"), temporal expressions (*e.g.* "today") and values (*e.g.* "40 million dollars").

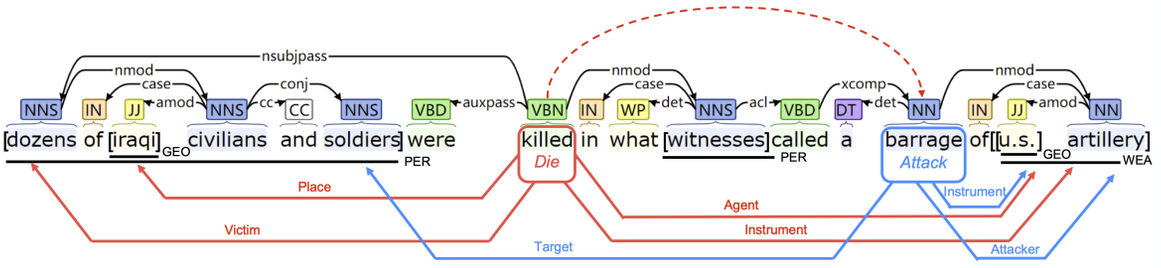

Figure 1.1: An example of event extraction results with dependency parse, POS tags and entity information. Figure from [28].

[Figure 1.1] shows a sentence that contains two events: a `Die` event triggered by the word "killed" with 4 arguments in red, and an `Attack` event triggered

by "barrage" with 3 arguments in blue. Entity mentions are underlined with their entity types in subscripts.

Event extraction consists of 2 tasks, Event Detection (ED) and Event Argument Extraction (EAE). Each task has 2 subtasks. Note that EAE is not a superset of ED as it does not involve predicting the trigger offsets.

Trigger identification An event trigger is correctly identified if its offsets match any of the reference.

Type classification An event trigger is correctly classified if it is correctly identified and its event subtype matches the reference.

Event detection (ED) Trigger identification + Type classification.

Argument identification An event argument is correctly identified if its event subtype matches the reference and its offsets match any of the reference.

Role classification An event argument is correctly classified if it is correctly identified and its role matches the reference.

Event argument extraction (EAE) Argument identification + Role classification.

1.2 Event Semantics and Syntax

The ACE schema views events as involving various arguments that play specific semantic roles. It also assumes that event is encoded by a trigger word. Such

concept of event as well as the very nature of the EE task– extracting items that defines an event from the surface sentence– could be understood in the context that argument structure is directly determined by the lexical properties of predicates. This is an idea widely present in various literature [25]. Fillmore's Case Grammar [10] states that a verb is defined by the semantic roles that it takes, *i.e.* case frame, such as agent, patient, instrument, and goal. Perlmutter and Poster [41] also argue the following:

Universal Alignment Hypothesis:

There exist principles of UG which predict the initial relation borne by each nominal in a given clause from the meaning of the clause.

Event structure is an approach to define semantic roles in terms of more basic elements of lexical semantic representation. A distinction between the two aspects of verb meaning have been drawn: the grammatically relevant part, and the other part specific to the verb. The former, "semantic structure" in Grimshaw's term [13] or the "structural" aspect in Hovav and Levin's term [16], is called the *event structure*. Semantic roles are considered labels for argument positions in an event structure. The latter, "semantic content" (Grimshaw) or "idiosyncratic" (Hovav and Levin), is called the constant. Hovav and Levin proposed a set of event templates and canonical realization rules that associate constants with the event templates, such as:

1. instrument $\rightarrow$ [x ACT$_{\text{<INSTRUMENT>}}$]

   (*e.g.* brush, hammer, saw, shovel)

2. place → [x CAUSE [BECOME [y <PLACE>]]]

   (*e.g.* bag, box, cage, crate, garage, pocket)

Rule 1 pairs a constant of `instrument` type (left-hand side) with the `activity` event template (right-hand side). Rule 2 pairs a constant of `place` type with the `accomplishment` event template. Then, each structure participant in the event structure is realized in their surface syntactic positions through linking rules. Thematic hierarchy theories align semantic roles to a prominence hierarchy that determines the realization of syntactic arguments. Note that the exact realization of a semantic role depends on concurring semantic roles. Following is a classic example from Filmore [10] showing that an instrument cannot be realized as subject in the presence of an agent.

> If there is an A [=Agent], it becomes the subject;
> otherwise, if there is an I [=Instrument], it becomes the subject;
> otherwise, the subject is the O [=Objective, *i.e.* Theme/Patient].

- Thematic hierarchy: Agent > Instrument > Theme/Patient
- Subject selection rule: The argument of a verb bearing the highest-ranked semantic role is its subject.
- Resulting grammaticality patterns:
  - The chisel opened the door.
  - Dana opened the door with a chisel.
  - *The chisel opened the door by Dana.

1.3 Purpose of Study

This paper is on Event Argument Extraction (EAE). EAE involves detecting events from a sentence, identifying the event types, and identifying the arguments as well as their semantic roles in the events. Considering the theories on the derivative status of argument realization from event semantics, it is natural to consider using syntactic information for EAE as a reverse engineering process. For example, in [Figure 1.1], the fact that "Iraqi" is a `Place` argument for the `Die` event while "U.S." cannot be known solely from knowing the ontological categories of the words because both "Iraqi" and "U.S." are location-related entities. Knowing that the trigger predicate "killed" has a passive nominal subject and "Iraqi" is an adjectival modifier of the nominal could help reaching the conclusion together with the ontological information that "Iraqi" is a geographical entity.

Indeed, dependency trees have been used as a source of grammatical information in many previous works [36, 45, 39, 28, 21, 6, 30, 53]. However, in most cases, they are not used in their *full* form; the label information is discarded and only the connection information is used. Differently put, the information whether two words form a head–dependent relation is used, but the nature of that relation is ignored. This is likely due to the burden of overparameterization when modelling a large number of dependency relations compared to the dataset size. Cui et al. [6] have empirically shown the benefits of using the dependency labels as edge types in dependency graphs for ED. Gains from using the linguistic cues present in dependency labels could be even greater in EAE,

where more diverse interdependencies between words need to be modelled than in identifying just the trigger word(s). This paper is the first work to use dependency labels for EAE. Through ablation studies, I empirically demonstrate that efficient modelling of the labels with basis decomposition improves performance.

In addition, this paper aims to provide an *interpretable* model for the reverse engineering process. I introduce the Trigger-Relevant Message Passing (`TRMP`) module, which is a graph neural network that models trigger-aware information flow on dependency graphs. The information flow is modelled with transitions in attention distribution. By analyzing the attention trajectory across words at each `TRMP` iteration, one can obtain the information on which words and dependency relations were considered important by the model in which order. Such information could serve as empirical data for bottom-up construction of theories on argument realization.

Finally, the model presented in this paper incorporates major improvements made in previous approaches: using a pretrained deep text encoder for better contextualization, and handling the role overlap problem [65]. It is trained in an end-to-end fashion and computes all relevant outputs at one sweep through the pipeline. The proposed model achieves the highest performance for the role classification subtask.

Chapter 2

Background

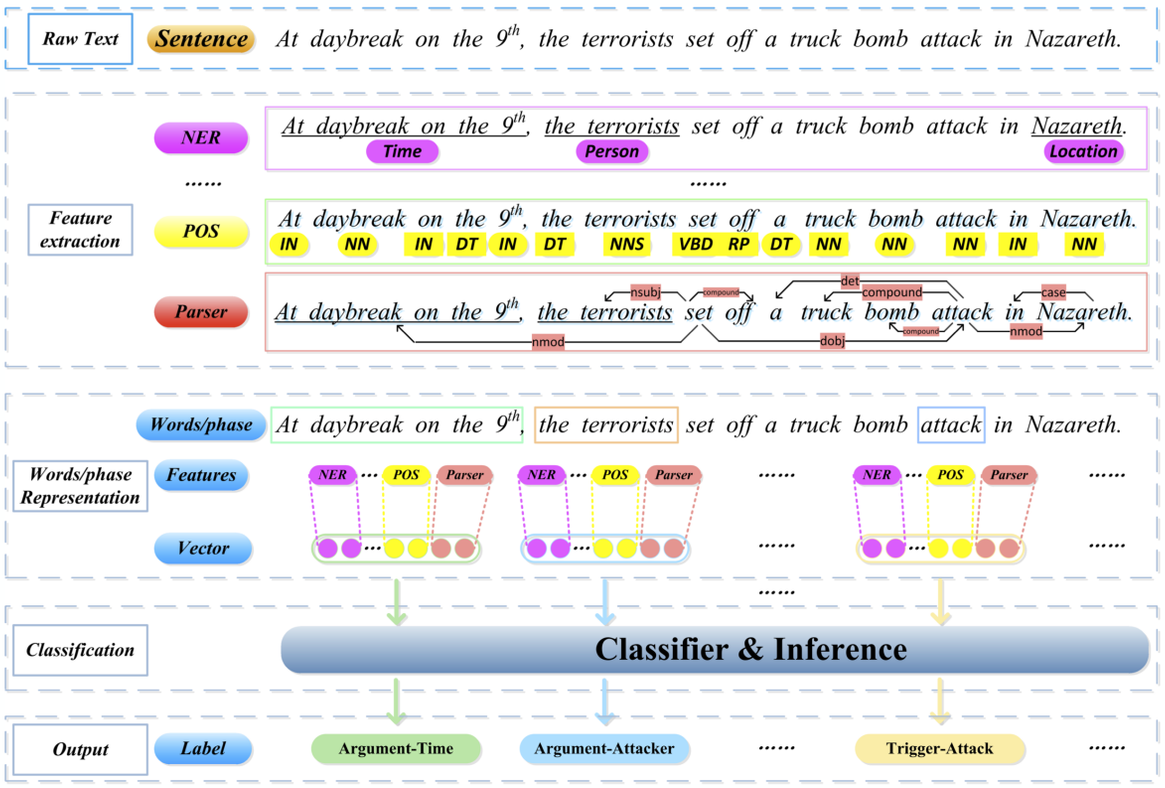

Figure 2.1: **General pipeline for event extraction.** Figure from [63].

[Figure 2.1] shows the general pipeline for event extraction based on deep learning. A number of features (*e.g.* lemmas, part-of-speech tags, entity types, syntactic parses, sequential positions) are extracted from the raw text through

feature engineering. Then, the words are embedded in a dense form with their extracted features. The encoded representations are fed into classifiers (fully-connected networks) to predict offsets and/or types of event triggers and/or arguments.

In Section 2.1, I introduce various encoding architectures proposed in the EE literature in the context of making better *contextual* embeddings that capture semantics above lexical-level by modelling interdependencies between words. In Section 2.2, I focus on the use of dependency parses as features and introduce previous works that applied GNNs on the dependency graphs.

2.1 Modelling Contextual Semantics for Event Extraction

It is true that memorizing word-to-event type mapping (*e.g.* destroy $\rightarrow$ `Attack`) often works for EE. Nevertheless, it is never sufficient as there are ambiguous cases that require contextual semantics in order to be resolved. See the following example:

He has *fired*$^{[\text{Trigger}]}$ his air defense chief.

What event type should we assign to the trigger word "fired"? Considering only its dictionary meaning, "fired" could be either of type `Attack` or `End-Position`. In order to select the latter, which is the reference event type, the EE model would need additional information from context that "air defense chief" is a job title.

Convolutional Neural Networks (CNN) [20] were the first architecture used for EE based on deep learning. CNN-based models run convolution operations on lexical features in order to capture contextual relations of a word to its neighboring words [37, 5]. Further improvements came from modelling long-range and non-consecutive dependencies on top of sequentially local dependencies captured by simple convolutions. Nguyen and Grishman [38] proposed to perform convolutions on all possible consecutive and non-consecutive k-grams in a sentence, and distinguish the most important k-gram with the convolution scores from the max-pooling function. Other works moved on to Recurrent Neural Networks (RNN) [3] which can model sequential dependencies between any two words in a sentence [11, 4, 36].

Dependency trees has been recognized as a valuable source of information for syntactic interdependencies between words. Some work used dependency relations at embedding-level as binary vectors [36, 40, 22]. RNNs with specialized dependency-based structures were also introduced. Sha et al. [45] used a Dependency-Bridged RNN where two RNN cells with a dependency relation have an additional connection so that each cell can directly receive information from syntactically related cells, apart from the last cell. Zhang et al. [68] used a Tree-LSTM [47] on transformed dependency trees where leaf nodes are words and internal nodes are dependency relations.

With the reign of Transformers [51], applying self-attention has been the dominant trend in EE. Self-attention is an attention mechanism relating different positions of a single sequence. Three values are computed from different

linear projections of the input sequence: *query, key*, and value. The task is to find the important words from the keys for the query word. This is done by passing the query and key to an affinity function (*e.g.* dot-product similarity) followed by the softmax normalization. With the softmax output as weight, the resulting context vector for the query is computed as a weighted sum of values. Following is the equation for scaled dot-product self-attention as in Vaswani et al. [51].

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^{\mathrm{T}}}{\sqrt{d_h}})\mathbf{V} \tag{2.1}$$

Transformers use multi-head attention in order to jointly attend to information from different representation subspaces at different positions.

