AN ABSTRACT OF THE DISSERTATION OF

J. Walker Orr for the degree of Doctor of Philosophy in Computer Science presented on June 21 2019.

Title: Towards Narrative Understanding with Deep Neural Networks and Hidden Markov Models

Abstract approved: _____

Prasad Tadepalli

Narratives are central to communication and the human experience. For a computer system to understand a narrative, it must be able to identify the key facts or plot elements that describe what happened or how the world has changed. These element are called events; establishing a document's events and the relationships between them is central to understanding a text's narrative. Events are related to each other temporally and causally by being part of the same story arc. Further, these event sequences typically follow patterns called scripts.

In this thesis, I explore three essential stages for narrative understanding. All three stages form an end-to-end system that starts with plain text documents and ultimately produces scripts, a generalization of narrative structure. The first two stages, event detection and event sequence extraction, analyze a document and extract the key information needed to understand a document's narrative. The final stage, script learning, generalizes the discovered event sequences to find common patterns between them.

First, I propose a neural network model based on grammar and syntax. It combines a left-to-right reading of the text along with a reading ordered by the sentence's syntactic tree. The model is an extension of Gated Recurrent Units and uses an attention mechanism to blend

both reading modes. This model achieves state-of-the-art performance on a well-studied task.

I present an evaluation that is the first to quantify the substantial variability of neural networks when applied to the nuanced problem of event detection. Two sources of variability are considered: the effect of local optimization of the neural networks' training procedure and the types of documents used for evaluation and training. I show that the variation involved is often greater than the differences in the state-of-the-art, demonstrating the need for a robust evaluation.

Second, the new task of event sequence extraction is addressed with a novel, interpretable neural network framework. The framework represents the problem as a series of graph transformations. By doing so, it allows for various neural network architectures to be combined while mirroring the structure of the task. Several models instantiated from the framework are evaluated against a strong baseline showing a substantial improvement on a difficult task. Further, I demonstrate the framework's flexibility by evaluating it on the entity relation extraction task.

Finally, I examine using Hidden Markov Models to learn scripts from event sequences with missing data. This formulation of scripts as Hidden Markov Models is novel and the first to explicitly account for missing observations in the context of natural language processing. The models are learned with a bottom-up induction algorithm based on Structural Expectation Maximization. The scripts are evaluated by inferring omitted events in event sequences and are shown to be more effective than an informed baseline.

©Copyright by J. Walker Orr
June 21 2019
All Rights Reserved

Towards Narrative Understanding with Deep Neural Networks and Hidden Markov Models

by

J. Walker Orr

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented June 21 2019
Commencement June 2020

Doctor of Philosophy dissertation of J. Walker Orr presented on June 21 2019.

APPROVED:

Major Professor, representing Computer Science

Head of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my dissertation will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my dissertation to any reader upon request.

J. Walker Orr, Author

ACKNOWLEDGEMENTS

I thank God for giving me the strength and perseverance to complete this work. My advisor Dr. Prasad Tadepalli was indispensable; his guidance was sincere, thoughtful and critical. He listened carefully to my ideas, regardless of their quality, and always gave constructive insight. The freedom and leeway he allowed gave me the opportunity to explore my own interests deeply. I cannot thank him enough.

Dr. Xiaoli Fern has been an excellent collaborator, she consistently provided helpful guidance and important correction. I am thankful for Dr. Thomas G. Dietterich and Dr. Alix Gitelman both for their vision and unique perspective. I am grateful for Dr. Matthew Campbell's willingness to be my Graduate Council Representative, for the time and flexibility he provided.

Altogether, I am thankful for the college of EECS at OSU. I appreciate the financial support of both graduate research and teaching assistance positions offered to me. Additionally, the facilities, both in terms of the beautiful building and the computational resources, benefited me greatly. The knowledge imparted through the instructors of the college was absolutely critical to my research. In addition to the those already mentioned, Dr. Fuxin Li's course on Deep Learning made most of this work possible. The clarity and insight from his course allowed me to adapt to a new paradigm and complete this work, saving me countless hours.

In general, I would like to thank the Statistics Department at OSU, completing a masters degree in statistics was both useful and personally rewarding. The knowledge I gained from their courses directly contributed to my research.

I had the privilege of meeting many wonderful students during my studies. I appreciate the good conversations, encouragement, and friendship of Sean McGregor, Jesse Hostetler, Aswin Raghavan. Also, Reza Ghaeini, Rasha Mohammad, Hamed Shahbazi, Jana Rao Doppa, and Chao Ma all provided me helpful criticism, insight, and practical advice.

During my internship at Johns Hopkins University, I had the opportunity to meet Dr. Frank Ferraro, Dr. Ben Van Durme, and Dr. Nathan Chambers. They gave me guidance and advice early in my studies which helped shape my present research.

Finally, I would like to thank my family and friends. My parents instilled the value of education which ultimately lead me to complete this work. I relied on their care and support throughout.

Completing this work simply would not have been possible without the love and support of my wife, Heather. Through both success and failure she was constant in her love and encouragement, I could not ask for a better partner in life. Last of all I would like to thank my daughter Kaelyn and my son John. Their hugs and smiles got me through the toughest times.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF APPENDIX TABLES

Chapter 1: Introduction

1.1 Motivation

Narratives range from mundane descriptions of common activities to grand stories that define peoples and cultures. The written word often takes a narrative form whether it be a correspondence between people, a news article, a historical text, or a work of fiction. Narratives are a framework for communication that span time and culture. It is theorized that narratives are how people store memories [74].

This dissertation defines narrative understanding as identifying the key elements of a story, the structure of an individual narrative, that is, the relationships between those elements, and finally the broader context or archetype to which the narrative belongs. This definition roughly corresponds to both a colloquial definition of understanding as well as a technical definition.

Besides the broader interest and implications of algorithmically understanding narratives, there are a variety of other applications. Extracting and representing narratives can be viewed as a comprehensive or holistic method for information extraction.

One such application of narrative understanding is text summarization. The key people, place, things and events of a document are a natural basis for summarization. Further, those elements, in conjunction with a document's narrative structure, could serve as a framework for comparison between documents. In short, narrative structure and elements would be a principled means of assessing the semantic similarity of two texts.

Additionally, narratives often omit "common-sense" or non-salient information for the sake of brevity. A model for narratives could be used to infer events or facts left out of individual stories. Co-reference relationships between entities could be identified by knowing how those entities relate to their roles within a narrative.

Finally, a model of narrative archetypes or scripts could be used as in generative framework for stories. In theory, a script could generate a story, a sequence of events with actors, and, in conjunction with a language model, create a textual representation of that story.

1.2 Natural Language Processing and Machine Reading

The three aspects of narrative understanding: identifying narrative elements, determining their relationships, and discovering the narrative archetypes, correspond in part to three separate natural language processing (NLP) tasks. They are event detection, relation extraction, and script learning. Each chapter of this thesis details a contribution to each of these tasks. When taken together, the systems presented can start with a plain document, identify events, place them in temporal order, and finally generalize them into a script.

This thesis is centered on several NLP concepts defined by the ACE2005 standard [84]. They are defined below:

- **Entity** - The concept of a person, place, or thing; ACE2005 defines seven types for entities: person, organization, geopolitical entity, location, facility, weapon, and vehicle. An entity is typically referenced by multiple entity mentions in the same document.
- **Entity Mention** - An entity rendered as a short span of text in the context of a document.
- **Event** - Something that happens in time and changes the state of the world. ACE2005 defines 33 types of events, they are enumerated in Table A.1.
- **Event Mention** - The instance of an event represented by a span of text in a document.
- **Relation** - An association between either a pair of entities or a pair of events. For this thesis, the relations considered between events are either "After" or "Parent-child" which indicate a temporal ordering or composition respectively. Relations between

entities are various kinds of composition and adjacency. More details are provided in Chapter 3.

Each of the following chapters is dedicated to a single NLP task: event detection, relation extraction, and script learning. Event detection involves identifying spans of texts that indicate an event occurred along with the type of the event. Recently, neural networks have achieved state-of-the-art performance. However, the models used have had three deficiencies. First, they are focused on individual sentences and ignore the larger document context. Second, the syntactic relationships found within the text are also ignored for a strict left-to-right reading. Finally, there is a high degree of variability in performance due to both the random initialization of the model and the differences in the distributions of the training and testing datasets.

For each of these drawbacks, Chapter 2 provides a solution. To incorporate the syntactic relationships between words, I purpose a model, DAG-GRU, that combines both a left-to-right reading of a sentence with a syntactic reading. The DAG-GRU model uses an attention mechanism to blend each reading into a more accurate representation.

Next, a variety of neural network models designed to represent an entire document were evaluated on the task of event detection. The intuition is that the broader document context can be useful for interpreting the meaning individual words. Three document level models implemented: doc2vec, EntNet, and Transformer. Of these models, EntNet was found to be on average the highest performing.

Finally, the variability of these neural network models was quantified. To address the two sources of variability considered, two experiential setting were employed. Overall, eight different models were implemented and evaluated. Five of those models are reproductions, two newly applied, and one new model was created for event detection. For the first experiment, each model was trained and evaluated with twenty different random initializations. The second experiment involved randomizing the training-testing partition ten times, again training and evaluating each model on each partition. It was shown that the variation was

often greater than the difference in the state-of-the-art models.

The next chapter focuses on two different formulations of relation extraction, event sequence extraction and entity relation extraction. I purpose a new framework for neural networks that represents relation extraction as a series of graph transformations. Each transformation corresponds to a task related to relation extraction. The transformations themselves are accomplished with a neural network. However, the framework is modular allowing for differing networks to be easily substituted to do different tasks. Various models produced by the framework were evaluated on the event sequence extraction task. One instance outperformed the baseline, the only previous work on the task. Additionally, the framework was competitive with a reproduction of a high performing model on the ACE2005 entity relation extraction task.

After identifying event sequences, Chapter 4 describes script learning which involves generalizing event sequences into archetypal patterns. Here scripts are presented for the first time as Hidden Markov Models (HMM). A novel formulation of HMMs allows missing and omitted events. The HMM are learned with a new structure learning algorithm based on Structural Expectation Maximization[29] and Bayesian Model Merging [79]. This method was evaluated against an informed baseline on the narrative cloze task, which involves filling in missing events in event sequences.

Overall, the primary contributions of this thesis are:

1. A new model, DAG-GRU, for event detection that incorporates syntax a robust evaluation and reproduction of many event detection models.

2. Demonstrating an on-average improvement over existing methods in event detection with the EntNet model.

3. A new modular neural network framework evaluated on two relation extraction tasks.

4. The formulation of scripts as Hidden Markov Models and an associated learning algorithm allowing for missing observations.

Chapter 2: Event Detection

2.1 Introduction

Events are the lingua franca of news stories and narratives and describe important changes of state in the world. Identifying events and classifying them into different types is a challenging aspect of understanding text. This chapter focuses on the task of *event detection*, which includes identifying the "trigger" words that indicate events and classifying the events into refined types. Event detection is the necessary first step in inferring more semantic information about the events including extracting the arguments of events and recognizing temporal and causal relationships between different events.

For event detection, ACE2005 provides both the primary definition of the task as well as a dataset for evaluation. One of the challenges is the size and sparsity of this dataset. It consists of 599 documents, which are broken into a training, development, testing split of 529, 30, and 40 respectively. This split has become a de-facto evaluation standard [47]. Furthermore, the test set is small and consists only of newswire documents, even though there are multiple domains within ACE2005. These two factors lead to a significant difference between the training and testing event type distribution. Though some work had been done comparing method across domains [62], variations in the training/test split including all the domains has not been studied.

There has been an explosion of different neural network models that address a variety of NLP tasks in recent years. Likewise, neural network models have been the most successful methods for event detection. Though these models have been successful, producing higher accuracy than previously achieved, their evaluation has been unsatisfactory. Due to the effect of random initializations as well as differences in training-testing splits, there is often significant variance in the models' performance. In this chapter, an empirical study of

the sensitivity of the system performance to the model initialization is conducted. Various neural network models are reproduced and put through a rigorous evaluation, quantifying the variance in their performance. Each model is evaluated in two ways, by averaging over twenty random initializations and over ten randomized training-testing splits.