Self-attention can be adapted to particularly handle dependency structures by altering the query, key and value matrices accordingly. Ma et al. [30] use a Syntax-Attending Transformer where one head only attends to neighbors in the dependency graph [1].

Since the seminal work BERT by Devlin et al. [7], Transformer-based pre-trained language models have been the mainstream in NLP. BERT is a stack of Transformer encoders pretrained on huge corpora (16GB BookCorpus and English Wikipedia) with the Masked Language Modelling objective (MLM) [2]. MLM is a denoising autoencoding technique; for a word sequence, MLM asks the model to predict not the next word but rather random words within the

[1] In Section 2.2, I introduce GATs [52] that performs masked self-attention in the graph domain.

[2] The next sentence prediction, a binary classification task predicting whether two input sequences are consecutive, was also used but have been considered only secondary or ineffective [66, 29].

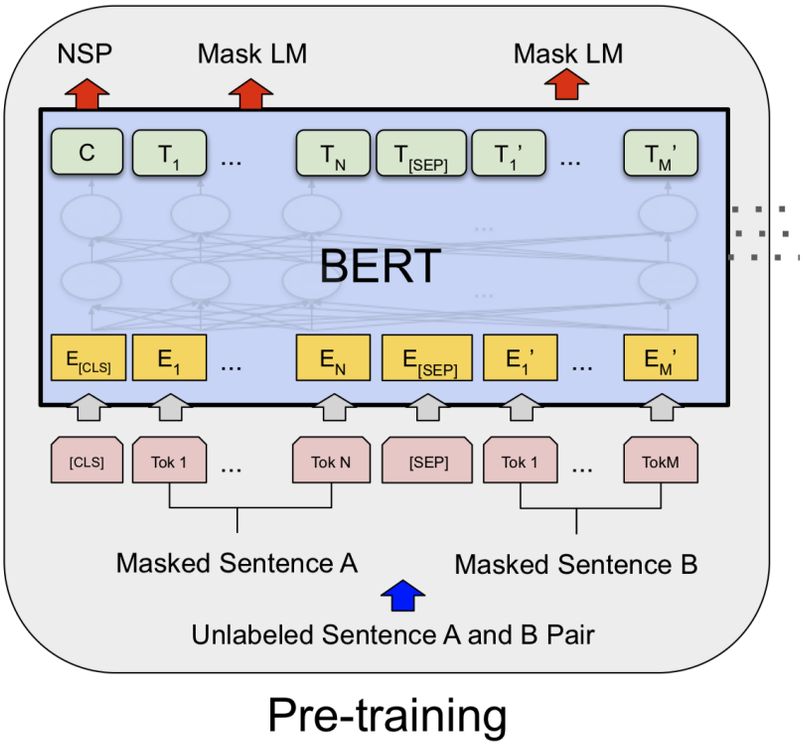

Figure 2.2: **BERT architecture for pretraining.** Figure from [7].

sequence after masking those out. By inferring the masked words, the model is asked to exploit information from other words, learning distributional contextual representations on top of the initial lexical embeddings (WordPiece [62] for BERT). Word representations from different layers encode different linguistic information [49, 48].

Some recent works solely rely on pretrained language models instead of using features contextualized on dependency graphs. Some use contextualized span extractions from BERT [65, 54]; others solve EE as a downstream task for BERT such as machine reading comprehension [27] or question-answering [9].

In the next section, I introduce works that apply GNNs on dependency

graphs. These works share a common design: word representations from pre-GNN sentence encoders such as RNNs [39, 28, 6] or BERT [30, 21, 53] are used as initial node embeddings instead of lexical embeddings. This gives rise to a question: If GNNs are themselves capable of modelling the node interdependencies, why contextualize the nodes before GNN modules at the first place? In GNNs, nodes obtain feature information from all nodes that are k hops away in the graph after k message passing iterations. As each GNN iteration typically corresponds to one layer pass, this implies that k layers are required to pass information between k-hop neighboring nodes. This effect is similar to CNNs, where depth increases the receptive field of internal features. However, having a large number of layers is undesirable due to overparameterization issues, especially for a just moderate-sized dataset like ACE 2005. Therefore, employing a pre-GNN sentence encoder serves as a strategy to model dependencies between words that may be far in the dependency graph while maintaining the GNN shallow.

2.2 Applying GNNs on Dependency Graphs

A graph is a pair $G = (V, E)$ where $V = \{v_1, \cdots, v_{|V|}\}$ is a set of nodes and $E = e_1, \cdots, e_{|E|}$ is a set of edges. $e_{ij} \in E$ is a pair (v_i, v_j) with $v_i, v_j \in V$ (called the ends of the edge). Nodes may have d-dimentional features $X \in \mathbb{R}^{|V| \times d}$, and same for the edges (although less common). A finite graph can be represented as a square adjacency matrix $\mathbf{A} \in \mathbb{R}^{|V| \times |V|}$, where the matrix elements indicate whether pairs of nodes have edges between them or not. A graph is directed

when the existence of $e_{ij} \in E$ does not necessarily imply $e_{ji} \in E$, *i.e.* the relationships are asymmetric. A graph is simple if it has no loops (edge where two ends are identical to each other) or parallel edges (two edges that have the same ends).

A dependency tree of a sentence is a directed simple graph with the following characteristics and constraints:

- A node corresponds to an existing word in the sentence.
- An edge denotes a dependency relation from a *head* to its *dependent*.
- There is a single designated root node with no incoming arcs.
- Except for the root node, each node has only one incoming arc.
- There is a unique path from the root node to each node in $\mathcal{V}$.

Since each word corresponds to a node in the dependency graph, EAE tasks are viewed as classification problems on node representations (when gold entity information is not given) or entity representations pooled from component node representations (when gold entity information is given) [3].

The node representations for the classifiers are computed with Graph Neural Networks (GNN), which are neural networks that operate on the graph domain. GNNs have been a popular choice for EE in order to exploit the topological information of dependency graphs. A typical GNN consists of multiple

[3] See Chapter 3 for details on gold entity information.

blocks where each block is a graph-to-graph module that takes a graph as input, performs computations over the structure, and returns a graph as output. [Algorithm 1] shows the computational steps within a generic GNN block [2]. It involves aggregation functions ρ and update functions ϕ. ρ should take variable numbers of arguments and must be invariant to permutations of their inputs, such as elementwise summation, mean, maximum, and minimum. ϕ are implemented with various neural networks.

Algorithm 1 `Computational steps within a GNN block.`

Require: $V = \{\mathbf{v}_i\}_{i=1:|V|}$ where $\mathbf{v}_i$ is a node feature
Require: $E = \{(\mathbf{e}_k, s_k, r_k)\}_{k:1:|E|}$ where $\mathbf{e}_k$ is an edge feature, s_k is the sender node index, r_k is the receiver node index
Require: $\mathbf{u}$, a global feature

function GNNBlock$(E, V, \mathbf{u})$
  for $k \in \{1, \cdots, |E|\}$ **do**
    $\mathbf{e}'_k \leftarrow \phi^e(\mathbf{e}_k, \mathbf{v}_{s_k}, \mathbf{v}_{r_k}, \mathbf{u})$ ▷ Compute updated edge features
  end for
  for $i \in \{1, \cdots, |V|\}$ **do**
    $E'_i \leftarrow \{(\mathbf{e}'_k, s_k, r_k)\}_{r_k=i, k=1:|E|}$
    $\bar{\mathbf{e}}'_i \leftarrow \rho^{e\rightarrow v}(E'_i)$ ▷ Aggregate edge features per node
    $\mathbf{v}'_i \leftarrow \phi^v(\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u})$ ▷ Compute updated node features
  end for
  $V' \leftarrow \{\mathbf{v}'_i\}_{i=1:|V|}$
  $E' \leftarrow \{(\mathbf{e}_k, s_k, r_k)\}_{k=1:|E|}$
  $\bar{\mathbf{e}}' \leftarrow \rho^{e\rightarrow u(E')}$ ▷ Aggregate edge features globally
  $\bar{\mathbf{v}}' \leftarrow \rho^{v\rightarrow u(V')}$ ▷ Aggregate node features globally
  $\mathbf{u}' \leftarrow \phi^u(\bar{\mathbf{e}}', \bar{\mathbf{v}}', \mathbf{u})$ ▷ Compute updated global feature
  return $(E', V', \mathbf{u}')$
end function

In this paper, I use the term *message-passing*. A Message-Passing Neural Network (MPNN) [12] is a GNN with the following configurations at each block

k:

- The message function M_k as ϕ^e but not taking $\mathbf{u}$ as input,
- Elementwise summation for $\rho^{e \to v}$,
- The update function U_k as ϕ^v,
- The readout function R as ϕ^u but not taking $\mathbf{u}$ or E' as input.

Specifically, in simpler notations,

$$m^{(k)} = \sum_{i \in NI(j)} M_k(\mathbf{h}_i^{(k-1)}, \mathbf{h}_j^{(k-1)}, \mathbf{e}_{ij}) \tag{2.2}$$

$$\mathbf{h}^{(k)} = U_k(\mathbf{h}_j^{(k-1)}, m^{(k)}) \tag{2.3}$$

$$\hat{y} = R(\{\mathbf{h}^{(K)} \mid v \in V\}). \tag{2.4}$$

Each node v_j receives *message* [Equation 2.2] propagated from its incoming nodes in $NI(j)$, *i.e.* the set of neighboring nodes along incoming edges. The message is aggregated over the neighboring node set. The aggregated message and (optionally) the node representation computed at the previous iteration are used for updating the node representation [Equation 2.3]. After K iterations of message passing, a feature vector of the whole graph is computed based on the final node representations [Equation 2.4].

Spatial Graph Convolutional Networks (GCN), a popular choice for EE [39, 28, 21], follow the MPNN framework. They define convolutions directly on the graph based on the graph topology. Convolutional filters in spatial GCNs may or may not be specific to edge types. Here, I explain with a more general

typed version as used in Nguyen et al. [39] and Liu et al. [28]. The convolution vector at the k-th layer for a node v_j is computed as:

$$\mathbf{h}_j^{(k)} = \sigma\Big(\sum_{i \in NI(j)} \mathbf{W}_e^{(k)} \mathbf{h}_i^{(k-1)} + \mathbf{b}_e^{(k)} \Big). \tag{2.5}$$

The message function (term inside the summation) is computed as a convolution of input features $\mathbf{H}$ with trainable filters $\mathbf{W}_e^{(k)} \in \mathbb{R}^{d_k \times d_{k-1}}$, $\mathbf{b}_e^{(k)} \in \mathbb{R}^{d_k}$ specific to edge type e. After being summed, it goes through a nonlinearity readout function σ such as ReLU.

As not all edges are equally informative, the message could be weighted according to its computed importance according to edge type as in Weighted GCN [46] [39, 28]. With $\mathbf{V}_e^{(k)} \in \mathbb{R}^{d_k \times d_{k-1}}$ and $\mathbf{d}_e^{(k)} \in \mathbb{R}^{d_k}$,

$$\alpha_e^{(k)} = \sigma(\mathbf{h}_i^{(k-1)} \mathbf{V}_e^{(k)} + \mathbf{d}_e^{(k)}) \tag{2.6}$$

$$\mathbf{h}_j^{(k)} = \sigma\Big(\sum_{i \in NI(j)} \alpha_e^{(k)} (\mathbf{W}_e^{(k)} \mathbf{h}_i^{(k-1)} + \mathbf{b}_e^{(k)}) \Big). \tag{2.7}$$

Liu et al. [28], the state-of-the-art work for the argument identification subtask, used a weighted GCN operating on biLSTM [15] outputs. They computed the context vector for each word w_i through self-attention on the final node representations $\mathbf{h}$:

$$\alpha_i = \text{softmax}(\mathbf{W}^{(2)}\, \sigma(\mathbf{W}^{(1)} \mathbf{h}_i + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}) \tag{2.8}$$

$$\mathbf{C}_i = \Big[\sum_{j \neq i} \alpha_j \mathbf{h}_j \,\Big\|\, \mathbf{h}_i \Big]. \tag{2.9}$$

Then, for each (trigger, entity mention) pair, they calculated the entity mention vector and trigger vector by max-pooling the context vectors of the words in

the subsequence along the sequence length dimension. The two vectors were concatenated and fed into the role classifier.

One can also determine the contributions of the incoming edges as the attention between the end nodes. Non-Local Neural Networks (NLNN) [58] are a generalization of attention-based approaches. They have the following configurations at each block k:

- The scalar pairwise interaction function f and the non-pairwise function g as ϕ^e but only taking $\mathbf{v}_{r_k}$ and $\mathbf{v}_{s_k}$ as input,
- Normalized weighted elementwise summation for $\rho^{e \to v}$,
- The update function U_k as ϕ^v.

Specifically, in simpler notations,

$$\alpha_{ij}^{(k)} = f(\mathbf{h}_i^{(k-1)}, \mathbf{h}_j^{(k-1)}) \tag{2.10}$$

$$\mathbf{b}_i^{(k)} = g(\mathbf{h}_i^{(k-1)}) \tag{2.11}$$

$$\mathbf{h}_j^{(k)} = U_k\Big(\frac{1}{\sum_{i \in NI(j)} \alpha_{ij}^{(k)}} \sum_{i \in NI(j)} \alpha_{ij}^{(k)} \mathbf{b}_i^{(k)}\Big) + \mathbf{h}_j^{(k-1)} \tag{2.12}$$

Each node update is based on a normalized weighted sum of its incoming neighbors whose representations may be transformed, plus the residual connection term [Equation 2.12]. [Equation 2.10] denotes the weight or the unnormalized attention coefficient.