Additionally, for further in-depth comparison of model performance, one additional evaluation was performed on a pair of models. The premise behind differing neural network architectures is that they capture different types of information. If that is the case, then significantly differing models should make different predictions on the same test data.

An analysis of each model's predictions was performed, comparing how often those models agreed with each other. Model instances at similar levels of performance are compared to eliminate accuracy as a factor that could influence the similarity or discrepancy between model predictions. Surprisingly, though the models have major differences in their network architectures, their agreement was high in both the cases when they were right and when they were wrong in their predictions. In some cases, the agreement was the same across different model types as within the same model type. This gives evidence that despite differences in network architectures, these models are largely capturing the same regularities in data. In addition to a more robust evaluation, this chapter purposes new models for event detection. Typically, current models work at the sentence level, ignoring the rest of the document. Further, the syntactic relationships in a text are also eschewed as they treat a sentence as simply as a sequence of words.

In order to model syntactic connections, a GRU-based model is proposed, called DAG-GRU, that captures both the context and the syntactic information through a bidirectional reading of the text with dependency parse relationships. A sequential reading of the text can be represented as a linear graph were each node is a word and each edge is an adjacency relationship. Dependency parse relationships are represented as additional edges between words. The DAG-GRU model combines the information communicated across multiple edges with the use of an attention mechanism.

Additionally, to establish a baseline of robust comparison, multiple sentence-level models were also reproduced. Three common models were implemented and evaluated: CNN [62], GRU [61], and a combination CNN+LSTM model [27]. A recent LSTM model that incorporates syntactic information [28] was also reproduced.

In addition to sentence-level models, several document-level models were also applied to the task of event detection and rigorously evaluated. It is easy to imagine that a document talking about political events might interpret the word "fire" as an employment termination event rather than a violent act. Since the topic of the document could conceivably be gleaned from any part of the document, it stands to reason that the overall context of the document might ease the event detection and classification.

This work considers three distinct neural network models to represent document-level context. Two of the three have not previously been applied to event detection though they have had considerable success in other NLP tasks, while the other is a replication of a recent method for event detection. Each model uses a unique network architecture suited to capture "long-distance" dependencies in the text. First, a document-level recurrent neural network model (RNN) [24] based on doc2vec [43] was reproduced. This model combines both word [58] and document embeddings as features for an LSTM [34] model. The document embeddings are intended to capture the semantics of the document, which provide context to individual predictions.

Second, as an alternative to representing the entire document as a vector, memory networks are designed to remember key portions of a document. One effective memory network design is the entity network model (EntNet), which has been shown to track both entities and important context in long term memory [33]. EntNet's memory blocks are built with simple gated recurrent neural networks and can target important pieces of document context with keyed gates. The memory blocks are accessed with an attention mechanism.

Finally, the transformer model is considered, which is based on self-attention [81]. Attention mechanisms are location-insensitive, allowing them to capture distant context in a

document. The self-attention mechanism allows for each word to be interpreted in light of every other word in the document. The transformer model combines self-attention with point-wise networks and residual links to build a representation of each word in a document contextualized by every other word.

Overall, there are four main conclusions in this chapter:

1. Random initializations and data splits are a significant source of variation, the spread is often larger than the differences in the state-of-the-art.

2. Document-level models, in particular EntNet, produce an on average improvement over existing approaches.

3. The DAG-GRU produces an on average improvement over sentence-level approaches.

4. The differences in neural network architecture yield modest but significant improvements in accuracy but are otherwise functionally equivalent according to the analysis of models' predictions.

2.2 Related Work

Event detection and extraction are well-studied tasks with a long history of research. Early works were typically classifiers that predicted an event type for all the candidate words in the dataset. The candidate word was represented by hand-crafted features based on lexical and syntactic information as well as corpus wide statistics or n-gram patterns[31, 1, 36, 13, 48, 35, 46, 11].

The state of the art system for several years has employed a joint model that combines event detection and argument extraction [20]. It is based on structured perceptron and uses inexact inference based on beam search [47]. The sentences are processed from left to right, where after each event is detected, its arguments are extracted.

Typical features are used to represent the document and candidate events and arguments

based on lexical, syntactic, and decision-history information. Furthermore, the model allows for complex relational features based on previously identified events and arguments. The most significant contribution is the use of a joint model to enable better inference. Continuing the thread of joint inference, a factor graph [86] was used to jointly identify events and entities. It also determines the argument roles between events and entities, adding to the evidence that event detection helps the extraction of arguments.

One of the weaknesses of these traditional machine learning approaches is that they required extensive feature engineering. Recently, neural network models have alleviated this problem by learning sophisticated, task-specific representations built on top of pre-trained word2vec vectors [58], producing competitive and better results. These methods turned their focus primarily to event detection rather than argument extraction.

For example, a dynamic convolutional neural network (CNN) model [12] uses a two part, branched model to represent the candidate event trigger and its containing sentence. The candidate trigger is represented with a word2vec [58] vector for the candidate and its adjacent neighboring words. For the sentence representation, the method applies a convolutional layer to each sentence followed by a dynamic k-maxpooling layer to select the most relevant parts of the convolutional feature map. In effect, it finds the most relevant parts of each sentence dynamically, summarizing the context of each side of the event candidate.

Similarly, a forward-backward RNN model [30] employs a pair of recurrent neural networks to represent the context immediately to the left and right of the candidate token.

Instead of dividing the region around the candidate trigger into two halves, another CNN model [62] builds a representation for the entire sentence. Three types of embeddings describe each word in the window: word, position, and entity type. Each word is represented by a concatenation of its word and entity type embeddings with the distance to candidate trigger. The distance measured in number of words from the word to the candidate trigger, has a randomly initialized embedding. Finally, the entity type taken from ACE2005's annotation, is also embedded starting with a random initialization. These concatenated embeddings are

the run for a convolutional layer of varying filter sizes. Global max-pooling summarizes the CNN filter and the result is passed to a linear classifier.

A follow-up approach based on a skip-gram based CNN model [63] allows the filters to skip non-salient or otherwise unnecessary words in the middle of word sequences. The model is largely the same as a previous CNN [62] model except for the use of non-consecutive filters.

A hybrid model [27] combined a previously used CNN [62] and a bi-directional LSTM [34] models to create a hybrid network. The outputs of both networks were concatenated together and fed to a linear model for final predictions.

A bidirectional GRU model [61] was used for sentence level encoding, and, in conjunction with a memory network, to jointly predict events and their arguments.

Also, a probabilistic soft logic model [50] was used to incorporate semantic frames from Framenet [5] in the form of extra training examples. Based on the intuition that entity and argument information is important for event detection, an attention model [51] was built to capture annotated arguments and the surrounding context to better model event trigger candidates.

To address the sparsity and limited size of the ACE2005 dataset, a cross language attention model [49] was created for event detection and was used for both the English and Chinese event detection tasks in the ACE2005 data. The model created a multi-lingual projection of the data via unsupervised alignments between languages, achieving high accuracy.

Another approach to address the minimal training data associated with the ACE2005 data studied jointly training several models on both the word sense disambiguation and event detection tasks [52]. Including joint training increased the performance of CNN, non-consecutive CNN, and hyrid CNN/RNN models.

A previous a drawback of early neural network methods was the lack of syntactic information. The early feature engineered systems included dependency parse information, but the neural network methods principally relied on word embeddings. Recently, a graph-CNN (GCCN) [64] model applies convolutional filters to syntactically dependent words in addition

to consecutive words. The addition of entity information into the network structure produced currently the best CNN model. Furthermore, the pooling layer of the model took the max of filters outputs over the annotated entities, rather than the whole sentence.

Another neural network model that includes syntactic dependency relationships is the DAG-based LSTM [28]. It combines the syntactic hidden vectors into a weighted average and adds them through a dependency gate to the output gate of the LSTM model. Finally, it uses a tensor to represent and jointly predict event-argument relationships, producing the current best performance for an RNN.

2.3 Models

The task of event detection is to predict, for each word in the corpus, whether or not it is an event trigger and if it is a trigger, its type. Hence, the task is formulated as a multi-class classification problem, where the input is a sequence of words $X = x_1 \dots x_n$ and the output is a sequence of event type labels, where the sequence of words is either a sentence or an entire document depending on whether the model is sentence-level or document-level respectively.

Each of the models considered uses the same multi-class classification setup. First the words of the document are embedded and represented as fixed k-length vectors. Second, the model takes the word embeddings X and produces a new representation of each word in the document, X'. Finally, dropout [78] with a rate of 0.5 is applied to the vectors during training, before a linear layer plus softmax activation is applied to each vector to make a classification. The difference between each approach is simply the representation X' it employs.

2.3.1 DAG GRU Model

The DAG-GRU model is an extension of the standard GRU model. It is a sentence-level model that operates on a directed acyclic graph rather than a sequential chain. The standard

GRU model works as follows:

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{2.1}$$

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{2.2}$$

$$\tilde{h}_t = \tanh(W_h x_t + r_t \odot U_h h_{t-1} + b_h) \tag{2.3}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{2.4}$$

The GRU model produces a hidden vector h_t of length d_h for each word x_t by combining its representation with the previous hidden vector. Thus h_t summarizes both the word and its prior context. In equation 2.4, h_t is calculated by making the trade-off between the previous hidden vector h_{t-1} and the candidate hidden vector $\tilde{h}_t$. The operator $\odot$ is element-wise product between vectors.

The "update gate" z_t and "reset gate" r_t are vectors where each element ranges from 0 to 1. In equation 2.2, z_t is computed as a function of the current word vector x_t and the previous hidden vector h_{t-1}. The "reset gate" r_t is computed similarly in equation 2.1. The matrices W_z, U_z, W_r, U_r along with the vectors b_z and b_r are parameters of model. Here σ is an element-wise application of the sigmoid function.

The candidate hidden vector $\tilde{h}_t$ is calculated by equation 2.3. $\tilde{h}_t$ is also function of w_t and h_{t-1}, however it also incorporates the "reset gate" r_t to determine how much of h_{t-1} to "forget." It also uses the parameters W_h, U_h and b_h. The function tanh is applied in an element-wise fashion.

All the W matrices are of size $d_h \times d_w$, where d_h and d_w are the size of the hidden and word vectors respectively. Likewise, the U matrices are of size $d_h \times d_h$ and the b vectors are of length d_h.

However, the GRU model reads the text in a sequential manner without considering syntax. One common representation of linguistic syntax is a dependency parse tree, as seen in Figure 2.1. The CoreNLP system [56] is used to produced a dependency parse tree for each

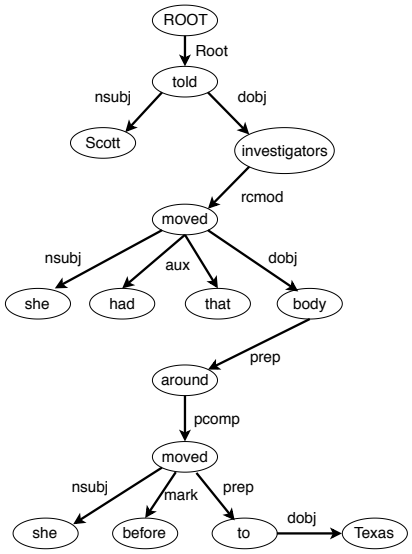

Figure 2.1: The dependency parse tree for the sentence "Scott told investigators she had moved that body around before she moved to Texas."

sentence in the ACE2005 corpus. Each node in the tree represents a word, and the directed edges between nodes are syntactic relationships. For example, if a "nsubj" edge indicates the child node is the subject of the parent node.

The proposed DAG-GRU model incorporates syntactic information through dependency parse relationships and is similar in spirit to GCNN [64] and DAG-LSTM [28]. However, unlike those methods, DAG-GRU uses attention to combine syntactic and temporal information. Rather than using an additional gate as in DAG-LSTM [28], DAG-GRU creates a single combined representation over previous hidden vectors and then applies the standard GRU model. Each relationship is represented as an edge, (t, t', e), between words at index t and t' with an edge type e. The standard GRU edges are included as $(t, t-1, temporal)$.