In this paper, I use a variant of Graph Attention Network (GAT) [52] that performs masked self-attention on the neighbor nodes. GATs follow the NLNN

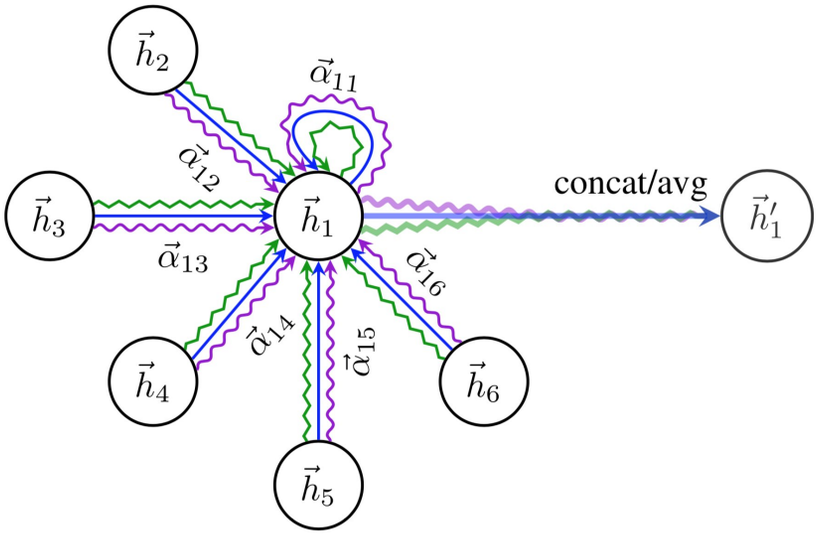

Figure 2.3: **GAT iteration** with 3 heads denoted in different colors. Figure from [52].

framework.

$$\alpha_{ij} = \underset{i \in NI(j)}{\text{softmax}}\Big(\text{LeakyReLU}(\mathbf{a}^{\text{T}}[\mathbf{W}\mathbf{h}_i \| \mathbf{W}\mathbf{h}_j])\Big) \tag{2.13}$$

$$\mathbf{h}_j = \sigma\Big(\sum_{i \in NI(j)} \alpha_{ij} \mathbf{W}\mathbf{h}_j\Big) \tag{2.14}$$

In the original paper, the attention mechanism is a single-layer FFN, parameterized by a weight vector $\mathbf{a} \in \mathbb{R}^{2d}$ and applying the LeakyReLU nonlinearity [Equation 2.13]. This term is, of course, subject to change.

In the original NLNN and GAT papers, edge features are solely determined by the node features of their ends. Still, explicit edge features could be used by adjusting f and/or g to be a function of $\mathbf{e}_k$. For example, Wang et al. [57] proposed Relational GAT, where node update involves an additional edge

attention mechanism that solely works on edge features.

$$\alpha_{ij}^{\text{rel}} = \underset{i \in NI(j)}{\text{softmax}}\Big(\sigma(\mathbf{W}^{(2)}(\text{ReLU}(\mathbf{W}^{(1)}\mathbf{h}_{ij} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}))\Big) \tag{2.15}$$

$$\mathbf{h}_j^{\text{rel}(k)} = \sum_{i \in NI(j)} \alpha_{ij}^{\text{rel}} \mathbf{W}\mathbf{h}_j^{\text{rel}(k)} \tag{2.16}$$

$$\mathbf{h}_j^{(k)} = \text{ReLU}\Big(\mathbf{W}^{(k)}[\mathbf{h}_j^{\text{att}(k)} \| \mathbf{h}_j^{\text{rel}(k)}] + \mathbf{b}^{(k)}\Big) \tag{2.17}$$

GATs operating on dependency graphs have not been used in EE but were applied to other NLP tasks, including sentence classification [17, 57] and preference classification [31].

Veyseh et al. [53], the state-of-the-art work for the role classification subtask, used a Graph Transformer Network (GTN) [67]. GTN is a spectral convolution approach where the convolution layer can be interpreted as the composition of fixed convolutions. The forward propagation in a spectral GCN [19] is defined as

$$\mathbf{H}^{(k)} = \sigma(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(k-1)}\mathbf{W}^{(k-1)}) \tag{2.18}$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I} \in \mathbb{R}^{|V| \times |V|}$ is the adjacency matrix with added self-loops, normalized by the degree matrix $\tilde{\mathbf{D}}$ where each entry on the diagonal $\tilde{\mathbf{D}}_{ii} = \sum_j \tilde{\mathbf{A}}_{ij}$.

GTN uses multiple adjacency matrices corresponding to distinct edge types. Veyseh et al. [53] used 4 types of adjacency matrices.

- Canonical binary adjacency matrix $\mathbf{A}^{\text{d}}$, where $\mathbf{A}_{ij}^{\text{d}} = 1$ if e_{ij} exists in the dependency tree and 0 otherwise.

- Entity-specific structure matrix $\mathbf{A}^{\mathrm{e}}$, where $\mathbf{A}^{\mathrm{e}}_{ij}$ is a learned function of $\mathbf{d}_i$ and $\mathbf{d}_j$. $\mathbf{d}_i$ is the embedded path length from the given entity to word w_i on the dependency graph.

- Trigger-specific structure matrix $\mathbf{A}^{\mathrm{t}}$, likewise.

- Semantic structure matrix $\mathbf{A}^{\mathrm{s}}$, where $\mathbf{A}^{\mathrm{s}}_{ij}$ denotes the self-attention score mediated by control vectors $\mathbf{c}$ specific to each (trigger, entity mention) pair. This is to filter relevant information about the given trigger and entity from word representations.

$$\mathbf{c}^{\mathrm{q}} = \sigma(\mathbf{V}_{\mathrm{q}}[\mathbf{h}_e \| \mathbf{h}_t]), \quad \mathbf{q}_i = \mathbf{c}^{\mathrm{q}}\mathbf{W}_{\mathrm{q}}\mathbf{h}_i \tag{2.19}$$

$$\mathbf{c}^{\mathrm{k}} = \sigma(\mathbf{V}_{\mathrm{k}}[\mathbf{h}_e \| \mathbf{h}_t]), \quad \mathbf{k}_i = \mathbf{c}^{\mathrm{k}}\mathbf{W}_{\mathrm{k}}\mathbf{h}_i \tag{2.20}$$

$$\mathbf{A}^{\mathrm{s}}_{ij} = \underset{j}{\mathrm{softmax}}(\mathbf{q}_j^{\mathrm{T}}\mathbf{k}_i) \tag{2.21}$$

A meta-path $v_0 \xrightarrow{t_1} v_1 \xrightarrow{t_2} v_2 \cdots \xrightarrow{t_l} v_l$ defines a composite relation $t_1 \circ t_2 \circ \cdots \circ t_l$ between the nodes. The adjacency matrix of the meta-path can be obtained by multiplying the corresponding adjacency matrices $\mathbf{A}_{t_1} \cdots \mathbf{A}_{t_k}$.

$$\mathbf{Q}^{(k)} = \sum_{t_l \in \mathcal{T}^e} \alpha^{(k)}_{t_l} \mathbf{A}_{t_l} \tag{2.22}$$

$$\mathbf{Q} = \prod_{k=1}^{K} Q^{(k)} \tag{2.23}$$

To denote an arbitrary meta-path, GTN learns a new graph structure at each layer as a convex combination of the adjacency matrices [Equation 2.22]. Then, the final adjacency matrix after stacking K layers is obtained by multiplying the newly learned structures [Equation 2.23]. This step is applied for multiple

channels by applying multi-channel 1×1 convolution to the adjacency matrices, hence $\mathbf{Q} \in \mathbb{R}^{|V| \times |V| \times C}$ (C being the number of channels). Finally, a spectral GCN is applied to each channel. Where $\tilde{\mathbf{Q}}^{(c)} = \mathbf{Q}^{(c)} + \mathbf{I}$ is the c-th channel from the final adjacency matrix, $\tilde{\mathbf{D}}$ is its degree matrix, $\mathbf{W} \in \mathbf{R}^{d \times d}$ is shared across channels, and $\mathbf{X} \in \mathbb{R}^{|V| \times d}$ is the feature matrix, the final node representations are obtained by concatenating all channels.

$$\mathbf{H} = \Big\Vert_{c} \sigma(\tilde{\mathbf{D}}^{(c)} \tilde{\mathbf{Q}}^{(c)} \mathbf{X} \mathbf{W}) \tag{2.24}$$

2.3 Model Design Specifics

2.3.1 Attention Flow

Jung et al. [18] introduced AttnIO for generating plausible dialog paths on knowledge graphs. It runs a GAT to update node representations. Besides, given a set of starting nodes denoting entities appearing in the last dialog, it models an *attention flow* from the starting nodes to the k-hop neighboring nodes. Here, attention indicates the relative importance of the neighboring nodes. The initial attention distribution is set to have all the probability mass (summing to 1) concentrated on the starting nodes. Attention is dispersed across nodes over several (2 in the original paper) computations on the graph. Following the nodes with the highest node attention value at each step gives a *path*.

Details are largely the same as [Equation 4.9] $\sim$ [Equation 4.12], with some differences:

- I add attention scaling as in Vaswani et al. [51] [Equation 4.10].

- AttnIO averages all heads in multi-head attention for computing the transition probability matrix.

$$T_{ij}^{(t)} = \frac{1}{M} \sum_{m=1}^{M} \underset{j \in NO(i)}{\text{softmax}}(\tau_{ij}^{(m,t)}) \tag{2.25}$$

  I instead maintain separate matrices for each head [Equation 4.10].

EAE involves capturing trigger-entity interactions as well as entity-entity interactions regarding the trigger. I use the attention flow mechanism to model the shift of such *trigger-relevant* information as path traversal starting from the trigger node and reaching down to other nodes.

2.3.2 Pretrained Deep Text Encoder

As presented in Section 2.1, it is important for an EE model to understand the contextual meanings of words. In this work, I incorporate the power of pretrained language models. Considering the just moderate size of ACE 2005, using a LM trained on a large amount of external resources particularly comes into handy. Namely, I use a pretrained BERT to better initialize node representations for the GNN. Veyseh et al. [53] reported +0.7 F1 gain in role classification for using BERT embeddings instead of Word2vec [35] embeddings as word representations.

2.3.3 Dependency Label as Edge Types

Although many previous works made use of dependency graphs [36, 45, 39, 28, 21, 6, 30, 53], most of them ignored dependency label information. Some works

[39, 28] used edge types in a minimal fashion with edge directions: head → dependent (as originally present in the dependency tree), dependent → head (added as reverse edge), or self-loop. Others did not use edge features at all. While not explicitly mentioned, this is likely due to a large number of dependency relation types compared to the just moderate dataset size. It is therefore critical to prevent overparameterization.

Cui et al. [6] modified the adjacency tensor used in GCNs to represent edge features. They constructed p-dimensional adjacency tensors $\mathbf{E} \in \mathbb{R}^{N \times N \times p}$ where $\mathbf{E}_{i,j,:}$ is initialized to the edge type embedding if a dependency edge e_{ij} exists, otherwise a zero-vector. The graph convolutional operation is then

$$\mathbf{H}_i^{(k)} = \mathbf{E}_{:,:,i}^{(k-1)} \mathbf{H}^{(k-1)} \mathbf{W} \tag{2.26}$$

$$\mathbf{H}^{(k)} = \sigma(\text{meanpool}(\mathbf{H}_1^{(k)}, \mathbf{H}_2^{(k)}, \cdots, \mathbf{H}_p^{(k)})) \tag{2.27}$$

where $\mathbf{E}_{:,:,i}$ denotes the i-th channel of the edge-aware adjacency matrix and $\mathbf{W} \in \mathbb{R}^{d \times d}$ is a learnable filter. Since only an embedding lookup table $\in \mathbb{R}^{\mathcal{T}_e \times p}$ needs to be trained, this approach is substantially parameter-efficient than RGCNs [42] where relation types are incorporated by using different learnable filters $\mathbf{W}_r$. Cui et al. reported +3.2 F1 increase for the type classification subtask compared to a RGCN baseline, and +1.9 F1 increase compared to a strong GCN-based baseline [64].

Inspired by this result, I also incorporate dependency label information with aware of overparameterization. I generate edge embeddings with basis decomposition as in CompGCN [50] and use the edge features as input for the attention

function [4].

2.3.4 Handling Role Overlap

Role overlap refers to the phenomenon where the offsets of distinct roles partially or entirely overlap. For example, in the sentence "The explosion killed the bomber and three shoppers", "the bomber" plays two roles at the same time: `Attacker` and `Victim`. Classifying each word and/or entity into one role as done in most exsiting works cannot handle role-overlapping events that account for about 10% of the ACE 2005 dataset [65]. To tackle this problem, I make separate node representations for different roles, and handle all representations with a single classifier. This approach is more parameter-efficient compared to training separate classifiers for different roles [65, 30].