Each dependency relationship may be between any two words, which could produce a graph with cycles. However, back-propagation through time [60] requires a directed acyclic graph (DAG). Hence the sentence graph, consisting of temporal and dependency edges E, is split into two DAGs: a "forward" DAG G_f that consists of only of edges (t, t', e) where $t' < t$, and a corresponding "backward" DAG G_b where $t' > t$. The dependency relation between t and t' also includes the parent-child orientation, e.g., *nsubj-parent* or *nsubj-child*

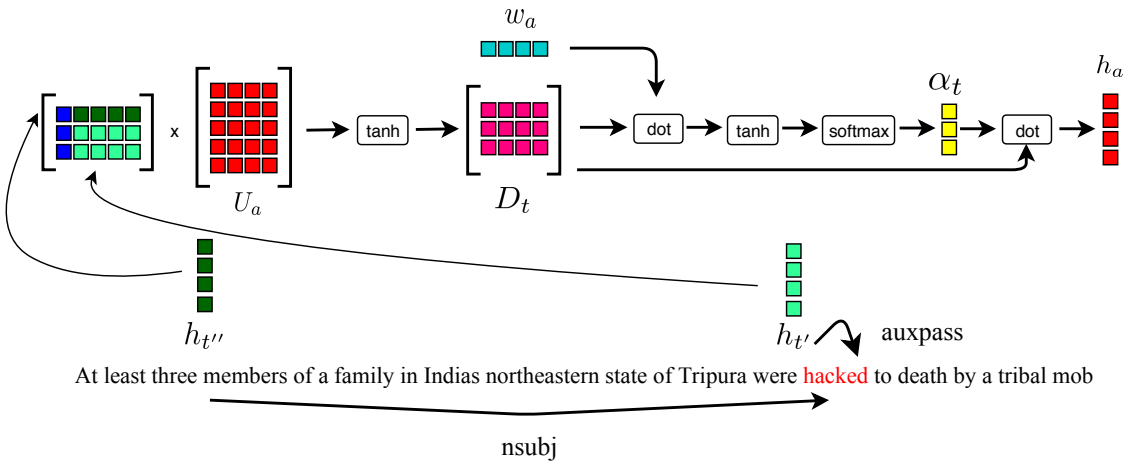

Figure 2.2: The hidden state of "hacked" is a combination of previous output vectors. In this case, three vectors are aggregated with DAG-GRU's attention model. $h_{t''}$, is included in the input for the attention model since it is accessible through the "subj" dependency edge. $h_{t'}$ is included twice because it is connected through a narrative edge and a dependency edge with type "auxpass." The input matrix is non-linearly transformed by U_a and tanh. Next, w_a determines the importance of each vector in D_t. Finally, the attention a_t is produced by tanh followed by softmax then applied to D_t. The subject "members" would be distant under a standard RNN model, however the DAG-GRU model can focus on this important connection via dependency edges and attention.

for a *nsubj* (subject) relation.

An attention mechanism is used to combine the multiple hidden vectors. The matrix D_t is formed at each word x_t by collecting and transforming all the previous hidden vectors coming into node t, one per each edge type e. α gives the attention, a distribution weighting importance over the edges. Finally, the combined hidden vector h_a is created by summing D_t weighted by attention.

$$D_t^T = [\tanh(U_e h_{t'}) | (t, t', e) \in E] \tag{2.5}$$

$$\alpha_t = \text{softmax}(\tanh(D_t w_a)) \tag{2.6}$$

$$h_a = D_t^T \alpha_t \tag{2.7}$$

However, having a set of parameters U_e for each edge type e is over-specific for small datasets. Instead a shared set of parameters U_a is used in conjunction with an edge embedding

v_e.

$$D_t^T = [\ \tanh(U_a h_{t'} \circ v_e) \mid (t, t', e) \in E\ ] \tag{2.8}$$

Here the operator $\circ$ denotes concatenation. The edge type embedding v_e is concatenated with the hidden vector $h_{t'}$ and then transformed by the shared weights U_a. This limits the number of parameters while flexibly weighting the different edge types. The new combined hidden vector h_a is used instead of h_{t-1} in the GRU equations. The model is run forward and backward with the output concatenated, $h_{c,t} = h_{f,t} \circ h_{b,t}$, for a representation that includes the entire sentence's context and dependency relations.

2.3.2 doc2vec

The doc2vec model [42] is an extension of the word2vec [58] model that builds an embedding for a document rather than individual words. Its goal is to create a single vector that represents an entire document, capturing the semantics of the text [43]. For the event detection task, the doc2vec model provides additional global context to individual event predictions.

The doc2vec model has two different formulations: a distributed memory model of paragraph vectors (PV-DM) and a distributed bag of words (DBOW). These formulations are analogous to the continuous bag of words and skip-gram models of word2vec [43, 58]. The use of doc2vec is intended as a reproduction of a recent method [24] which is based on the PV-DM model. However, doc2vec has suffered from some reproduction issues, and it has been shown that the DBOW model is much more consistent in its performance [42]. A pretrained DBOW model [42] was used for this thesis. The DBOW model uses the following objective, which maximizes the probability of the words of the document, represented by word vectors x given the document vector x_d.

$$P(x_1, \ldots, x_n|d) \tag{2.9}$$

The doc2vec model produces a vector that represents an individual document. However, the event detection task requires a decision per individual word in each document. Hence the document vector x_d is combined with the individual word vectors by concatenation i.e. $x_i^* = x_i \circ x_d$.

Long short-term memory (LSTM) recurrent neural network is used to create a contextualized word representation based on the combined word vectors, x_i^* [34]. LSTM uses memory cells to remember key portions of the input and solve the vanishing gradient problem [34]. The model is as follows:

$$f_t = \sigma(W_f x_t^* + U_f h_{t-1} + b_f) \tag{2.10}$$

$$i_t = \sigma(W_i x_t^* + U_i h_{t-1} + b_i) \tag{2.11}$$

$$o_t = \sigma(W_o x_t^* + U_o h_{t-1} + b_o) \tag{2.12}$$

$$\tilde{c_t} = \tanh(W_c x_t^* + U_c h_{t-1} + b_c) \tag{2.13}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c_t} \tag{2.14}$$

$$h_t = o_t \odot \tanh(c_t) \tag{2.15}$$

Like the GRU model, LSTM produces a hidden vector h_t of length d_h for each word represented by a vector x_t^*. The primary diffence between LSTM and GRU is the memory cell c_t LSTM maintains. LSTM uses the cell c_t to form the hidden vector h_t. LSTM combines the previous hidden vector h_{t-1} and the current word vector x_t through a series of gates.

The "forget gate" f_t, "input gate" i_t, "output gate" o_t, and candidate memory cell $\tilde{c_t}$ are all functions of the previous hidden vector h_{t-1} and the current word vector x_t^*. They are parameterized by their respective matrices W and U and the bias vector b. The equation for

the candidate memory cell $\tilde{c}_t$ is show in equation 2.13. Since the forget, input, and output gates use the sigmoid function σ, the value of each of their dimensions' range in value from 0 to 1. This enables the gates to regulate the contributions of other vectors via element-wise product, denoted by $\odot$. The forget and input gates respectively control the how much of the previous memory cell c_{t-1} and candidate memory cell $\tilde{e}_t$ to be included in the updated memory cell c_t as show in equation 2.14. In equation 2.15, the hidden vector h_t is produced by a combination of the output gate o_t and the memory cell c_t.

LSTM is run twice, one starting the beginning of the document and the other starting at the end, creating two sequences: $h_{f,1} \dots h_{f,n}$ and $h_{b,n} \dots h_{b,1}$. The output of the bi-directional LSTM model is the concatenation of both directions vectors: $X' = \Big[h_{f,1} \circ h_{b,1} \dots h_{f,n} \circ h_{b,n}\Big]$. Processing the text in both directions allows for both the left-hand and right-hand contexts to be combined into a single representation.

2.3.3 Transformer

The transformer model uses self-attention to build a representation for each word based on all the other words in a document. Self-attention can capture long-range dependencies between words because the distance between words is not a consideration. There is evidence that self-attention mimics syntactic relationships in a sentence [81].

The transformer model is comprised of four operations. The first is multi-head self-attention. Self-attention transforms its input into three components: a key K, a value V, and a query Q. The transformation is learned by $d_h \times d_h$ linear projections for each component: W_k, W_v, and W_q. If the input is specified by the $n \times d_h$ matrix X, the transformation are as

follows:

$$K = \text{ReLU}(XW_k) \tag{2.16}$$

$$V = \text{ReLU}(XW_v) \tag{2.17}$$

$$Q = \text{ReLU}(XW_q) \tag{2.18}$$

Each projection is of the dimension $n \times d_h$. The projected components Q and K are combined to form attention over the projected component V. The attention mechanism used by the transformer model is scaled dot product. Scaled dot product attention in effect is finding a weight for each pair of words. These weights are used to make a weighted average for each word vector.

$$A = \text{softmax}\left(\frac{QK^T}{\sqrt{d_h}}\right)V \tag{2.19}$$

The softmax function is applied over the first dimension, creating an importance weight per hidden dimension for each input vector. The result, the matrix A, is of size $n \times d_h$ and consists of a composite representation for each word, is based on all the other words vectors in the document.

Next, the transformer applies a residual link and layer normalization [45]. The residual link simply consists of adding the matrix A to the input matrix X as see in equation 2.20. The second half of the transformer model applies a point-wise feed forward network, that is, a pair of convolutional neural networks (CNN) with a window size of 1, as in equations 2.21 and 2.23. Finally, it applies another residual link and layer normalization. The equations are

as follows:

$$L_0 = \text{normalize}(X + A) \tag{2.20}$$

$$F_1 = \text{CNN}(L_0) \tag{2.21}$$

$$F_2 = \text{ReLU}(F_1) \tag{2.22}$$

$$F_3 = \text{CNN}(F_2) \tag{2.23}$$

$$L = \text{normalize}(L_0 + F_3) \tag{2.24}$$

The result L, seen in equation 2.24, is a $n \times d_h$ matrix that represents a single application of transformer's self-attention. However, it has been observed that multiple parallel applications of transformer's self-attention are more effective than a single application. Each parallel application of attention is a "head" and the hidden dimension d_h is split among multiple heads. Each head is has its own set of parameters: W_k, W_v, and W_q. This allows each head to target a different type of interaction. Multi-head attention is formulated as follows:

$$L_m = L_1 \circ L_2 \circ \ldots \circ L_h \tag{2.25}$$

$$X' = L_m W_m \tag{2.26}$$

Where X' is the final representation of the document produced by the transformer model, L_i is the output of a single attention head, W_m is a $d_h \times d_h$ learned linear projection and $\circ$ is the concatenation operator, joining matrices on their second dimension.

2.3.4 The EntNet Model

Despite the effectiveness of RNN models such as LSTM or gated recurrent units (GRU) [19], RNNs are still not effective at long-term memory [33]. A variety of memory networks have been created to address this need. EntNet is a memory network which uses simple RNNs for the memory cells. It is designed to store key parts of the document, key actors or facts, and

use those stored memories later for contextualizing individual predictions per word.

The EntNet model was found to be most effective when combined with a standard RNN. First, GRU was applied to create a locally contextualized representation per word before applying EntNet. For each word, x_t, the GRU model produces a d_h-length hidden vector h_t by combining the word vector with the previous hidden vector h_{t-1}.

The model is applied to both the word sequence and its reverse, creating two hidden vectors per word, $h_{t,l}$ and $h'_{t,r}$. The two vectors are concatenated to produce a single vector representing the word, $h_t = h_{t,l} \circ h_{t,r}$.