2.3.5 Scalability

Veyseh et al. [53] used 4 adjacency matrices for their GTN-GCN model. 3 of them ($\mathbf{A}^{\mathrm{e}}$, $\mathbf{A}^{\mathrm{t}}$, $\mathbf{A}^{\mathrm{s}}$) are specific to each (trigger, entity mention) pair. Therefore, adding an argument candidate to be tested requires an additional run through the whole pipeline. I design a more *scalable* model in the sense that all node representations are computed at one sweep over the GNN module regardless of the number of argument candidates.

In addition, unlike all previous works where multi-layer GNNs with layer-specific parameters were used, the GNN module in this paper has only one layer. In canonical GNNs, the number of layers inevitably affects performance

[4] See Paragraph 4.1 for details.

by limiting the number of iterations on the graphs; the fewer the iterations, the less the message propagates. This is harming when treating long sequences with deep syntactic trees. In contrast, I design a model where all iterations take place in one layer, and suggest a *scaling* mechanism which allows a large number of iterations to be performed without the risk of message over-propagation. Therefore, the proposed model is expectedly scalable to longer sequences.

Chapter 3

Dataset

The ACE 2005 dataset [55] is the standard benchmark for event extraction. It consists of texts from various fields including newswire, broadcast news, broadcast conversation, weblog, discussion forums, and conversational telephone speech.

	Split		
	Train	Dev	Test
Document	529	30	40
Sentence	14,724	875	713
Trigger	4,420	505	424
Argument	7,816	933	892
Entity mention	53,045	4,050	4,226

Table 3.1: **Dataset size.**

Following previous work, I use the data split shared in the literature since [5]. I use Stanford CoreNLP 3.9.2 [32] for preprocessing (chunking and dependency parsing). I use the enhanced++ dependency parser [43]. Among 224

relations that are in the training set, I select 112 relations with ≥ 10 training samples. I set the remaining dependency labels to `UNK`, including those that appear only in validation/test sets. [Figure 3.1] shows the training set statistics of the dependency labels used in this work.

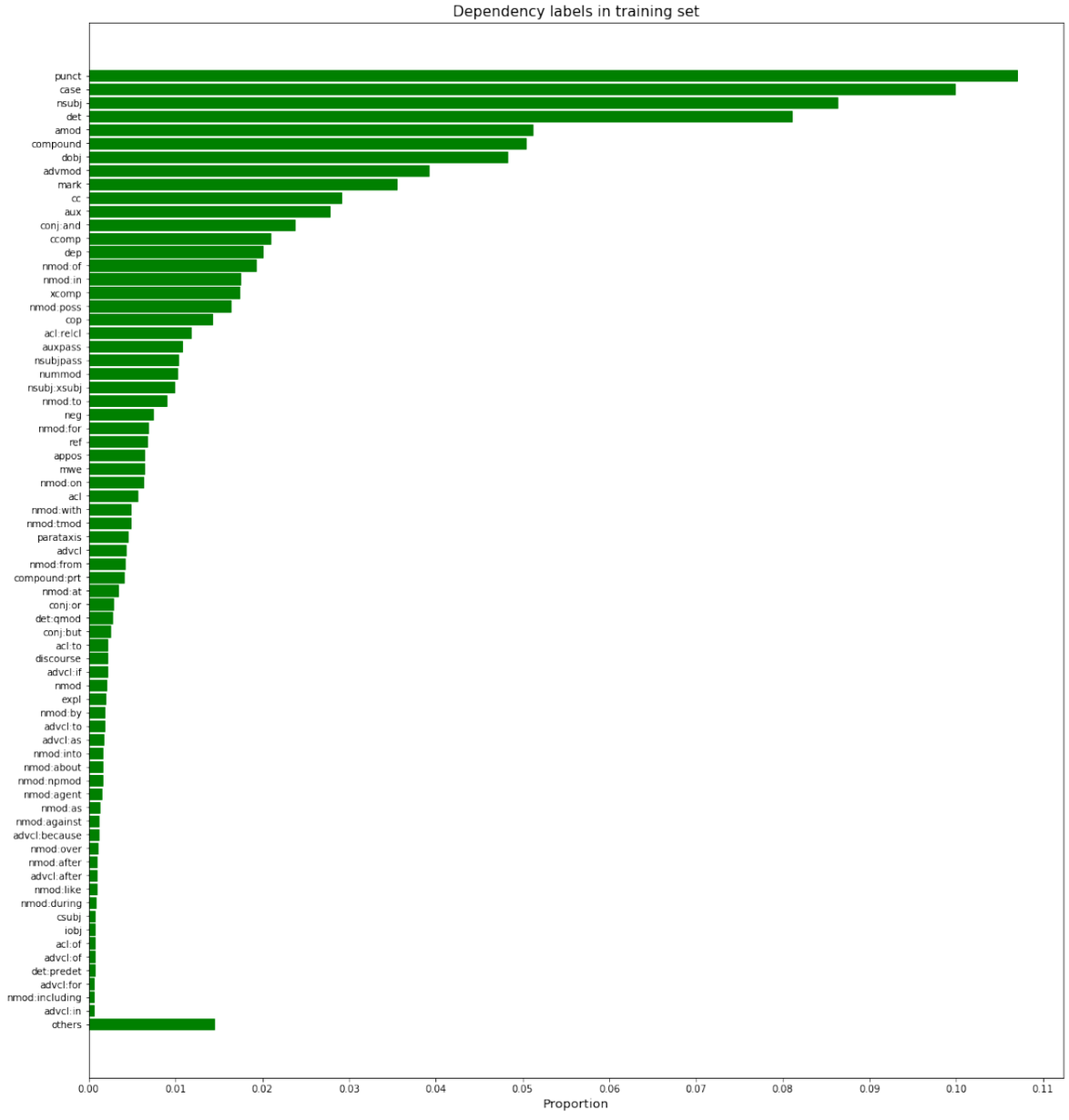

Figure 3.1: **Dependency label statistics** for the training set. Labels that account for $< 0.05\%$ of the dataset is marked as "others".

Following previous works, I add reverse edges to allow information flow against the annotated direction. A reverse edge share the same type with the corresponding original edge but have different direction information. I also add self-loops to prevent over-propagation, plus `Is-Right/Is-Left` edges between consecutive words to denote linear sequence information. Unlike a normal dependency tree, the resulting graph is not a simple graph anymore.

Category	**#Class**	**Specification**
Event	34	33 annotated + `No-Event`
Relation	115	112 annotated + `Self-Loop, Is-Right, Is-Left`
Role	37	35 annotated + `Trigger, No-Role`

Table 3.2: **Number of classes used.** Classes that are not originally annotated in the dataset are added only for training purposes and are not used during evaluation.

Not all roles are applicable to all event types. I make a list of valid roles for each event type, extracted from the official annotation guidelines [26]. I use this information during role classification so that the role predictions for a predicted event type are made within the valid role pool.

Event type	**Role**
Life	
`Be-Born`	`Person`
`Marry`	`Person`

Divorce	Person
Injure	Agent, Victim, Instrument
Die	Agent, Victim, Instrument
Movement	
Transport	Agent, Artifact, Vehicle, Price, Origin, Destination, -Place
Transaction	Beneficiary
Transfer-Ownership	Buyer, Seller, Artifact, Price
Transfer-Money	Giver, Recipient, Money
Business	Org
Start-Org	Agent
Merge-Org	
Declare-Bankruptcy	
End-Org	
Conflict	
Attack	Attacker, Target, Instrument
Demonstrate	Entity
Contact	Entity
Meet	
Phone-Write	-Place
Personnel	Person, Entity, Position
Start-Position	
End-Position	

Nominate	
Elect	
Justice	Crime
Arrest-Jail	Person, Agent
Release-Parole	Person, Entity
Trial-Hearing	Defendant, Prosecutor, Adjudicator
Charge-Indict	Defendant, Prosecutor, Adjudicator
Sue	Plaintiff, Defendant, Adjudicator
Convict	Defendant, Adjudicator
Sentence	Defendant, Adjudicator, Sentence
Fine	Entity, Adjudicator, Money
Execute	Person, Agent
Extradite	Agent, Person, Destination, Origin
Acquit	Defendant, Adjudicator
Pardon	Defendant, Adjudicator
Appeal	Defendant, Prosecutor, Adjudicator

Table 3.3: **Full list of main and sub-event types and their valid roles.** The roles written at a main event type row are shared by the subevents that belong to that category. The 8 time-related roles `Time-At-Beginning`, `Time-Within`, `Time-Ending`, `Time-Holds`, `Time-At-End`, `Time-Starting`, `Time-Before`, `Time-After` are implied for all event types. The role `Place` is implied unless marked as `-Place`.

There are two settings for EAE on ACE 2005: with or without gold entity mentions. Experiments in this paper follow the former setting where offsets and

types of entity mentions are available to the model. As arguments are selected among entity mentions, gold entity information relieves the burden of predicting the exact argument offsets.

Chapter 4

Model

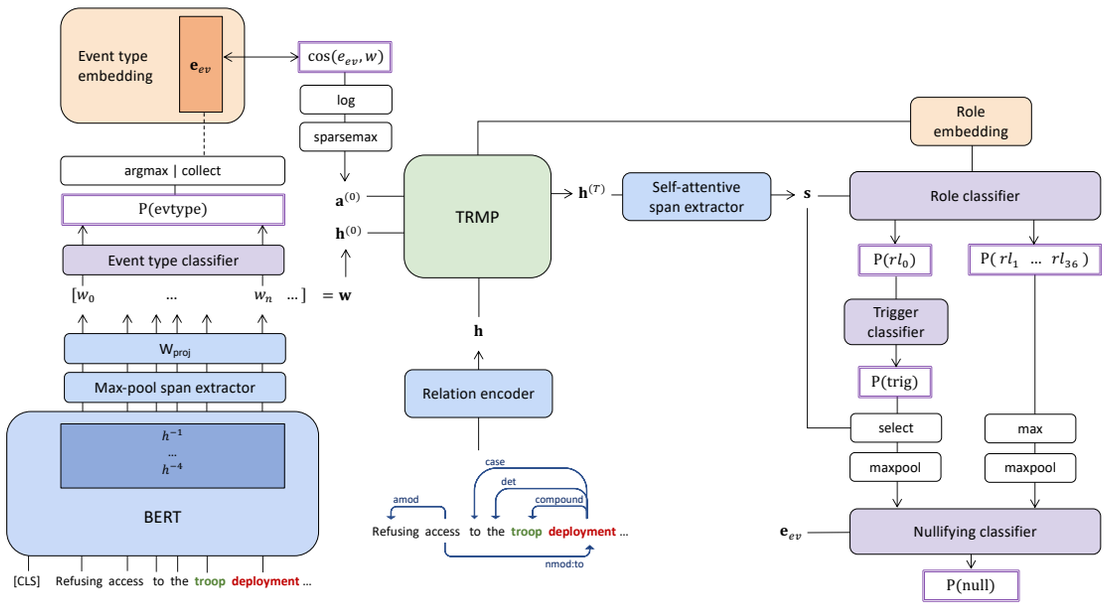

Figure 4.1: Full architecture. Values in purple double-bordered boxes are trained with objectives.

[Figure 4.1] shows the full architecture. It consists of three parts: 1) a trigger-relevant message passing (`TRMP`) module to encode dependency graphs, 2) pre-`TRMP` encoder modules to generate inputs for the `TRMP` module, and 3) post-`TRMP` decoder modules to make predictions out of final node representations from the `TRMP` module.

4.1 Encoder Modules

The encoder modules prepare ingredients for the `TRMP` module: initial node representations, their attention values, preliminary predictions of event type, and edge representations.

Initial node representations. First, a deep text encoder generates contextualized representations of every token in a source sentence. I use `bert-large-uncased-whole-word-masking` pretrained by HuggingFace [61]. I concatenate the top n_l layers for token representations. BERT parameters in the top n_l layers are finetuned, and the rest are frozen during training.

Next, a span extractor combines the subword token representations into whole word representations by max-pooling along the sequence length dimension. The outputs from the span extractor are projected to dimension d with $\mathbf{W}_{\text{proj}} \in \mathbb{R}^{d \times d_{\text{BERT}} \times n_l}$. With each word as a graph node, the projected outputs are used as initial graph node representations $\mathbf{h}^{(0)} \in \mathbb{R}^d$.

Event type prediction. Word representations are fed into an event type classifier that predicts one event type per word. This corresponds to mining the lexical properties of words (predicates). The probability of w_i bearing an event of type ev is:

$$p_i^{ev} = \underset{ev \in \mathcal{T}_{\text{event}}}{\text{softmax}}\big(\mathbf{W}^{(1)}_{\text{evtype}} \text{GeLU}(\mathbf{W}^{(2)}_{\text{evtype}} \mathbf{h}_i^{(1)} + \mathbf{b}^{(1)}_{\text{evtype}}) + \mathbf{b}^{(2)}_{\text{evtype}}\big). \tag{4.1}$$

I empirically find out that this classifier has the tendency of over-predicting events. Therefore, I make decisions from the event type classifier non-final and

allow the predicted events to be *nullified* at post-TRMP decoding phase. See Paragraph 4.3 for details.