The next layer is the EntNet model, which was created to maintain long term memory. EntNet consists of a set of memory blocks and each block is a simple gated RNN. The updated equations for the j^{th} block of memory, e_j are as follows:

$$g_j = \sigma(h_t^T e_j + h_t^T w_j) \tag{2.27}$$

$$\tilde{e}_j = \phi(U_e e_{j-1} + V_e w_j + W_e h_t) \tag{2.28}$$

$$e'_j = e_{j-1} + g_j \odot \tilde{e}_j \tag{2.29}$$

$$e_j = \frac{e'_j}{\|e'_j\|} \tag{2.30}$$

Here U_e, V_e, W_e, and w_j are parameters of the model. The matrices U_e, V_e, and W_e control the update gate, which is a function of the current memory block e_j, the "key" vector w, and the current hidden vector h_t, as shown in equation 2.28. In the equation 2.27, the vector w_j is a "key" vector for the memory block and opens the gate function g_j when the current input, h_t matches the key. Similarly, $h_t^T e_j$ opens the gate when the current input matches the current memory. Together they control the gate g_j which determines how much the memory will be updated given the current input h_t. The gate g_j is a vector where each dimension ranges in values from 0 to 1. This controls how much of the update vector $\tilde{e}_j$ to add to the existing memory block e_{j-1} in the equation 2.29.

In equation 2.28, the vector $\tilde{e}_j$ is the updated to the memory block and it is combined by

the element-wise product $\odot$ with gate function g_j output. However, a simplified version of EntNet was used since it has been shown to be just as effective with fewer parameters [33]. The matrices are set to $U_e = V_e = 0$ and $W_e = I$ and the activation function ϕ is likewise the identity function. Thus the equation 2.28 is replaced with $\tilde{e}_j = h_t$. In either case, the new update vector e'_j is normalized, which allows it to forget part of its previous memory as shown in equation 2.30.

EntNet is used to read the sequence of hidden vectors $h_1 \dots h_n$ in a left-to-right fashion producing m memory cells. These cells are combined with the hidden vector via a bi-linear attention model:

$$a_t = \tanh(EW_a h_t) \tag{2.31}$$

$$\alpha_t = \text{softmax}(a_t) \tag{2.32}$$

$$e^*_t = E^T \alpha \tag{2.33}$$

Here W_a is a $d_h \times d_h$ matrix and are the parameters of the attention mechanism. The matrix E is the $m \times d_h$ matrix of all the memory blocks. This allows for an interaction between the word's hidden vector h_t and each of the memory blocks. The combined memory block representation e^*_t is combined with the hidden vector h_t via concatenation:

$$h'_t = h_t \circ e^*_t \tag{2.34}$$

The combined representation h'_t is used for the final representation of the document:

$$X' = \left[h'_1 \dots h'_n\right]^T \tag{2.35}$$

2.4 Experiments

The ACE2005 dataset is used for evaluation. Each word in each document is marked with one of the thirty-three event types or *Nil* for non-triggers. Several high-performance models were reproduced for comparison. Each is a good faith reproduction of the original with some adjustments to level the playing field.

For word embeddings, Elmo was used to generate a fixed representation for every word in ACE2005 [67]. The three vectors produced by Elmo per word were concatenated together for a single representation. Entity type embeddings were not used for any method. The models were trained to minimize the cross entropy loss with Adam [38] with L2 regularization set at 0.0001. The learning rate was halved every five epochs starting from 0.0005 for a maximum of 30 epochs or until convergence as determined by F1 score on the development set.

The same training method and word embeddings were used across all the methods. Based on preliminary experiments, these settings resulted in better performance than those originally specified.

2.4.1 Models & Baselines

Several baseline models and some variations of the aforementioned models were evaluated. A description of each follows:

- **GRU:** a bi-directional GRU [61] sentence-level model with a hidden dimension of 128. Notably this was not used as a joint model and does not use a memory network. Instead, the final vectors from the forward and backward pass concatenated to each timestep's output for additional context.

- **DAG-GRU:** a bi-directional DAG-GRU sentence-level model described in section 2.3.1 with a hidden dimension of 128.

- **CNN:** a reproduction of a CNN sentence-level model with the number of filters per con-

volution size was reduced to 50 [62].

- **CNN+LSTM:** a reproduction of a sentence-level combination CNN and LSTM model with no modifications [27].
- **DAG-LSTM:** a reproduction of a sentence-level DAG based LSTM model [28], also not used as as joint model.
- **Document GRU:** a bi-directional GRU model, the same as the sentence-level GRU but applied to entire documents.
- **GRU + EntNet:** a document-level model, the combination of EntNet and GRU as described in Section 2.3.4.
- **doc2vec + LSTM:** a document-level model described in Section 2.3.2.
- **doc2vec + LSTM + EntNet:** a document-level model which combines the doc2vec-based model with an EntNet layer after the LSTM layer.
- **Transformer:** a document-level model as described in Section 2.3.3. The first layer is a CNN with window size 1 to "tune" the word vectors and project them into the correct dimension. It uses a hidden dimension of 256 and 16 heads.

Variant A of the DAG-GRU model utilized the attention mechanism, while variant B used averaging, that is $D_t = \frac{1}{|E(t)|} \sum_{(t',e) \in E(t)} \tanh(U_a h_{t'} \circ v_e)$

2.4.2 Effects of Random Initialization

Given that ACE2005 is small as far as neural network models are concerned, the effect of the random initialization of these models needs to be studied. Although some methods include tests of significance, the type of statistical test is often not reported. Simple statistical significance tests, such as the t-test, are not compatible with a single F1 score, instead the average of F1 scores should be tested [85].

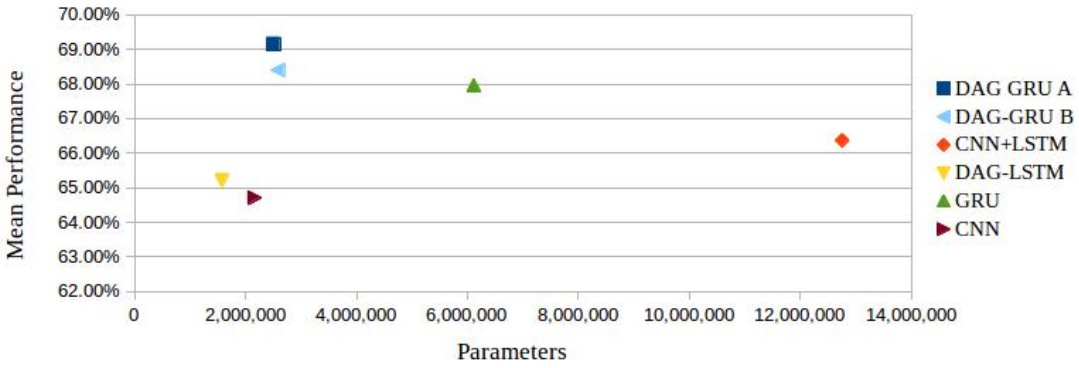

Figure 2.3: A comparison of mean performance versus number of parameters.

Five different systems were reproduced, two others applied, and one newly created and evaluated with different initializations to assess the effect of initialization. The experiments were done on the standard ACE2005 split, and the aggregated results over 20 random seeds are given in Table 2.1. The random initializations of the models had a significant impact on their performance. The variation was large enough that the observed range of the F1 scores overlapped across almost all the models. However the differences in average performances of different methods, except for CNN and DAG-LSTM, were significant at $p < 0.05$ according to the t-test, not controlling for multiple hypotheses.

Both the GRU [61] and CNN [62] models perform well with their best scores being close to the reported values. The CNN+LSTM model's results were significantly lower than the published values, though this method has the highest variation. It is possible that there is some unknown factor such as the preprocessing of the data that significantly impacted the results or that the value is an outlier. Likewise, the DAG-LSTM model underperformed. However, the published results were based on a joint event and argument extraction model and probably benefited from the additional entity and argument information.

DAG-GRU A consistently and significantly outperforms the other sentence-level methods in this comparison. The best observed F1 score, 71.1%, for DAG-GRU is close to the published state-of-the-art scores of DAG-LSTM and GCNN at 71.9% and 71.4% respectively. With additional entity information, GCNN achieves a score of 73.1%. The attention mech-

anism used in DAG-GRU A shows a significant improvement over the averaging method of DAG-GRU B. This indicates that some syntactic links are more useful than others and that the weighting attention applies is necessary to utilize that syntactic information.

Another source of variation was the distributional differences between the development and testing sets. Further, the testing set only includes newswire articles whereas the training and development sets contain informal writing such as web log (WL) documents. The two sets have different proportions of events of each type and each model saw at least a 2% drop in performance between development and test on average. At worst, the DAG-LSTM model's drop was 5.26%. This is a problem for model selection, since the development score is used to choose the best model, hyperparameters, or random initialization. The distributional differences mean that methods which outperform others on the development set do not necessarily perform as well on the test set. For example, DAG-GRU A performs worse than DAG-GRU B on the development set, however it achieves a higher mean score on the testing set.

The results show that there is about a 1% improvement by simply moving from a sentence level model to the document-level model. The additional document-level context is useful without more advanced methods.

Out of the three document-level models evaluated, transformer performed the worst with an average F1 score of 68.71%. It was also quite variable with a standard deviation of 0.88%. Transformer's self attention is not aware of word location, which was likely a problem since the task involves a close reading of the text. Both the positional encoding and embedding additions to transformer were evaluated [81]. However they both hurt performance. Transformer's expressiveness could also be the cause of its poor performance given that the ACE2005 dataset is small.

The doc2vec + LSTM model generally matches its reported performance [24]. The reported F1 score of 70.5%[24] is observed to be within 3 standard deviations of the mean 69.29%. It also had a 0.26% improvement over the document-level GRU model and was the

Model	Dev Mean	Mean	Min	Max	Std. Dev.	Published
Sentence-Level						
DAG-GRU A	70.3%	**69.2% ± 0.42**	67.8%	71.1%	0.91%	-
DAG-GRU B	71.2%	68.4% ± 0.45	67.39%	70.53%	0.96%	-
GRU	70.3%	68.0% ± 0.42	66.4%	69.4%	0.86%	69.3%†
CNN+LSTM	69.6%	66.4% ± 0.62	63.6%	68.1%	1.32%	73.4%
DAG-LSTM	70.5%	65.2% ± 0.44	63.0%	66.8%	0.94%	71.9%†
CNN	68.5%	64.7% ± 0.65	61.6%	67.2%	1.38%	67.6%
Document-Level						
Doc. GRU	69.69%	69.03% ± 0.35	0.75%	67.01%	70.14%	-
GRU + EntNet	67.19%	69.72% ± 0.32	0.68%	68.03%	70.75%	-
d2v + LSTM	71.85%	69.29% ± 0.42	0.90%	67.62%	71.72%	70.5%
d2v + LSTM + EntNet	70.86%	**70.10% ± 0.30**	0.63%	68.70%	71.73%	-
Transformer	68.98%	68.72% ± 0.41	0.88%	67.20%	70.56%	-

Table 2.1: The statistics of the 20 random initializations experiment. † denotes results are from a joint event and argument extraction model.

most variable method with a standard deviation of 0.42%.

The most effective model was EntNet. It yielded a 0.68% statistically significant improvement over the baseline GRU model, according to Welch's two-sample t-test at the 0.05 level. Further, when combined with doc2vec + LSTM, it produced the best overall results with a mean of 70.10%, again, statistically significant improvement of 0.81% over the doc2vec + LSTM model. EntNet was also the most consistent, having the lowest variability of any model. In fact, it made the most variable method into the least, when it was added to doc2vec + LSTM.

The EntNet model is comparable to the state-of-the-art on the random initialization evaluation. The sentence-level best, the DAG-GRU model, achieved an average F1 score of 69.2%, versus 70.10% for EntNet [65]. Again, the improvement is statistically significant at the 0.05% level according to Welch's two-sample t-test.

2.4.3 Bootstrap

One method of model selection over random initializations is to train the model k times and pick the best one based on the development score. Repeating this model selection procedure

Model	Dev Mean	Mean	Std. Dev.
DAG-GRU A	72.0%	69.2% ± 0.04%	0.68%
DAG-GRU B	72.0%	67.9% ± 0.04%	0.60%
GRU	71.5%	68.4% ± 0.05%	0.80%
CNN+LSTM	70.8%	66.8% ± 0.07%	1.08%
DAG-LSTM	70.4%	65.5% ± 0.02%	0.40%
CNN	69.6%	65.4% ± 0.09%	1.49%

Table 2.2: Bootstrap estimates on 1000 samples for each model after model selection based on dev set scores.

many times for each model is prohibitively expensive, so the experiment was approximated by bootstrapping the 20 instantiations per model [26]. For each sentence-level model, 5 development & test score pairs were sampled with replacement from the twenty available pairs. The initialization with the best development score was selected and the corresponding test score was taken. This model selection process of picking the best of 5 random samples was repeated 1000 times and the results are shown in Table 2.2. This process did not substantially increase the average performance beyond the results in Table 2.1, although it did reduce the variance, except for the CNN model. It appears that using the development score for model selection is only marginally helpful.

2.4.4 Randomized Splits

In order to explore the effect of the de facto training/testing split [47] , a randomized cross validation experiment was conducted. From the set of 599 documents in ACE2005, 10 random splits were created maintaining the same 529, 30, 40 document counts per split, of training, development, testing, respectively. This method was used to evaluate the effect of the standard split, since it maintains the same data proportions while only varying the split. The results of the experiment are found in Table 2.3.

The effect of the split is substantial. Almost all models' performance dropped except for DAG-LSTM; however the variance increased across all models. In the worst case, the standard deviation increased threefold from 0.75% to 2.65% for the document-level GRU

Method	Dev Mean	Mean	Min	Max	Std. Dev.
Sentence-Level					
DAG-GRU A	71.4%	**68.4% ± 1.85**	65.7%	74.1%	2.59%
DAG-GRU B	70.9%	**68.4% ± 1.88**	64.19%	73.59	2.63%
DAG-LSTM	68.9%	67.3% ± 1.43	63.5%	70.7%	2.00%
GRU	69.8%	66.6% ± 1.86	62.5%	71.1%	2.60%
CNN+LSTM	69.8%	66.3% ± 2.03	60.1%	70.3%	2.83%
CNN	68.0%	65.4% ± 1.59	60.7%	69.2%	2.22%
Document-Level					
Doc. GRU	70.24%	68.26% ± 1.68	63.42%	71.90%	2.54%
GRU + EntNet	71.36%	69.03% ± 1.00	64.92%	71.73%	2.13%
doc2vec + LSTM	71.10%	69.04% ± 0.76	65.44%	71.23%	1.63%
doc2vec + LSTM + EntNet	71.56%	**69.60% ± 0.67**	66.67%	71.80%	1.44%
Transformer	69.40%	67.12% ± 1.01	64.16%	71.01%	2.15%

Table 2.3: Average results on 10 randomized splits.

model. This aligns with cross domain analysis; some domains, such as WL, are known to be much more difficult than the newswire domain which comprises all of the test data under the standard split [62]. Further, the effect of the difference in splits also negates the benefits of the attention mechanism of DAG-GRU A. This is likely due to the test partitions' inclusion of WL and other kinds of informal writing. The syntactic links are much more likely to be noisy for informal writing, making the syntactic information not as useful and reliable.

Moving to a document-level model helps on average about 1.64%, EntNet adds about 0.7% on average. EntNet's improvement is consistent across evaluations. Due to the variablity of this experiment, the same randomized splits were used across each model. This enables the use of a paired t-test, which found EntNet's improvement over doc2vec + LSTM to be significant at the 0.05 level.

Further, the EntNet models showed a large drop in standard deviation when combined with either model. The consistency of results on event detection has seen limited research, though is important with regards to reproducibility and practical application. The doc2vec + LSTM model performed the most consistently between evaluations, only losing 0.25% on average between experiments. Overall, doc2vec + LSTM + EntNet performed the best. However, the randomization of the partitions did impact it significantly.

Statistic	**EntNet**	**Trans.**	**EntNet vs. Trans.**
Agree & correct	275	284	276
Agree & wrong	199	197	192
Disagree (false negative)	29	35	37
Disagree (false positive)	36	47	49
Disagreement to Agreement Ratio	13.71%	17.05%	18.38%

Table 2.4: Prediction and error correlation counts for the doc2vec + LSTM + EntNet and Transformer models. The "EntNet" and "Trans." columns compare two models within the same time but trained with different random initializations. Both the cases of disagreement involve one model making a correct prediction and the other making the stated error. The ratio is calculated by dividing the sum of the disagreement counts by the sum of agreement counts.

All these sources of variation are greater than most advances in event detection, so quantifying and reporting this variation is essential when assessing model performance. Further, understanding this variation is important for reproducibility and is necessary for making any valid claims about a model's relative effectiveness.

2.4.5 Error Correlations

In addition to the statistical comparison of models, an analysis is performed on their predictions and errors. The premise of the different models evaluated is that their different network architectures allow them to capture different types of information. Here the models considered are the farthest from each other in terms of design: EntNet and transformer. EntNet is a memory network with memory cells consisting of simple RNNs, and in this work it is also used in conjunction with GRU and LSTM. However, the transformer model eschews the RNN structure completely, relying on self-attention instead to capture the context of the document.

The goal is to determine whether EntNet and transformer make similar decisions in spite of their vastly different architectures. If the models are capturing different kinds of information then they should be making different types of predictions. That is, they should both have their own set of correct predictions, and the intersection of their sets should be minimal.

However, to determine how the models correlate with each other, there needs to be a baseline for comparison. Hence, two instantiations of each model are considered, all of them at similar levels of accuracy 70.1%, taken from the random initialization experiment. Next, for each pair, four different statistics were collected: the count of events they both correctly predict, the count of non-events or mistyped events they both incorrectly predict, the count of events one predicts correctly but the other counts as *Nil*, and the count of non-events that one incorrectly predicts as an event. Note that correctly predicting *Nil*, i.e. a true negative, is not included in the counts, as in F1. A summary of the results is shown in Table 2.4.

The pairs within each model type establish the baseline, that is, it determines the level of variation in predictions for the type. Given the difficulty of the task, it is expected that the variation in the model initialization will result in variation of the model decisions. Hence the variation is compared within a model type versus the variation across model types. If the two types of models are actually capturing different information then they should disagree with each other much more than they disagree within type.

By dividing the disagreement counts by the agreement counts, a ratio is constructed per pair. It is observed that for EntNet the ratio is 13.71%, for transformer it is 17.05% and for EntNet versus transformer it is 18.38%. The lowest value the disagreement ratio between EntNet versus transformer can realistically take is 17.05%, since that is the higher within type ratio. However, the ratio is only 1.33% higher, indicating the models mostly agree, almost as often as within the transformer type. This gives evidence that these models are effectively making the same decisions at a given level of accuracy despite the fact they are quite different in structure. There is no evidence that the differences in model architecture resulted in different information or relationships being represented from a predictive standpoint.

2.5 Conclusions

Altogether, five models were reproduced, two models were applied, and a new model was created for event detection. The DAG-GRU model was introduced to use syntactic information through an attention mechanism. It was shown to be competitive with the state-of-the-art at the sentence-level. Further, document-level models were shown to be in general higher in performance than sentence-level models. In particular, EntNet demonstrated the highest performance on both the random initializations and training-test split evaluations on ACE2005. Further, these experiments in general show that there are several significant sources of variation which had not been previously studied and quantified. At a minimum, it suggests that the community should move away from evaluations based on single random initializations and single training-test splits. Finally, it was demonstrated that despite the major architectural differences, transformer and Entnet were qualitatively similar except for a modest but significant difference in accuracy.

Chapter 3: Relation Extraction

3.1 Introduction

Complex AI problems such as natural language understanding consist of a variety of tasks from low level tasks such as part of speech tagging and parsing to more higher level tasks such as event detection, event sequence extraction, and question answering. The current best approaches are based on neural networks and are typically focused on optimizing the performance on one task at a time. However, it is not clear how to integrate multiple tasks in a unified framework that provides a clean separation between the tasks, while also taking advantage of their mutual relationships to constrain learning. Part of the problem is that the primitive elements in each task are different and are created by previous tasks. For example, while a document can be described as a sequence of words at the lowest level, it consists of mentions of entities at the next level with syntactic relationships between them. At higher level of processing, the mentions are clustered into entities with a variety of semantic relationships between them.

In this chapter, a flexible architecture called Graph Transforming Neural Network (GTNN) is proposed. It allows the integration of multiple tasks into a single architecture, taking into account the different kinds of information represented and manipulated at different levels of processing. The architecture consists of multiple layers, where each layer is represented as a graph. The nodes in the graph at a higher layer are composed of nodes at a lower layer. Edges at a higher layer are predicted based on the nodes and edges at the lower layer. The different layers can be trained simultaneously if the training data is available. If not some of the decisions can be hard-coded or heuristically guided. Importantly the architecture supports modularity and creation of new objects from objects at lower levels, a requirement for AI to robustly address the full problem of language understanding.

The GTNN framework is illustrated in the context of event sequence extraction which has several subtasks. Events are key for understanding narratives and news stories. Further, placing events into a temporal sequence is a means of representing narratives. Hence, identifying the temporal and compositional relationships between events is a necessary and challenging aspect of representing and understanding narratives. In this context, the proposed framework starts with an input graph that is a linear chain of tokens or words and applies a series of three transformations to it. The first transformation converts it into a graph of entity and event mentions which are connected by co-reference links. The second transforms this graph into an event-argument graph, where event instances are connected to their argument instances. Finally, the third transformation creates event instances (which aggregate events and their arguments in the second layer) and links them with containment and temporal relationships.

GTNN is a flexible framework that can be combined with a variety of graph-based neural network architectures to create more specialized networks. The second main contribution is a specific instantiation of GTNN to event sequence extraction by combining attention and max-pooling. This model is motivated by the intuition that stories are centered around a few entities, i.e. people, places, and things. Together, the attention mechanism and the max-pooling allow the model to ignore the irrelevant entities and focus on the key entities.

A robust empirical evaluation of the model is done on the TAC KBP 2017 dataset on event sequence extraction and is compared to a baseline method and several variations of the model. The results based on 10-fold cross validation show that the proposed method significantly outperforms the baseline.

3.2 Event Sequence Extraction

At the highest level, the event sequence extraction task is to determine if there is an “After” or “Parent-child” relationship between a given pair of events or not. The relationships are

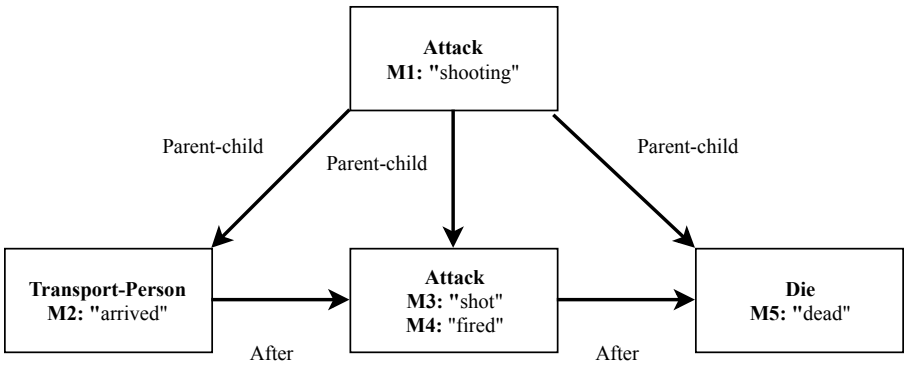

Figure 3.1: An example of the graph for the text, "The shooting occurred late Tuesday afternoon. The assailant arrived at the marketplace shot a shopkeeper and then fired on a nearby crowd, leaving four dead". A single event mention, "shooting," is a summary, describing a sequence of events and is indicated as such by "Parent-child" relations. In this case two mentions make up the "Attack" event i.e. "shot" and "fired." Note, the summary event "shooting" is not necessary for the other events to have an "After" relationship.

only defined to hold if the events in question are related in some way, i.e. parts of the same "script." A script is a stereotypical sequence of events and a story is an instance of a script [74]. The "After" relation temporally orders events that occur within the same narrative. However, the task is not to temporally order all the events in a document. Only those events which are part of the same story have "After" relations defined even if the events can be placed in chronological order. Consider the following example of "After" relations.