As multiple events can exist in one sentence, a sample is duplicated by the number of predicted event mentions so that one graph represents a single event. I use gold event samples and their trigger offsets at training time. I also add one erroneous sample per batch in order to suggest the model to fix its initial event type prediction if needed. An erroneous sample is randomly selected from a pool of wrong event type predictions made in the batch. If all predictions are correct, an arbitary non-gold event is used.

Initial node attention distribution. TRMP requires that each node v_i holds an initial attention value $a_i^{(0)}$ denoting its probability of being the event trigger, so that $\sum_i a_i = 1$. This value is determined by the similarity between the node's initial representation $\mathbf{h}_i^{(0)}$ and the predicted event type embedding $\mathbf{e}_{ev} \in \mathbb{R}^d$. Let N_g be the graph node set. At training time,

$$a_i^{(0)} = \underset{k \in N_g}{\text{softmax}}(\log a_i') \tag{4.2}$$

$$a_i' = \begin{cases} \max\left(0, \cos\left(\mathbf{h}_i^{(0)}, \mathbf{e}_{ev}\right)\right) & \text{if } i \text{ is a trigger word and } ev \text{ is valid} \\ 0 & \text{otherwise.} \end{cases} \tag{4.3}$$

At inference time, I use sparsemax [34] to concentrate attention on a few nodes as in training time when only trigger words are given non-zero attention.

$$a_i' = \max\left(0, \cos\left(\mathbf{h}_i^{(0)}, \mathbf{e}_{ev}\right)\right) \tag{4.4}$$

$$a_i^{(0)} = \underset{k \in N_g}{\text{sparsemax}}(\log a_i') \tag{4.5}$$

Edge representations. To learn a large number of dependency relation types without overfitting, I reduce the number of parameters by generating edge embeddings with basis decomposition as in CompGCN [50].

$$\mathbf{z}_e = \sum_{b=1}^{B} \alpha_e^{(b)} \mathbf{v}^{(b)} \tag{4.6}$$

$$\mathbf{h}_{ij} = \mathbf{W}_{\text{dir}} \mathbf{z}_e \tag{4.7}$$

With basis transformations $\mathbf{v}^{(b)} \in \mathbb{R}^{d_b}$ and coefficient $\alpha_e^{(b)} \in \mathbb{R}$, representation of e_{ij} with type e is initially defined as their linear combination [Equation 4.6]. Then, it is transformed by weights corresponding to the edge direction $\mathbf{W}_{\text{dir}} \in \mathbb{R}^{d \times d_b}$ [33, 50].

$$\mathbf{W}_{\text{dir}}(e) = \begin{cases} \mathbf{W}_{\text{orig}} & \text{if } e \text{ is an original edge in the dependency parse tree} \\ \mathbf{W}_{\text{rev}} & \text{if } e \text{ is a reverse edge} \\ \mathbf{W}_{\text{self}} & \text{if } e \text{ is a self-loop} \end{cases} \tag{4.8}$$

For `UNK` edges, I average $\mathbf{z}$ of all originally annotated edge types (excluding `Self-Loop, Is-Right, Is-Left`).

4.2 Trigger-Relevant Message Passing

Trigger-relevant message passing (`TRMP`) is a GNN that iteratively propagates attention along the graph and updates node representations. Since the initial node attention values $\mathbf{a}^{(0)}$ are larger for predicted trigger nodes and are smaller (desirably, zero) for non-trigger nodes, this process models path traversal starting from the event trigger, hence the name *trigger-relevant*. Unlike canonical

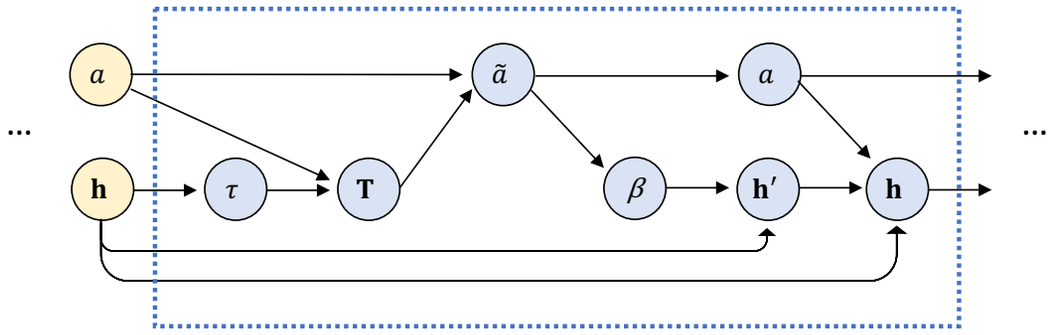

Figure 4.2: `TRMP` **iteration.** At each iteration, variables in blue are computed. Variables in yellow are from the previous iteration.

GNNs where one iteration corresponds to passing through one layer (each holding a distinctive set of parameters from other layers), iterations in `TRMP` altogether forms a *flow* in a single layer.

Addressing the role overlap problem, I maintain n_{rl} representations for each node so that $\mathbf{H}_i \in \mathbb{R}^{n_{rl} \times d_h}$. This allows a node to be classified into multiple roles. I use multi-head attention [51, 52] with n_h heads, each of dimension $d_h = d/n_h$. To prevent notational clutter, I hereafter use $\mathbf{h}_i \in \mathbb{R}^{d_h}$ to denote the representation of the node v_i corresponding to role rl in a single head.

The attention propagation mechanism largely follows AttnIO [18]. Where $NO(i)$ is the destination node set of outgoing edges from node v_i and $NI(j)$ is the source node set of incoming edges to node v_j, at each iteration $t \in \{1, \cdots, T\}$:

$$\tau_{ij}^{(t)} = \frac{(\mathbf{W}_{\mathrm{q}}(\mathbf{h}_i^{(t-1)} + \mathbf{e}_{rl}))^{\mathrm{T}}(\mathbf{W}_{\mathrm{k}}(\mathbf{h}_j^{(t-1)} + \mathbf{h}_{ij}))}{\sqrt{d_h}} \tag{4.9}$$

$$\mathbf{T}_{ij}^{(t)} = \underset{j \in NO(i)}{\mathrm{softmax}}(\tau_{ij}^{(t)}) \tag{4.10}$$

$$\tilde{a}_{ij}^{(t)} = T_{ij}^{(t)} \cdot a_i^{(t-1)} \quad \text{s.t.} \quad \sum_{i,j} \tilde{a}_{ij}^{(t)} = 1 \tag{4.11}$$

$$a_j^{(t)} = \sum_{i \in NI_j} \tilde{a}_{ij}^{(t)}. \tag{4.12}$$

I compute the transition score from node v_i to a neighboring node v_j in respect with a query role rl with dot-product attention. Such function design reflects the following question: "I am looking for a particular role at this node; which neighboring node should I be interested in, considering the node semantics and the edge type?" I form the query with node representation of v_i and the role embedding $\mathbf{e}_{rl} \in \mathbb{R}^{d_h}$ of interest, and the key with edge representation of e_{ij} and its destination node v_j. I compute the scaled dot-product of the query and the key after transforming them with $\mathbf{W}_{\mathrm{q}}$ and $\mathbf{W}_{\mathrm{k}} \in \mathbb{R}^{d_h \times d_h}$, respectively [Equation 4.9]. Scaling with the head dimension prevents the softmax function from having a very small gradient which hinders efficient training in case of large inputs [51]. Transition scores are normalized over all neighboring nodes so that $\sum_j \mathbf{T}_{ij}^{(t)} = 1$, resulting in transition probabilities [Equation 4.10]. Then, the amount of attention to propagate from v_i to its outgoing edge e_{ij} is computed with the transition probability T_{ij} and the node attention value that v_i was originally holding [Equation 4.11]. Finally, the node attention value of each node is updated with an aggregation of the attention on incoming edges [Equation

4.12].

At every iteration, GNNs update node representations by aggregating message from incoming nodes. Unlike in AttnIO where attention flow does not affect node updates, `TRMP` controls the amount of message to be delivered according the propagated attention values. Here, node updates come to reflect path traversal history.

$$\beta_{ij}^{(t)} = \underset{i \in NI_j}{\text{softmax}}(\log \tilde{a}_{ij}^{(t)}) \tag{4.13}$$

$$\mathbf{h'}_j^{(t)} = \sum_{i \in NI_j} \beta_{ij}^{(t)} \cdot \mathbf{W}_{\text{v}} \mathbf{h}_i^{(t-1)} \tag{4.14}$$

$$\mathbf{h}_j^{(t)} = \sqrt{a}_j^{(t)} \cdot \mathbf{h'}_j^{(t)} + (1 - \sqrt{a}_j^{(t)}) \cdot \mathbf{h}_j^{(t-1)} \tag{4.15}$$

I first compute the message coefficient determining the amount of information to be delivered from v_i to v_j [Equation 4.13]. The function is designed to be zero when unattended, *i.e.* $\tilde{a}_{ij} = 0$. Representations of neighboring nodes in NI_j are transformed with $\mathbf{W}_{\text{v}} \in \mathbb{R}^{d_h \times d_h}$ and are aggregated [Equation 4.14]. The aggregated message $\mathbf{h'}_j$ is reflected to the new node representation by the amount proportional to the attention value that v_j holds.

Note that the original node representation is added with a scaling factor. This is to ensure $\mathbf{h}_j$ to not change when no attention propagates into v_j at this iteration, *i.e.* $a_j = 0$ [Equation 4.15]. This procedure naturally mitigates the over-smoothing problem [23, 56] that deep GNNs typically suffer from. Over-smoothing refers to node representations becoming undesriably indistinguishable after several iterations of canonical message passing as the nodes get information from their k-hop neighbors after k iteratons. `TRMP` prevents unnec-

essary node updates by making the node representations remain the same until the attention flow hits them.

After a sufficiently large number of iterations, the final node representation $\mathbf{h}_i \in \mathbb{R}^d$ is obtained by concatenating the representations $\mathbf{h}_i^{(T)} \in \mathbb{R}^{d_b}$ from all heads.

4.3 Decoder Modules

The decoder modules predict the argument offsets and roles, and decide whether the graphs are valid event samples.

Span representations. For each entity mention *ent* that spans over position $[s, e)$, I compute the span representation $\mathbf{s}_{ent}$ with self-attention on the final representations of words inside the span. With $\mathbf{v}_a \in \mathbb{R}^d$,

$$\alpha_i = \underset{i \in [s,e)}{\text{softmax}}(\mathbf{v}_a^{\mathrm{T}} \mathbf{h}_i) \tag{4.16}$$

$$\mathbf{s}_{ent} = \sum_{i \in [s,e)} \alpha_i \mathbf{h}_i. \tag{4.17}$$

Note that span representations are unique to each role.

Role classification. The role classifier gets the concatenation of the span representation $\mathbf{s}_{ent} \in \mathbb{R}^d$ and role embedding $\mathbf{e}_{rl} \in \mathbb{R}^d$ as input, and computes the probability of the entity *ent* being an argument of the specified role *rl*.

$$p_{ent}^{(rl)} = \text{sigmoid}\big(\mathbf{W}_{\text{rl}}^{(2)} \text{GeLU}(\mathbf{W}_{\text{rl}}^{(1)} \big[\mathbf{s}_{ent} \,\big\|\, \mathbf{e}_{rl}\big] + \mathbf{b}_{\text{rl}}^{(1)}) + \mathbf{b}_{\text{rl}}^{(2)}\big) \tag{4.18}$$

Probabilities for roles that are not assigned to the current predicted event type are manually set to 0. This allows the model to predict roles only within the valid role pool as in [Table 3.3];

Finalizing event type prediction. I make use of two pooled representations for finalizing the event type prediction for the current graph: (1) one on entity mentions, and (2) one on predicted trigger nodes. The rationale is to obtain a graph-level representation based on the elements that together define an event: namely, the trigger word/phrase, and all the entity mentions with respect to their predicted roles.

First, for each entity mention, I select the role-specific span representation of the highest probability among 36 non-`Trigger` roles. `No-Role` is included in order to reflect whether an entity mention is a valid argument or not. I max-pool the selected role representations of all entities. Let the pooled representation be $\mathbf{c}_{\text{ent}}$.

For each word, I compute the probability of belonging to the trigger word/phrase. The trigger classifier gets the node representations corresponding to `Trigger` as input.

$$p_i = \text{sigmoid}\big(\mathbf{W}^{(2)}_{\text{trig}}\text{GeLU}(\mathbf{W}^{(1)}_{\text{trig}}\mathbf{h}_i + \mathbf{b}^{(1)}_{\text{trig}}) + \mathbf{b}^{(2)}_{\text{trig}}\big) \tag{4.19}$$

I max-pool the representations of words with $p \geq 0.5$. Let the pooled representation be $\mathbf{c}_{\text{trig}}$. If $p_i < 0.5 \ \ \forall i$, I choose the node with the highest probability.