> One person **died** from **injuries** after Croat and Muslim fans **clashed** . . . there were several others **injured**, while fans **demolished** shops . . .

According to the event sequence task, there are two separate stories. The first involves a single person dying from fans fighting i.e. "clashed" ⇒ "injuries" ⇒ "died" with each arrow indicating "After." The other story is about fans destroying shops and hurting people as a result i.e. "demolished" ⇒ "injured." Though both stories involve the same generic group of fans and are related to the same circumstances, they are considered separate. This is due to the difference in the victims: a single person in the first versus a small group in the second. Since the participants are different, yet are related in content, the two stories are arguably different instantiations of the same script.

The second relation defined by the task is "Parent-child." This relation indicates that a single event is a summarization or composition of a sequence of events. The "Parent-child" relation is illustrated by the following example.

> Ali Ibrahim Al-Sudani was **detained** and **sent back** to Egypt under an emergency **deportation** order ...

Here "detained" and "sent back" are event mentions of "Arrest-Jail" and "Transport-Person" types respectively. The event mention "deportation" is also of type "Transport-Person," and is also the "Parent" of both "detained" and "sent back." It is clear here that a deportation logically implies both an arrest and a transportation of a person. In this case, the term "deportation" serves as a summary or as a label for a composition of events.

The relations are defined to hold at the event level, that is, for a collection of event mentions associated through co-reference. Event co-reference relations, along with event mention annotations are given as input for event sequence extraction. An example of the graph structure of the task is given in Figure 3.1.

This task is surprisingly difficult for several reasons. The data is usually sparse due to the cost of annotation. The classes are unbalanced in the sense that the vast majority of event pairs (about 99%) are unrelated to each other. Further, the relations are nuanced in that judging whether two events are part of the same story requires a careful reading of the text with particular consideration for the entities involved. However, entities, i.e. people, places, things, etc. are not a part of the input for the task despite their importance. Finally, there is the issue of representing the elements of the task, i.e. the events, their mentions, and their relationship with each other. The state of the art methods on this task have only been able to achieve 6-7% F1.

3.3 Related Work

Prior work on event sequence extraction has been limited. The task was introduced for a TAC KBP pilot study and benchmark competition in 2017. The only published work framed the problem as a multi-label classification problem on pairs of event mentions represented by a bi-directional GRU [19] model [76]. Though the relations were originally defined for events, they were later expanded to event mentions. That is, if a relation held between events, it was then defined to hold between all pairs of corresponding mentions. Each relation type, "After" and "Parent-child" was inverted into "Before" and "Child-Parent" to avoid comparing both orderings of event mentions. The pairs were ordered according to their narrative ordering. Additionally, the type "Nil" was added for non-existent relations, making the total number of relation types five.

Each pair was represented by a feature vector comprised of three parts: the GRU hidden vectors for the left and right mentions, a path embedding, and additional knowledge base features. First, each word was represented with a concatenation of word and part-of-speech embeddings. A bi-directional GRU model was then used to create a contextualized representation for all the words in each document. The vectors corresponding to each event mention were selected as the first part of the feature vector.

The vector of the word previous to the second mention vector was used as a path representation. That is, if the mention vectors are given as x_t and $x_{t'}$ and $t < t'$, then the path representation is $x_{t'-1}$. Finally, features based on event type, lemmas, realis (hypothetical event versus actual occurrence), sentence distance, and semantic relations from VerbOcean [15] and ConceptNet [77] were also included. Each pair was classified with Multi-layer Perceptron (MLP) making use of dropout [72, 78].

A related task is temporal relation extraction. The aim of this task is to predict the temporal ordering between pairs of events or between events and particular times. There is a long history of research on this topic, with many contributions resulting from the SemEval and TempEval competitions [82, 83, 80, 7, 8, 55].

Recently, neural network methods have risen to the top, replacing hand-engineered feature-based systems. Multiple methods of applying bi-directional LSTMs [34] to dependency parse paths have generated rich representations for temporal relation classification [16, 14]. Further, convolutional neural networks (CNN) [44] have been shown to out-perform bi-directional LSTM models on a clinical temporal relation dataset [23].

Another related task is event co-reference resolution, which has the goal of determining if pairs of event mentions refer to the same event. Event trigger detection and event co-reference were jointly modeled by structured perceptron [3]. More closely related, a max-ent model was applied for event co-reference with the "Parent-child" relation to identify the structure of events and mentions [2]. Recently, an integer linear programming joint model of event co-reference, "Parent-child" relations, and document topics demonstrated that solving the related tasks improves event co-reference accuracy [17].

Neural network models for graph-based relational reasoning have shown success. Rel-Net is a network architecture for modeling relationships between objects in images and text [73]. It combines a memory network structure with a MLP to represent complex relationships and achieves state-of-the-art results on challenging visual and text-based question answering datasets.

Graph convolutional neural networks (GCNN) likewise utilize graph structures to model complex tasks [37]. GCNNs have been successful at the event detection task, effectively leveraging dependency parse information [64]. They have been shown to model sentences successfully for semantic role labeling [57]. Finally, GCNNs have been used in a semi-supervised learning setting on citation network and knowledge graph datasets [39]. Though not exactly a GCNN, dependency CNNs also use the graph structure of a sentence's dependency parse to achieve high accuracy on question answering and sentiment analysis datasets [54].

Finally, attention models have been successful on a variety of NLP tasks. Attention models have made significant improvements in machine translation [53] and document classification [87]. They have also been applied to the event detection task [51, 49]. The DAG-GRU

model further applies attention to the graphical structure of a dependency parse for event detection [65].

3.4 Graph Transforming Neural Networks

While event mentions and co-references are typically given as inputs for the task of event sequence extraction, I hypothesize that entities and their roles in events play an important role. The task has many elements to consider: the document text, event mentions, entity mentions, co-reference relations, events, entities, argument relations, and instantiated events. These elements can be represented in a graph structure. However, they are at different levels of abstraction. For example, events are groups of event mentions connected by co-reference relations. Creating a single graph out of all the elements is complex and computationally cumbersome.

However, there are often simple relationships between successive levels of abstraction. The entities at a given level are formed from the entities at lower level through some kind of composition. For example, an event is made of a set of event mentions and a set of mentions represent an entity. Therefore, a graph can be defined for each level of abstraction as well as a transformation between successive levels. Hence, this architecture is framed as a series of graph transformations.

More formally, a graph is defined as a set of nodes and edges, $G = (V, E)$. A graph transformation $H_l(G_l)$ takes as input a graph G_l and produces a graph G_{l+1}, i.e. $G_{l+1} = H_l(G_l)$. H_l is decomposed into two functions f and g, which transform nodes and edges respectively:

$$H_l(G_l) = \big(V_{l+1}, g(V_{l+1}) | V_{l+1} = f(G_l)\big) \tag{3.1}$$

The function f aggregates nodes in G_l to produce a new, smaller set of nodes V_{l+1}. Specifically, $f(G_l) = \{r(s, G) | s \in S_l\}$ where r is a function that generates a representation

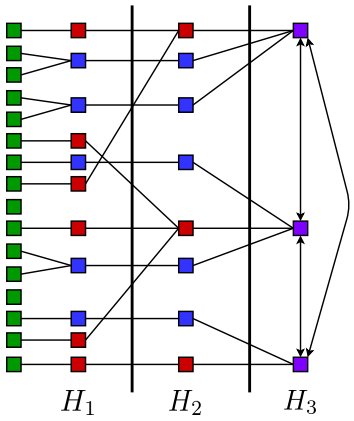

Figure 3.2: An overview of GTNN. H_1 transforms the word representations (green) into event (red) and entity (blue) mention nodes. Next, H_2, utilizing event co-reference, creates event and entity nodes. Finally, instantiated event nodes are created in H_3 by heuristically linking events and entities that co-occur in the same sentence. All pairwise combinations of instantiated events are evaluated for a possible "Parent-child" or "After" relation.

for each subset of the graph. Some possible models that can be used for r are GCNN [37, 39, 57], Rel-Net [73], DCNN [54], or even an RNN [19] with pooling. In general, f needs to summarize a graph as a set of vectors which comprise the nodes in the new graph.

These nodes are linked with edges produced by g. g needs to evaluate pairs of nodes, and determine if an edge should exist between them. For example, a linear model or multi-layer perceptron (MLP) operating on pairs of nodes could predict edges. Alternatively, prior knowledge or heuristic-based functions can be substituted. This framework allows for the neural network to mirror the natural decomposition of a problem, and exploit it for efficient learning and inference. The framework is illustrated in the context of event sequence and entity relation extraction in the remained of the chapter.

3.5 First Transformation

Three transformations are defined for the event sequence extraction as shown in Figure 3.2. The first transformation converts a graph representing the document into a graph of event and entity mentions connected by edges indicating co-reference. Figure 3.3 contains an illustration of the first layer. The initial graph G_0, is comprised of the document text.

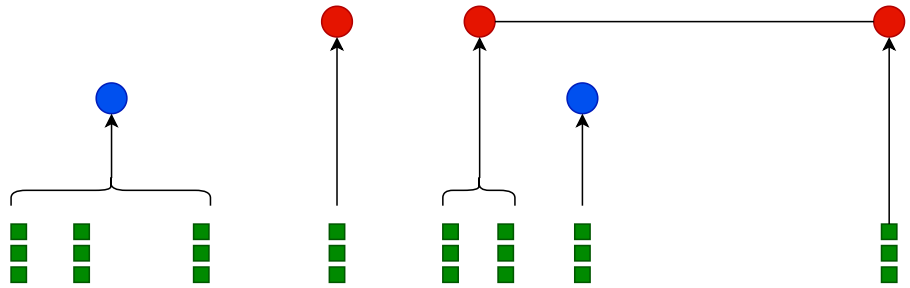

Figure 3.3: An example of the first layer of GTNN for event sequence extraction. A GRU model produces a vector per word. Spans of words are averaged into a single vectors represented by the blue and red nodes. Since "sent back" and "deportation" refer to the same event they are linked an edge. The spans for event mentions are part of the dataset's annotation, while the entity spans are predicted.

Each token in the document is a node and the narrative ordering of the text forms the edges. The representation function r_1 consists of a word embedding layer followed by GRU with average pooling [19]. Average pooling is applied to the subsets in S_1, which are event and entity mention spans. The linking function g_1 simply connects nodes based on the given co-reference relations. For entities, rather than attempting to solve the challenging problem of entity co-reference, g_1 creates a fixed number of entities per type and links each mention to each entity of the same type. This avoids making co-reference decisions but allows for the next transformation layer to create entity representations. Altogether, this transformation H_1, applies word embeddings, GRU, and average pooling defined by mention spans. The result is a graph consisting of event and entity mentions with the event mentions connected according to co-reference relations and the entity mentions grouped according to entity type.

3.6 Second Transformation

The second layer transforms the mention graph into an event argument graph. That is, a single representation is created for each event and each entity out of their corresponding mentions. In the mention graph, an event or entity is a sub-graph of mentions. These

subgraphs constitute the subsets in S_2. The representation function r_2 could be any graph neural network, but for this work four kinds of neural networks were considered. GTNN is modular, allowing for any one of the following graph neural networks to be used as a representation function $r_2(s_2, G)$:

- GCNN applies a transformation to each mention $m \in s_2$, sums the result, and applies a transformation. It is denoted with the following: $r_{2,\text{gcnn}}(s_2, G) = \text{ReLU}\Big(\sum_{m \in s_2} W_r m\Big)$
- Rel-Net applies a multi-layer perceptron network to pairs of mention vectors. The equation is: $r_{2,\text{rel-net}}(s_2, G) = \sum_{m,m' \in s_2} \text{MLP}\Big(m \circ m'\Big)$
- Max-pooling simply takes largest value of each dimension in the mention vectors. The equation is: $r_{2,\text{max-pool}}(s, G) = \text{row-max}\Big(\big[m | m \in s\big]\Big)$
- Attention effectively makes a weighted average of the mention vectors. The equations are:

$$M^T = \big[m | m \in s\big] \tag{3.2}$$

$$\alpha = \text{softmax}(\tanh(M w_r)) \tag{3.3}$$

$$r_{2,\text{attention}}(s, G) = M^T \alpha \tag{3.4}$$

Each of the representation functions contains trainable parameters except for max-pooling. The mention vector m is composed of the output of the previous layer h_1 and an embedded mention type e_t, i.e. $m = h_1 \circ e_t$. Here the $\circ$ is the concatenation operator. The linking function g_2 is heuristically based and adds an argument edge between any event and entity if they contain mentions that co-occur in a sentence. In summary, H_2 builds event and entity nodes by applying a graph neural network to aggregate mentions into a single canonical vector. These event and entities are then linked by textual co-occurrence. The result is a graph consisting of events linked with possible arguments.