Finally, I concatenate the two pooled representations with the previously predicted event type embedding $\mathbf{e}_{ev}$ and feed it into a nullifying classifier. The

outputs from the nullifying classifier determines whether this graph is a valid event or not.

$$p = \text{sigmoid}\big(\mathbf{W}^{(2)}_{\text{null}}\text{GeLU}(\mathbf{W}^{(1)}_{\text{null}}[\mathbf{c}_{\text{ent}} \,\|\, \mathbf{c}_{\text{trig}} \,\|\, \mathbf{e}_{ev}] + \mathbf{b}^{(1)}_{\text{null}}) + \mathbf{b}^{(2)}_{\text{null}}\big) \tag{4.20}$$

if $p > 0.5$, this sample is removed from the final predictions.

4.4 Training Objectives

The full model is trained in an end-to-end fashion on a joint of 5 objectives: $\mathcal{L}_{\text{evtype}} + \mathcal{L}_{\cos} + \mathcal{L}_{\text{rl}} + \mathcal{L}_{\text{trig}} + \mathcal{L}_{\text{null}}$.

Due to class imbalance, the event type classifier is trained with a biased cross entropy loss $\mathcal{L}_{\text{evtype}}$. The bias strengthens the influence of less prevalent labels during training. Where $p_i^{(v)}$ is the probability of the word w_i having the event type v as computed from the event type classifier, and $N(v)$ is the number of samples for event type v in the training set,

$$\gamma_v = 1 - \frac{N(v)}{\sum_v N(v)} \tag{4.21}$$

$$\mathcal{L}_{\text{evtype}}(i) = \sum_v \gamma_v \log p_i^{(v)} \tag{4.22}$$

$$\mathcal{L}_{\text{evtype}} = \sum_i \mathcal{L}_{\text{evtype}}(i). \tag{4.23}$$

The event type embedding is randomly initialized and trained with a cosine

embedding loss $\mathcal{L}_{\cos}$.

$$\mathcal{L}_{\cos}(i, ev) = \begin{cases} 1 - \cos\left(\mathbf{h}_i^{(0)}, \mathbf{e}_{ev}\right) & \text{if } i \text{ is a trigger word and } ev \text{ is valid} \\ \max\left(0, \cos\left(\mathbf{h}_i^{(0)}, \mathbf{e}_{ev}\right)\right) & \text{otherwise} \end{cases} \tag{4.24}$$

$$\mathcal{L}_{\cos} = \sum_{i} \sum_{ev} \mathcal{L}_{\cos}(i, ev). \tag{4.25}$$

The role classifier is trained with a binary cross entropy loss $\mathcal{L}_{\text{rl}}$. For a certain role rl, where $y_{ent} \in \{0, 1\}$ denotes whether the entity ent has that reference role and p_{ent} denotes the probability of ent having rl as computed from the role classifier,

$$\mathcal{L}_{\text{rl}}(ent, rl) = -\Big(y_{ent} \log p_{ent} + (1 - y_{ent}) \log\left(1 - p_{ent}\right)\Big) \tag{4.26}$$

$$\mathcal{L}_{\text{rl}} = \sum_{ent} \sum_{rl} \mathcal{L}_{\text{rl}}(ent, rl). \tag{4.27}$$

The trigger classifier is trained with a binary cross entropy loss $\mathcal{L}_{\text{trig}}$. Where $y_i \in \{0, 1\}$ denotes whether the word w_i is part of the trigger and p_i denotes the probability of w_i being the trigger as computed from the trigger classifier,

$$\mathcal{L}_{\text{trig}}(i) = -\Big(y_i \log p_i + (1 - y_i) \log\left(1 - p_i\right)\Big) \tag{4.28}$$

$$\mathcal{L}_{\text{trig}} = \sum_{i} \mathcal{L}_{\text{trig}}(i). \tag{4.29}$$

The nullifying classifier is trained with a binary cross entropy loss $\mathcal{L}_{\text{null}}$. Where $y \in \{0, 1\}$ denotes whether the event type for the current graph sample is wrongly predicted, *i.e.* the current graph is an invalid sample, and p denotes the probability of nullyfing the current graph as computed from the nullyfing

classifier,

$$\mathcal{L}_{\text{null}} = -\Big(y \log p + (1-y) \log (1-p)\Big). \tag{4.30}$$

Chapter 5

Experiments

5.1 Training Details

BERT embedding size (d_{BERT})	1024
number of BERT layers (n_l)	4 (from top)
graph representation size (d) (node, edge, event type, role)	512
edge basis embedding size (d_B)	64
number of edge bases (n_b)	20
number of attention heads (n_h)	8
number of `TRMP` iterations (T)	15
optimizer	AdamP [14]
learning rate	2e-4
lr scheduler	cosine
weight decay	0.1
gradient clip norm	1.0
batch size	8
number of gradient accumulation steps	2

number of epochs	100

Table 5.1: **Hyperparameters.**

5.2 Results

I report the precision (P), recall (R), and F1 (F1) scores on the ACE 2005 test set. I validate the model every epoch and use the checkpoint with the best validation F1 on role classification. [Table 5.2] show the results.

Model	**Argument Identification**			**Role Classification**		
	P	R	F1	P	R	F1
Chen et al.† [5]	68.8	51.9	59.1	62.2	46.9	53.5
Sha et al.† [44]	63.2	59.4	61.2	54.1	53.5	53.8
Nguyen et al. [36]	61.4	64.2	62.8	54.2	56.7	55.4
Liu et al.† [27]	63.0	64.2	63.6	-	-	-
Yang et al.† [65]	**71.4**	60.1	65.3	62.3	54.2	58.0
Sha et al. [45]	71.3	64.5	67.7	66.2	52.8	58.7
Wang et al.† [60]	-	-	-	62.2	56.6	59.3
Liu et al. [28]	**71.4**	65.6	**68.4**	66.8	54.9	60.3
Ma et al. [30]	64.8	63.7	64.2	61.1	60.6	60.8
Veyseh et al. [53]	-	-	-	**69.3**	55.9	61.9
`TRMP`	70.83	61.17	65.65	66.71	58.18	**62.16**
– dependency label	66.41	59.82	62.94	61.56	55.72	58.49

– dependency graph	45.25	**67.61**	54.21	40.15	**61.21**	48.49
– BERT contextualization	52.16	44.24	47.87	45.34	40.19	42.61
– nullify	64.04	58.24	61.00	59.10	54.82	56.88

Table 5.2: **EAE results on ACE 2005 test set** under the gold entity setting. Models that do not use dependency graphs are marked with †. Rows marked with – show ablation results.

`TRMP` achieves the highest F1 score on the role classification task.

F1 Performance on argument identification does not reach state-of-the-art, however, and without a clear explanation for now. `TRMP` is somewhat comparable to Liu et al. in terms of precision, but has a notably lower recall by $>$ 4pt. Considering that capturing the associations between triggers (events) that are present together in a sentence is what Liu et al.'s work focused on[1], one may infer that lack of explicit modelling of event co-occurrence in this paper is to blame; the node representations in each sample graph are specific to the predicted event type, and the only source of possibilities of other events in the same sentence is the contextual information from pretrained BERT. However, this possibility can be ruled out by comparing the performance on sentences with one event with those with two or more events: argument identification performance is higher for the latter sentences (77.35/62.38/69.06 (P/R/F1)) compared to the former (62.70/59.31/60.96), both in precision and recall.

[1]Some events co-occur more with certain events but not others, *e.g.* `Injure` and `Die` are likely to co-occur; `Attack` and `Born` are not. Liu et al. made use of trigger context vectors computed through self-attention with other trigger candidates.

Ablation. I perform several ablation studies to configure the rationale of this work.

[– dependency label] shows the results when edge directions (original, reverse, or self-loop) instead of dependency labels are used as edge types [39, 28]. As the number of relations is small (only 3), I make a $\mathbb{R}^{3\times d}$ size edge embedding instead of using basis decomposition. F1 scores on both subtasks substantially decrease, empirically demonstrating the power of exploiting rich dependency label information.

When fully-connected graphs are used in place of dependency graphs [– dependency graph], recall improves but precision drops by a larger margin. Here, edge features $\mathbf{h}_{ij}$ are removed from [Equation 4.9]. This shows that `TRMP`'s outperforming all baselines that do not use dependency graphs does not solely comes from using an alternative architecture. Still, higher recall in this setting implies that not all informational relations are covered by head–dependency relations.

[– BERT contextualization] shows the results when initial node representations are generated only from the WordPiece [62] embeddings at BERT layer 0. The performance significantly drops. This shows the importance of having well-contextualized node embeddings at the first place, plus the benefits from exploiting a a deep LM encoder pretrained on huge external corpora.

Lastly, removing post-`TRMP` event nullifying (*i.e.* the model is trained without $\mathcal{L}_{\text{trig}}$ and $\mathcal{L}_{\text{null}}$) hurts performance, especially precision [– nullify]. This shows that `TRMP` effectively regularizes detection of events.

5.3 Attention Visualization

For an intuitive presentation of the path traversal process in TRMP, I make a case study with visualization of node attentions. Consider the following sentence:

Early Saturday[Time-Holds], more units[Attacker] were waiting
in Kuwait[Place] to *smash through*[Trigger] any Iraqi resistance.

This sentence contains a Conflict:Attack event triggered by "smash through" and with 3 arguments of distinct roles.

[Figure 5.1] shows attention flow on node representations corresponding to the role Place. Each subfigure shows node attention distribution averaged over all heads at every designated step, where a darker node means stronger attention. At the beginning (step 0), all attention is concentrated on the node "smash". This indicates that the model predicts "summit" as the trigger word. At step 1, attention is disseminated to direct neighbors of "summit". At step 2, attention reaches "Kuwait", which is at 2-hop from the predicted trigger. Around step 8, "Kuwait" is strongly attended. The attention distribution converges and does not change much until the last step, demonstrating that TRMP is capable of preventing over-propagation. As a result, the entity "Kuwait" is correctly classified as Place.

Note that EAE succeeds even though the event trigger offsets were not exactly correct (failure of ED); only the semantically informative "smash" part out of "smash through" holds non-zero initial attention.

Now I turn to a less straightforward case. [Figure 5.2] shows attention flow on

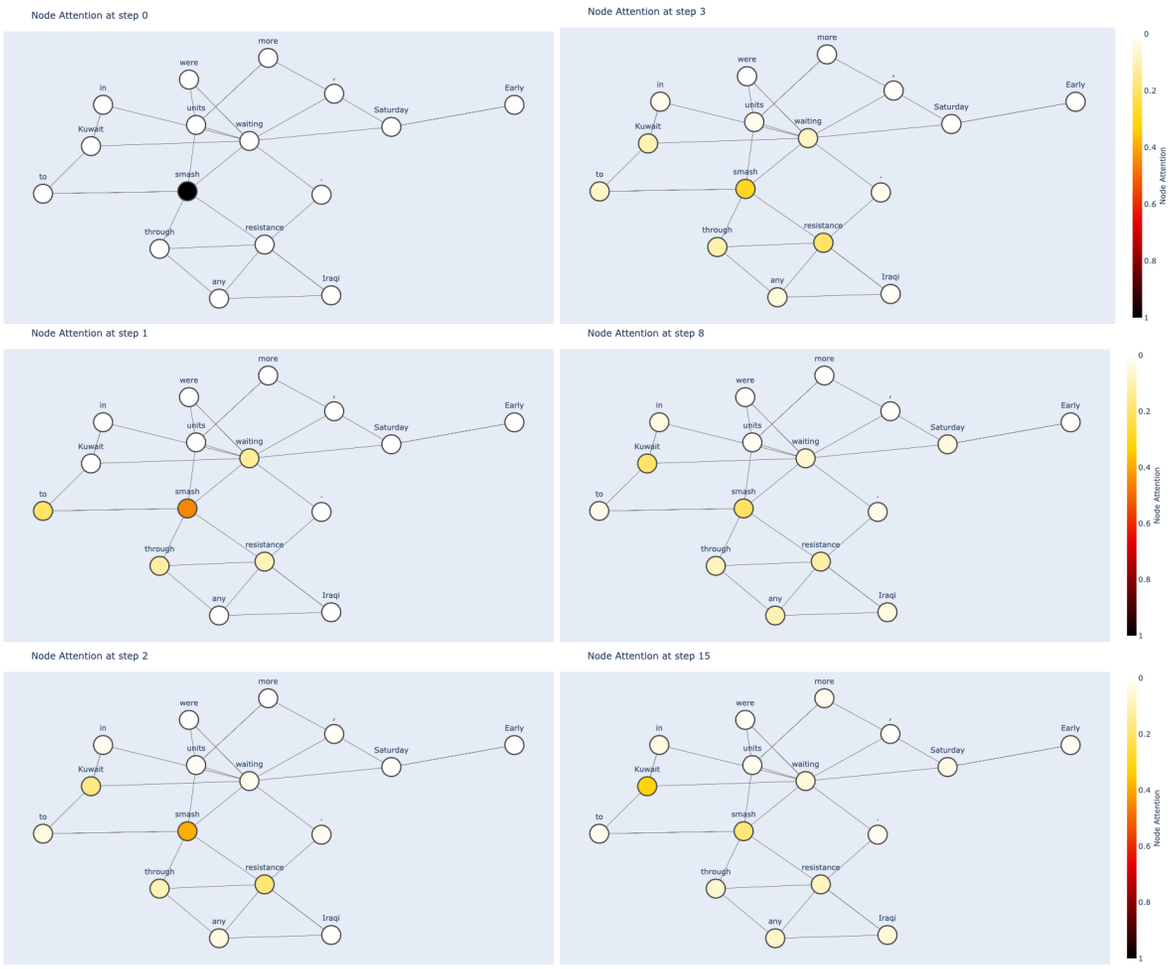

Figure 5.1: **Node attention history for role `Place`.** Self-loops are removed to prevent visual clutter.

node representations corresponding to role `Attacker`. Attention converges with high value on "units" but not on "more" (step 10). Still, with the information that "more" and "units" together form an entity mention, high attention on "units" is sufficient to generate an adequate span representation for that entity to be correctly classified as `Attacker`. Under the alternative setting without gold entities, the model might fail to identify the argument offsets.

[Figure 5.3] is a case where role classification fails. The first 5 subfigures show attention flow on node representations corresponding to the gold role

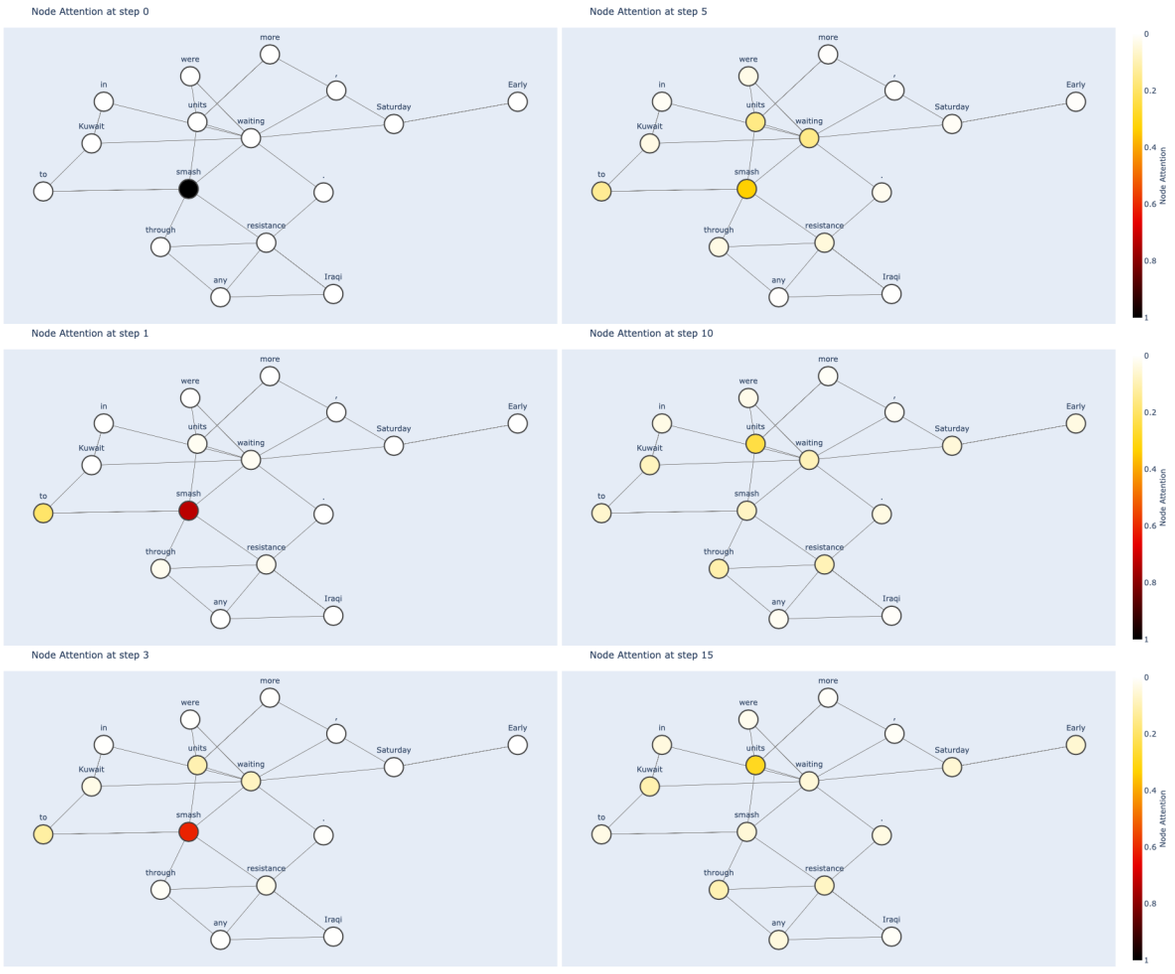

Figure 5.2: **Node attention history for role `Attacker`.**

`Time-Holds`. "Early" and "Saturday" are supposed to have high attention values, but neither of them actually are until the last iteration. Instead, the two nodes have a moderate level of attention for the role `Time-Within` which is the model's actual prediction.

Now I present a sample where roles overlap.

U.S.[`Attacker`] aircraft[`Instrument`] *bombed*[`Conflict:Attack`]

Iraqi tanks holding bridges close to the city.

Figure 5.3: **Node attention history for roles `Time-Holds` and `Time-Within`.** The first 5 subfigures and the last 1 subfigure with red border, respectively.

In this sentence, two arguments with distinct roles overlap at the word “U.S.”.

[Figure 5.4] shows attention flow on node representations corresponding to role `Instrument`, and [Figure 5.5] is for role `Attacker`. The attention distribubtion for the two roles are similar until step 6. At the final step, for the role `Instrument`, “aircraft” gets a high level of attention while “U.S.” does not. For the role `Attacker`, “U.S.” gets the most attention. As a result, the one-word entity “U.S.” is predicted to have the `Attacker` role but not `Instrument`, whereas

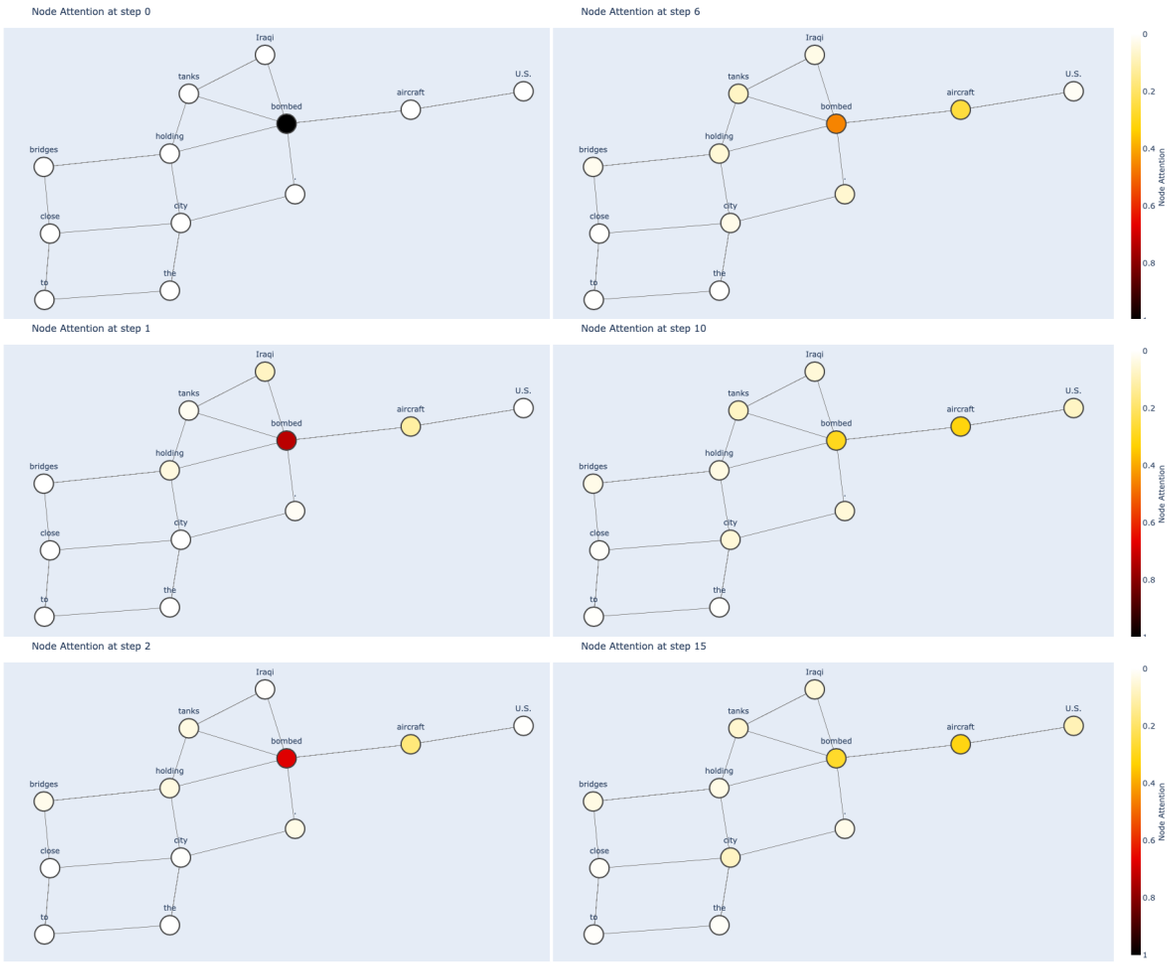

Figure 5.4: **Node attention history for role `Instrument`.**

the two-word entity "U.S. aircraft" is predicted to have the `Instrument` role due to selective attention on its component word "aircraft".

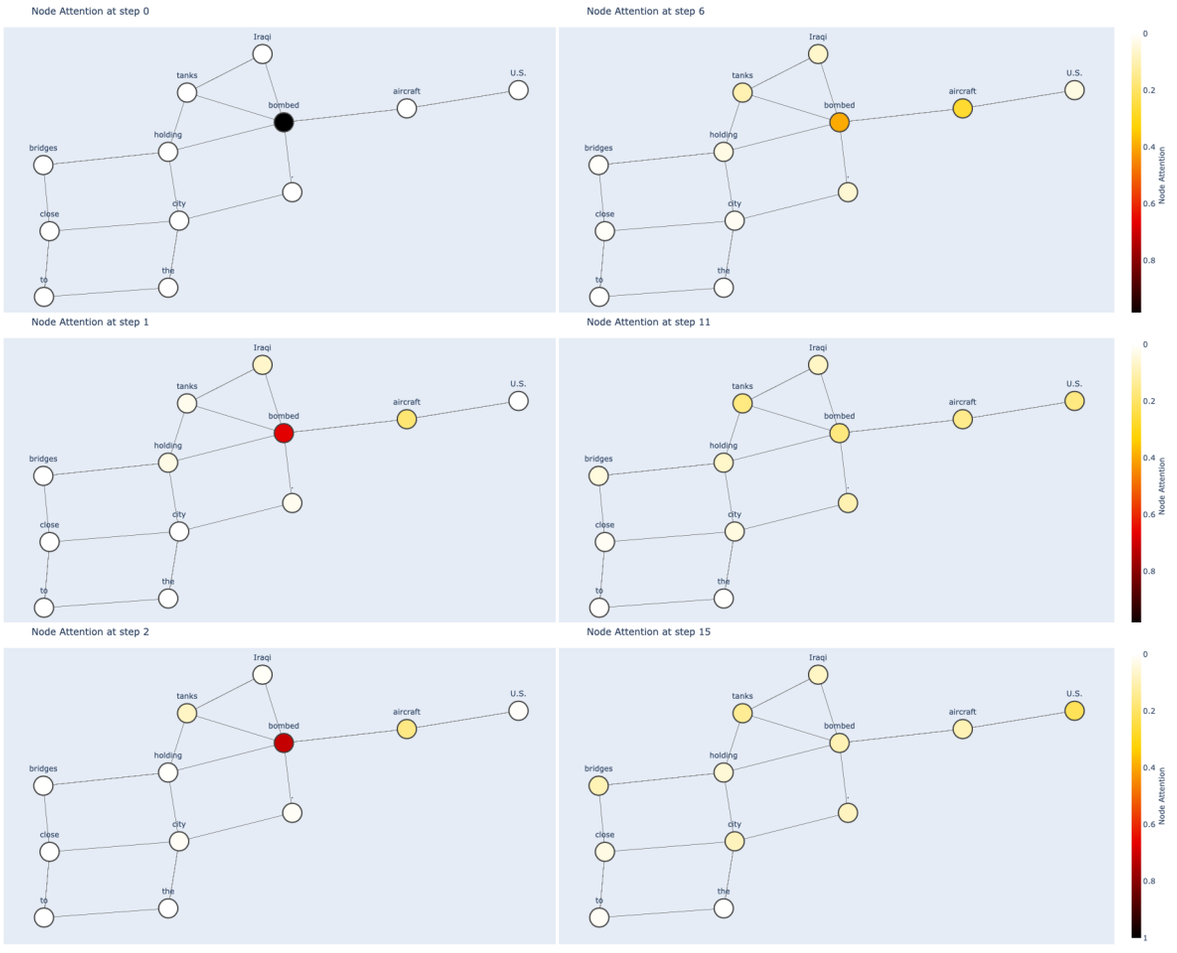

Figure 5.5: **Node attention history for role `Attacker`.**

Chapter 6

Conclusion

In this paper, I proposed an EAE model that captures trigger-relevant interactions with attention flow on dependency graphs. In this framework, a graph sample is generated for every predicted event, where each node is given an attention value that corresponds to the probability of being the trigger. The Trigger-Relevant Message Passing (`TRMP`) algorithm defines a path traversal by means of attention propagation starting from the predicted trigger nodes. By using the propagated attention values for interposing node updates, `TRMP` makes the node representations reflect the path traversal history and prevents unnecessary node updates that could lead to node-smoothing, even with a large number of iterations. I also incorporate previous attempts in EE including using pretrained LMs, dependency label information and handling role overlaps.