Method	Dev Avg.	Test Avg.	Std. Dev.	Min	Max	Avg. Δ
Baseline	7.69%	6.36%	2.38%	2.42%	9.21%	-
Baseline+GCNN	6.20%	4.44%	3.08%	0.00%	8.86%	-1.92% ± 2.43%
Baseline+Rel-Net	4.99%	1.05%	1.01%	0.00%	2.89%	-5.31% ± 1.65%
Baseline+Attention	8.21%	4.14%	2.31%	0.00%	7.57%	-2.22% ± 1.87%
GCNN	8.69%	6.18%	2.72%	3.82%	12.14%	-0.18% ± 1.45%
GCNN+max	7.30%	5.98%	3.48%	2.78%	11.76%	-0.37% ± 1.94%
GCNN+max+ents	7.67%	6.53%	2.95%	2.58%	11.49%	0.17% ± 1.53%
Rel-Net	8.35%	4.64%	2.87%	0.0%	10.86%	-1.71% ± 2.46%
Rel-Net+max	7.97%	6.32%	3.44%	1.09%	12.41%	-0.04% ± 2.26%
Rel-Net+max+ents	7.13%	5.42%	3.11%	0.00%	10.04%	-0.94% ± 1.84%
Attention	6.88%	5.36%	1.74%	2.58%	7.72%	-1.00% ± 0.89%
Attention+max	8.59%	7.83%	2.58%	3.92%	11.71%	1.47% ± 1.45%
Attention+max+ents	8.36%	**8.07%**	3.21%	2.83%	13.55%	1.71% ± 1.39%

Table 3.1: F1 scores on 10 fold cross validation results on TAC KBP 2017. The "Avg. Δ" is the average improvement over the baseline.

3.7 Third Transformation

Finally, the third layer transforms the event-argument graph into a graph of instantiated events with edges indicating either a "Parent-child" or "After" relationship, the target of the event sequence extraction task. The inputted event-argument graph consists of sub-graphs, each containing a single event and associated arguments. For this layer, each set $s_3 \in S_3$ is a set edges between a single event and all its arguments i.e. $e, a \in s_3$. The graph neural networks used for r_3 are similar to those used for r_2 except that they are event-centric. The networks are as follows:

- GCNN applied to the event-argument graph concatenates the event representation e with the argument representation a, effectively aggregating the edge relationships: $r_{3,\text{gcnn}}(s_3, G) = \text{ReLU}\Big(\sum_{e,a \in s_3} W_r(e \circ a)\Big)$
- Rel-Net likewise builds a representation of each event-argument link with a multi-layer perceptron function: $r_{3,\text{rel-net}}(s_3, G) = \sum_{e,a' \in s_3} \text{MLP}\big(e \circ a'\big)$
- Attention uses a bilinear projection to combine the event representation with all the

associated arguments before performing a weighted average:

$$M^T = \left[a|e, a \in s_3\right] \tag{3.5}$$

$$\alpha = \text{softmax}(\tanh(MW_a e)) \tag{3.6}$$

$$r_{3,\text{attention}}(s_3, G) = W_r\Big((M^T\alpha) \circ e\Big) \tag{3.7}$$

MLP denotes a multi-layer perceptron network. GCNN and Rel-Net here are modified slightly to model the interaction between an event and each of its arguments. The attention network is significantly different than the version for r_2. The importance of each argument is determined by the bilinear function $MW_a e$ rather than by a single weight vector. This also models the interaction of event and argument and weights each argument accordingly. Finally, it produces a linear projection of both the event and attention weighted argument vectors. The output of r_3 is a vector that represents an instantiated event.

The linking function g_3 predicts edges between pairs of instantiated events. This is framed a multi-class classification problem, where the class labels are "Parent-child," "After," or "Nil." Dropout [78] is applied before a linear classifier where softmax is used to make the prediction. The final graph consists of nodes representing instantiated events with relationships forming edges between them.

3.8 Experimental Results

The TAC KBP 2017 event sequence extraction dataset is used for evaluation. It consists of 158 documents, taken from newswire, weblog, and discussion forum sources. Additionally, a entity tagging system was trained on ACE2005 to predict entity mentions. It consists of a three layer GRU model with softmax classifier and was trained for BIO tagging [70] on the train-dev-test split of ACE2005 [84], achieving an F1 Score of 82% on test. This system was used to predict entity mentions according to the ACE2005 ontology on the TAC dataset.

The data was randomly split into 10 folds, each consisting of 88, 30, and 40 documents for training, development, and test respectively. Splitting this way maintains consistent amounts of training and testing data, making comparisons across the splits meaningful. Each method was evaluated on all the folds and measured against the baseline performance for each fold. The difference was evaluated by a paired t-test. Performance was measured with F1, the harmonic mean between precision and recall. The average F1 difference between the baseline and the method was used as the basis for the evaluation of each method.

3.9 Baseline

The baseline system consists of a straight-forward GRU-based model similar to that used in the TAC 2017 competition [76]. It was recreated since it is the only prior work and so the model can be evaluated fairly in the same experimental setup. The baseline can be viewed as a simplified version of a graph transforming neural network, utilizing H_1 and H_2 without any entities or entity mentions, with $r_2 = r_{2,\text{max-pool}}$.

The model consists of four layers. The first is the word embedding layer, followed by a bi-directional GRU. The result is a contextualized representation for each word in the document. The third layer applies a max-pooling for each event over all the event's mentions. Finally, a linear softmax classifier predicts the relation type between each pair of events. Unlike the TAC 2017 model, the path embedding, part-of-speech embedding, and additional features were removed. With the addition of the max-pooling layer over entity mentions, the same information is available to all the models, making the comparison as fair as possible.

Some extensions of the baseline were also considered. The max-pooling layer was replaced with either GCNN, Rel-Net, or the attention model. Baseline+GCNN, Baseline+Rel-Net, Baseline+Attention use $r_{2,\text{gcnn}}$, $r_{2,\text{rel-net}}$, and $r_{2,\text{attention}}$ for r_2 respectively as shown in Table 3.2. These models made use of the event structure without any entity or entity mention information.

Model	$r_{2,\text{event}}$	$r_{2,\text{entity}}$	r_3
Baseline	$r_{2,\text{max-pool}}$	-	-
Baseline+GCNN	$r_{2,\text{gcnn}}$	-	-
Baseline+Rel-Net	$r_{2,\text{rel-net}}$	-	-
Baseline+Attention	$r_{2,\text{attention}}$	-	-
GCNN	$r_{2,\text{gcnn}}$	-	$r_{3,\text{gcnn}}$
GCNN+max	$r_{2,\text{max-pool}}$	-	$r_{3,\text{gcnn}}$
GCNN+max+ents	$r_{2,\text{max-pool}}$	$r_{2,\text{gcnn}}$	$r_{3,\text{gcnn}}$
Rel-Net	$r_{2,\text{rel-net}}$	-	$r_{3,\text{rel-net}}$
Rel-Net+max	$r_{2,\text{max-pool}}$	-	$r_{3,\text{rel-net}}$
Rel-Net+max+ents	$r_{2,\text{max-pool}}$	$r_{2,\text{rel-net}}$	$r_{3,\text{rel-net}}$
Attention	$r_{2,\text{attention}}$	-	$r_{3,\text{attention}}$
Attention+max	$r_{2,\text{max-pool}}$	-	$r_{3,\text{attention}}$
Attention+max+ents	$r_{2,\text{max-pool}}$	$r_{2,\text{attention}}$	$r_{3,\text{attention}}$

Table 3.2: Baseline & GTNN model variations

3.10 GTNN Models

Multiple instantiations of the GTNN framework were evaluated. All variations of GTNN use the same H_1: word embeddings followed by GRU with average pooling over mentions. The word embeddings for all the systems were initialized with a pre-trained 300 dimension word2vec model [58] with the parameters fixed during training. Three types of graph neural networks were considered, each with three variations, detailed in Table 3.2.

Each model with a "+max" variation is a direct extension of the baseline. These models represent each event with a max-pool of its mentions. However, in the second transformation, entity mentions are not grouped: each one is left unaltered and passed to the third transformation. The final transformation uses the nominal network to combine entity mentions and events into instantiated events, e.g. "GCNN+max" uses $r_{3,\text{gcnn}}$.

Each nominal variation is an extension of the corresponding baseline, e.g. "Rel-Net" extends "Baseline+Rel-Net." Unlike the "+max" variation, the nominal model uses that representation function to aggregate event mentions in order to represent events, e.g. "Rel-Net" uses $r_{2,\text{event}} = r_{2,\text{rel-net}}$. Their final transformation is the same as the "+max" variation.

Finally, the "+max+ents" variation is an extension of the "+max" variation. The primary difference is in the second transformation, using a representation function for events and a

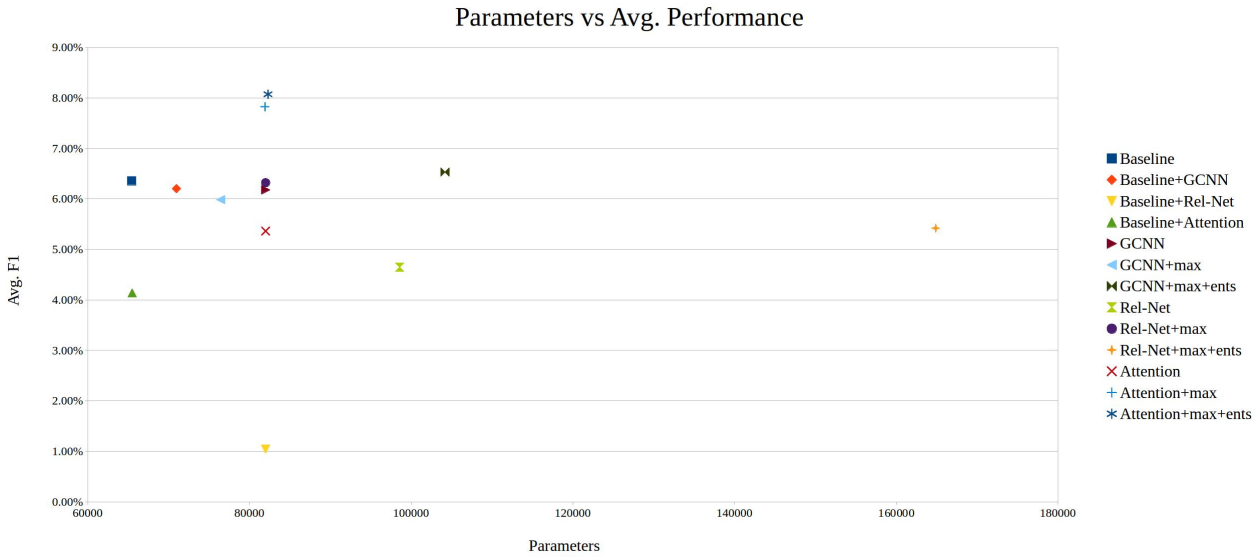

Figure 3.4: A comparison of each model's parameter count versus the its average performance.

set of representation functions for entities. Instead of solving the entity co-reference problem, for each entity type, a fixed number of fully connected entities are created. A corresponding number of representation functions are used to model the entities. This allows the models to learn entity representations useful to the task by consolidating entity mentions. The number of entities is set to be 5 via hyper-parameter tuning on the dev sets.