The proposed model achieves the strongest F1 performance for the role classification task. It is also preferrable to the previous SOTA model [53] in terms of scalability.

A major limitation of this work comes from the ACE 2005 dataset. The dataset is neither comprehensive to be used for real-world applications, nor

compact to be readily beneficial to theoretical linguistics. Although it is the benchmark for EE, it has a limited number of event types and is overly focused on the legal domain. At the same time, a large portion of the defined argument roles are in fact obliques (*e.g.* 8 time-related roles), and the roles do not comply with the literature on semantic roles. Fortunately, attempts to build more comprehensive datasets for event extraction recently have taken place. Events from FrameNet [1] have received attention [24]; Wang et al. [59] actually constructed the MAVEN dataset for ED that has 118K samples of 168 event types.

Applying `TRMP` to more general datasets would be a valuable addition. With a proper dataset, path traversal on graphs would provide interpretable data for reverse engineering the argument realization process. As this work benefits from using the *full* dependency structures including the edge labels, analyzing the attention transition at each step according to edge types potentially benefits syntax-semantic interface studies.

References

[1] Collin F. Baker, Charles J. Fillmore, and John B. Lowe. The berkeley framenet project. In *Proceedings of ACL-COLING*, pages 86–90, 1998.

[2] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[3] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. In *IEEE Transactions on Neural Networks*, volume 5, pages 157–166, 1994.

[4] Yubo Chen, Shulin Liu, Shizhu He, Kang Liu, and Jun Zhao. Event extraction via bidirectional long short-term memory tensor neural networks. In *China National Conference on Chinese Computational Linguistics Inter-*

national Symposium on Natural Language Processing Based on Naturally Annotated Big Data, pages 190–203, 2016.

[5] Yubo Chen, Liheng Xu, Kang Liu, Daojian Zeng, and Jun Zhao. 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 (Volume 1: Long Papers)*, pages 167–176, 2015.

[6] Shiyao Cui, Bowen Yu, Tingwen Liu, Zhenyu Zhang, Xuebin Wang, and Jinqiao Shi. Edge-enhanced graph convolution networks for event detection with syntactic relation. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2329–2339, 2020.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. 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)*, pages 4171–4186, 2019.

[8] George Doddington, Alexis Mitchell, Mark Przybocki, Lance Ramshaw, Stephanie Strassel, and Ralph Weischedel. The automatic content extraction (ace) program- tasks, data, and evaluation. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC'04)*, 2004.

[9] Xinya Du and Claire Cardie. Event extraction by answering (almost) natural questions. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 671–683, 2020.

[10] Charles J. Fillmore. *The Case for Case*, pages 1–88. New York: Holt, Rinehart, and Winston, 1968.

[11] Reza Ghaeini, Xiaoli Fern, Liang Huang, and Prasad Tadepalli. Event nugget detection with forward-backward recurrent neural networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 369–373, 2016.

[12] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

[13] Jane Grimshaw. Semantic structure and semantic content, 1993.

[14] Byeongho Heo, Sanghyuk Chun, Seong Joon Oh, Dongyoon Han, Sangdoo Yun, Gyuwan Kim, Youngjung Uh, and Jung-Woo Ha. Adamp: Slowing down the slowdown for momentum optimizers on scale-invariant weights. In *International Conference on Learning Representations (ICLR)*, 2021.

[15] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. In *Neural Computation*, volume 9, pages 1735–1780, 1997.

[16] Malka Rappaport Hovav and Beth Levin. *Building Verb Meanings*. Stanford, California: CSLI Publications, 1998.

[17] Binxuan Huang and Kathleen M. Carley. Syntax-aware aspect level sentiment classification with graph attention networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pages 5469–5477, 2019.

[18] Jaehun Jung, Bokyung Son, and Sungwon Lyu. Attnio: Knowledge graph exploration with in-and-out attention flow for knowledge-grounded dialogue. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3484–3497, 2020.

[19] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *26st Conference on Neural Information Processing Systems (NIPS 2012)*, 2012.

[21] Viet Dac Lai, Tuan Ngo Nguyen, and Thien Huu Nguyen. 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 (EMNLP)*, pages 5405–5411, 2020.

[22] Lishang Li, Yang Liu, and Meiyue Qin. Extracting biomedical events with parallel multi-pooling convolutional neural networks. In *IEEE/ACM*

transactions on computational biology and bioinformatics, volume 17, pages 599–607, 2020.

[23] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *The Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, pages 3538–3545, 2018.

[24] Wei Li, Dezhi Cheng, Lei He, Yuanzhuo Wang, and Xiaolong Jin. Joint event extraction based on hierarchical event schemas from framenet. In *IEEE Access*, volume 7, pages 25001–25015, 2019.

[25] Jimmy Lin. *Event Structure and the Encoding of Arguments: The Syntax of the Mandarin and English Verb Phrase.* PhD thesis, Massachusetts Institute of Technology, 2004.

[26] Linguistic Data Consortium. *ACE (Automatic Content Extraction) English Annotation Guidelines for Events: Version 5.4.3*, 2015.

[27] Jian Liu, Yubo Chen, Kang Liu, Wei Bi, and Xiaojiang Liu. Event extraction as machine reading comprehension. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1641—1651, 2020.

[28] Xiao Liu, Zhunchen Luo, and Heyan Huang. 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, 2018.

[29] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[30] Jie Ma, Shuai Wang, Rishita Anubhai, Miguel Ballesteros, and Yaser Al-Onaizan. Resource-enhanced neural model for event argument extraction. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3554—-3559, 2020.

[31] Nianzu Ma, Sahisnu Mazumder, Hao Wang, and Bing Liu. Entity-aware dependency-based deep graph attention network for comparative preference classification. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5782–5788, 2020.

[32] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, 2014.

[33] Diego Marcheggiani and Ivan Titov. Encoding sentences with graph convolutional networks for semantic role labeling. In *Proceedings of the 2017*

Conference on Empirical Methods in Natural Language Processing, pages 1506–1515, 2017.

[34] André F. T. Martins and Ramón F. Astudillo. From softmax to sparsemax: A sparse model of attention and multi-label classification. In *Proceedings of the 33rd International Conference on Machine Learning*, pages 1614–1623, 2016.

[35] Tomas Mikolov, Kaim Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *International Conference on Learning Representations (ICLR)*, 2013.

[36] Thien Huu Nguyen, Kyunghyun Cho, and Ralph Grishman. 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, 2016.

[37] Thien Huu Nguyen and Ralph Grishman. Event detection and domain adaptation with 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*, pages 365–371, 2015.

[38] Thien Huu Nguyen and Ralph Grishman. Modeling skip-grams for event detection with convolutional neural networks. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 886–891, 2016.

[39] Thien Huu Nguyen and Ralph Grishman. Graph convolutional networks with argument-aware pooling for event detection. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[40] Trung Minh Nguyen and Thien Huu Nguyen. One for all: Neural joint modeling of entities and events. In *The Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)*, 2019.

[41] David Perlmutter and Paul Postal. *The 1-Advancement Exclusiveness Law*, pages 81–125. Chicago, Illinois: University of Chicago Press, 1984.

[42] Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. *arXiv preprint arXiv:1703.06103*, 2017.

[43] Sebastian Schuster and Christopher D. Manning. Enhanced english universal dependencies: An improved representation for natural language understanding tasks. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 2371–2378, 2016.

[44] Lei Sha, Jing Liu, Chin-Yew Lin, Sujian Li, Baobao Chang, and Zhifang Sui. Rbpb: Regularization-based pattern balancing method for event extraction. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1224–1234, 2016.

[45] Lei Sha, Feng Qian, Baobao Chang, and Zhifang Sui. Jointly extracting event triggers and arguments by dependency-bridge rnn and tensor-based argument interaction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[46] Chao Shang, Yun Tang, Jing Huang, Jinbo Bi, Xiaodong He, and Bowen Zhou. End-to-end structure-aware convolutional networks for knowledge base completion. In *The Thirty-Third AAAI Conference on Artificial Intelligence (AAAI 2019)*, 2019.

[47] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1556–1566, 2015.

[48] Ian Tenney, Dipanjan Das, and Ellie Pavlick. Bert rediscovers the classical nlp pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4593–4601, 2019.

[49] Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R. Thomas McCoy, Najoung Kim, Benjamin Van Durme, Samuel R. Bowman, Dipanjan Das, and Ellie Pavlick. What do you learn from context? probing for sentence structure in contextualized word representations. In *International Conference on Learning Representations (ICLR)*, 2019.

[50] Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha Talukdar. Composition-based multi-relational graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2020.

[51] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jacob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *31st Conference on Neural Information Processing Systems (NIPS 2017)*, pages 5998–6008, 2017.

[52] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations (ICLR)*, 2018.

[53] Amir Pouran Ben Veyseh, Tuan Ngo Nguyen, and Thien Huu Nguyen. Graph transformer networks with syntactic and semantic structures for event argument extraction. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, page 3651–3661, 2020.

[54] David Wadden, Ulm Wennberg, Yi Luna, and Hannaneh Hajishirzi. Entity, relation, and event extraction with contextualized span representations. 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 5784–5789, 2019.

[55] Christopher Walker, Stephanie Strassel, Julie Medero, and Kazuaki Maeda. Ace 2005 multilingual training corpus ldc2006t06. *Linguistic Data Consortium*, page 57, 2006.

[56] Guangtao Wang, Rex Ying, Jing Huang, and Jure Leskovec. Improving graph attention networks with large margin-based constraints. *arXiv preprint arXiv:1910.11945*, 2019.

[57] Kai Wang, Weizhou Shen, Yunyi Yang, Xiaojun Quan, and Rui Wang. Relational graph attention network for aspect-based sentiment analysis. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3229–3238, 2020.

[58] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[59] Xiaozhi Wang, Ziqi Wang, Xu Han, Wangyi Jiang, Rong Han, Zhiyuan Liu, Juanzi Li, Peng Li, Yankai Lin, and Jie Zhou. Maven: A massive general domain event detection dataset. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pages 1652–1671, 2020.

[60] Xiaozhi Wang, Ziqi Wang, Xu Han, Zhiyuan Liu, Juanzi Li, Peng Li, Maosong Sun, Jie Zhou, and Xiang Ren. Hmeae: Hierarchical modular event argument extraction. 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 5777—5783, 2019.

[61] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davidson, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing.

[62] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Quin Gao, Klaus Macherey, Jeff Klinger, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[63] Wei Xiang and Bang Wang. A survey of event extraction from text. In *IEEE Access*, volume 7, pages 173111–173137, 2018.

[64] Haoran Yan, Xiaolong Jin, Xiangbin Meng, Jiafeng Guo, and Xueqi Cheng. 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 5766–5770, 2019.

[65] Sen Yang, Dawei Feng, Linbo Qiao, Zhigang Kan, and Dongsheng Li. 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, 2019.

[66] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*, 2019.

[67] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J. Kim. Graph transformer networks. In *33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*, pages 11960–11970, 2019.

[68] Wenbo Zhang, Xiao Ding, and Ting Liu. Learning target-dependent sentence representations for chinese event detection. In *Information Retrieval*, pages 251–262, 2018.

국문초록

사건 논항 추출(EAE)은 자연어 텍스트로부터 일련의 사건들과 그에 참여하는 논항들을 추출하고, 논항들이 수행하는 역할을 예측하는 과제이다. 본 논문은 의존 구문 분석 트리를 EAE에 사용하는 연구들의 계열에 놓여 있다. 기존 연구들은 의존 관계 레이블을 활용하지 않고 의존 구문 분석 트리를 동종 그래프로 다루었으나, 본 논문은 레이블 정보를 엣지의 타입으로 활용한다. 본 논문이 제안하는 모델은 의존 구문 분석 그래프 위에서 어텐션(attention)의 흐름을 통해 단어간의 상호작용을 모델링하며, 해석 가능하다. 이 모델은 이들 상호작용이 사건의 종류와 사건 술어의 뜻에 의해 결정됨에 착안하여 사건 술어와 관련된 방식으로 단어의 표상을 계산한다. 사건 술어 관련 메시지 전달(TRMP) 모듈은 모델의 핵심으로, 의존 구문 분석 그래프 위에서 어텐션을 전파하고, 전파된 어텐션에 의거해 노드들의 표상을 업데이트하는 그래프 신경망이다. 제안된 모델은 EAE의 두 과제 중 의미역 분류에서 현재 최고 수준의 성능을 내며, 이 결과는 의존 관계 레이블을 사용하는 것이 효과적임을 보인다.

주요어: 사건 논항 추출, 사건 추출, 의존 구문 분석, 어텐션 흐름, 메시지 전달, 그래프 신경망, 딥 러닝

학번: 2018-23942