The hyper-parameters and settings are the same across all the models and the baseline. The expanded relation type strategy used in the TAC competition system is followed [76]. The models were trained with the Adam algorithm [38] with a learning rate of 0.001 which is halved every 75 epochs for a maximum of 200 epochs. The hidden dimension of the GRU in H_1 for a single direction is 32, and the mention type embedding size is 10. Hence the dimension of the subsequent layer is 74. The dropout rate is 0.5. The models minimizes cross-class entropy loss.

3.11 Results

The task is challenging, and the baseline method was formidable. A summary of the results of the experiments is shown in Table 3.1. The baseline method, i.e. Baseline, beat all its extensions, Baseline+GCNN, Baseline+Rel-Net, Baseline+Attention, by a large margin. The representations of events other than max-pool simply were not helpful.

Only the attention family of models, Attention, Attention+max, Attention+max+ents, displayed significant improvements over the baseline. Both the variation with max-pooling of events, Attention+max, and the variation with entities, Attention+max+ents, showed statistically significant improvements over the baseline with average scores of 7.83% and 8.07% respectively. These models were the highest scoring across all models and variations and were able to effectively utilize the entity and entity mention information. The combination of entity information with attention could indicate that entity information is important however, only some entities are relevant. This matches the intuition that stories are written around a handful of important people, places, and things. All the entity mentions were predicted by a separate system trained on ACE2005 and thus are noisy. The attention mechanism perhaps allowed for filtering of irrelevant or noisy entities or entity mentions. Further, the attention models have relatively few parameters, possibly helping these models avoid overfitting.

The GCNN based models, GCNN, GCNN+max, GCNN+max+ents, were statistically indistinguishable from the baseline and generally worse on average despite using more parameters. Only the GCNN with entities, GCNN+max+ents, was better on average than the baseline, with a small 0.17% improvement; however the difference is not statistically significant. GCNN's relatively simple architecture did not effectively use its parameters. When compared to models with a similar number of parameters, its performance lagged as seen in Table 3.1.

The Rel-Net based models, Rel-Net, Rel-Net+max, Rel-Net+max+ents were not successful, scoring the lowest of the methods evaluated. Each of these variations was worse than the baseline. Rel-Net's network structure is the most complex of the models evaluated, and it

was likely to their detriment. In particular, using Rel-Net to model pairs of event mentions, i.e. the "Rel-Net" and "Baseline+Rel-Net" models, were some of the worst with average scores of 4.64% and 1.05% respectively. This demonstrates that modeling the interaction of event mention pairs is likely not helpful. As seen in Figure 3.4, the relatively high number of parameters used by Rel-Net likely caused overfitting.

One trend across all the families of models was the importance of max-pooling for event mentions. In almost every case, the models that included max-pooling of event mentions out-performed their counterparts. In particular, all the extensions of the baseline were significantly worse on average. This could indicate that for each event, there is a canonical mention that best represents it.

Additionally, the three base models, GCNN, Rel-Net, and attention, all have a common structure, a summation over all given mentions. Each model is in its essence a different strategy for weighting each mention before aggregating them with a sum. If each event does have a single or a small number of mentions that best represent it, these models may be diluting them by summing over all mentions.

Another positive effect was that of entities. The representation of entities was simplistic, small fixed number of entities per type were created. Each entity was connected to all the mentions of its type. Despite the limited nature of their representation, this variation of each model family was the best on average except for Rel-Net+max+ents. It used far more parameters than any other model or variation and very likely overfit the data. The success of the other two "+ents" variations could indicate that aggregating entity mentions is important for utilizing their information. Further, it may provide for the ability to filter noisy or otherwise unimportant entity mentions.

3.11.1 Entity Relation Results

In addition to the event sequence task, the GTNN framework was evaluated on the ACE2005 entity relation extraction task. Pairs of entity mentions are defined to have one of seven possible relations or *Nil* if no relations exists.

For this task, an instantation of the GTNN framework similar to the Attention+max+ents model was used on the event sequence task. However in the event sequence extraction task, entity mentions were included as additional predicted information. For entity relation extraction, the situation is reversed with event mentions being included as additional information. Since, the event mentions as well as the event argument links are included in the ACE2005 dataset. The links and mentions were simply included from the ACE2005 annotations, rather than heuristically created or predicted with a separate model.

For comparison, a high performance baseline model was reproduced [18]. The baseline uses bi-directional LSTM with a pairwise attention model over entity mentions. This model was chosen as a baseline since its components are similar though it does not fit within the GTNN framework. Both models in this evaluation use GloVe word embeddings [66]. GTNN produced an F1 score of 46.98% while the baseline was evaluated at 46.83%. These values are likely equivalent since they are close and this thesis has already established the variability of neural networks. It does show that GTNN is likely competitive with a current, high performance entity relation extraction system.

However, the reproduction of the baseline was far below the reported value of 61.4% [18]. There are a few possible explanations for this. First, the training-testing partition used for this evaluation was different, so the comparison is not completely fair. Also, the word embeddings employed differ from the reported source as well, which can significantly impact performance [67, 66]. Naturally, it is possible there is an oversight in the implementation or there is some other significant factor not stated or implied. To reiterate, this experiment does show that the two methods are roughly equivalent in a fair evaluation setting.

3.12 Conclusions and Future Work

The graph transforming neural network framework was introduced and many derived models was evaluated on the task of event sequence extraction. While the task is difficult even for sophisticated neural network models, two models managed to outperform a competitive baseline demonstrating the usefulness of the framework. This chapter establishes a robust evaluation and a benchmark for future work on this task. Experimental evidence suggests that entities are important for understanding event sequences.

Many NLP tasks can be framed as a transformation of one type of graph into another. The GTNN framework is sufficiently general that it can be applied to many other problems such as entity linking, event and argument extraction, entity and event co-reference, and dependency parsing. Many NLP tasks could benefit from the additional abstraction provided by GTNN.

While the GTNN framework makes local decisions, extending the framework into a joint or structured prediction framework would be helpful for many NLP tasks including event sequence extraction. For example, identifying entity roles, i.e. event argument relationships, could be beneficial to event sequence extraction. In general, replacing the heuristic linking functions used in this work with predicted links between events, entities, and entity mentions could allow the model to identify key relationships. Both unsupervised and supervised joint learning models may be helpful. Additional supervision will most likely be useful since it provides more information. However, a joint model with unsupervised tasks could essentially provide constraints for the overall task.

An additional transformation H_4 could be added to the work of this chapter. Aggregating event sequences would produce a representation for a story. Predicting a "same script" relation between stories could be used for script learning. This effectively allows for script learning to be incorporated into the framework as a joint task. Hence event detection, event sequence extraction, and finally script learning can all be jointly learned in the same framework.

Event sequence extraction provides the input for script learning which is explored in

Chapter 4. While script learning is the ultimate product of this thesis, the knowledge that is represented in scripts may be helpful for event sequence extraction and possibly even event detection. Many of the "after" relationships in event sequences are implied by the script to which the event sequence belongs. That is, the ordering of events can be inferred by common sense.

Chapter 4: Script Learning

4.1 Introduction

Scripts were developed as a means of representing stereotypical event sequences and interactions in narratives. The benefits of scripts for encoding common sense knowledge, filling in gaps in a story, resolving ambiguous references, and answering comprehension questions have been amply demonstrated in the early work in natural language understanding [74]. The earliest attempts to learn scripts were based on explanation-based learning, which can be characterized as example-guided deduction from first principles [21, 22]. While this approach is successful in generalizing from a small number of examples, it requires a strong domain theory, which limits its applicability.

More recently, some new graph-based algorithms for inducing script-like structures from text have emerged. "Narrative Chains" is a narrative model similar to Scripts [10]. Each Narrative Chain is a directed graph indicating the most frequent temporal relationship between the events in the chain. Narrative Chains are learned by a novel application of pairwise mutual information and temporal relation learning. Another graph learning approach employs Multiple Sequence Alignment in conjunction with a semantic similarity function to cluster sequences of event descriptions into a directed graph [71]. More recently still, graphical models have been proposed for representing script-like knowledge, but these lack the temporal component that is central to this chapter and to the early script work. These models instead focus on learning bags of related events [9, 40].

While the above approches demonstrate the learnability of script-like knowledge, they do not offer a probabilistic framework to reason robustly under uncertainty taking into account the temporal order of events. In this chapter, I present the first formal representation of scripts as Hidden Markov Models (HMMs), which support robust inference and effective

learning algorithms. The states of the HMM correspond to event types in scripts, such as entering a restaurant or opening a door. Observations correspond to natural language sentences that describe the event instances that occur in the story, e.g., "John went to Starbucks. He came back after ten minutes." The standard inference algorithms, such as the Forward-Backward algorithm, are able to answer questions about the hidden states given the observed sentences, for example, "What did John do in Starbucks?"

There are two complications that need to be dealt with to adapt HMMs to model narrative scripts. First, both the set of states, i.e. event types, and the set of observations are not pre-specified but are to be learned from data. The set of possible observations and the set of event types is assumed to be bounded but unknown. A clustering algorithm [71] is employed to reduce the natural language sentences, i.e. event descriptions, to a small set of observations and states based on their Wordnet similarity.

The second complication of narrative texts is that many events may be omitted either in the narration or by the event detection process. More importantly, there is no indication of a time lapse or a gap in the story, so the standard forward-backward algorithm does not apply. To account for this, the states are allowed to skip generating observations with some probability. This kind of HMM, with insertions and gaps, have been considered previously in speech processing [4] and in computational biology [41]. These models are refined by allowing state-dependent missingness, without introducing additional "insert states" or "delete states" as in [41]. In this work, attention is restricted to the so-called "Left-to-Right HMMs" which have acyclic graphical structure with possible self-loops, as they support more efficient inference algorithms than general HMMs and suffice to model most of the natural scripts.

The problem considered is that of learning the structure and parameters of scripts in the form of HMMs from sequences of natural language sentences. The proposed solution to script learning is a novel bottom-up method for structure learning, called *SEM-HMM*, which is inspired by Bayesian Model Merging (BMM) [79] and Structural Expectation Maximization (SEM) [29]. It starts with a fully enumerated HMM representation of the event sequences

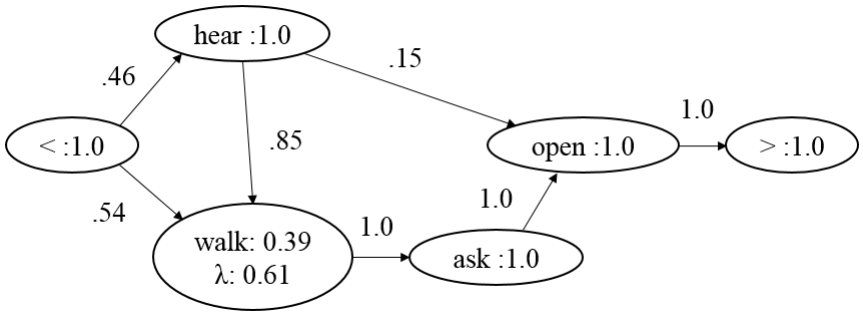

Figure 4.1: A portion of a learned "Answer the Doorbell" script

and incrementally merges states and deletes edges to improve the posterior probability of the structure and the parameters given the data. This approach is compared to several informed baselines on many natural datasets and shows its superior performance. I believe this work represents the first formalization of scripts that supports probabilistic inference and paves the way for robust understanding of natural language texts.

4.2 Problem Setup

Consider an activity such as answering the doorbell. An example HMM representation of this activity is illustrated in Figure 4.1. Each box represents a state, and the text within is a set of possible event descriptions (i.e. observations). Each event description is also marked with its conditional probability. Each edge represents a transition from one state to another and is annotated with its conditional probability.

In this thesis, a special class of HMM is considered with the following properties. First, some observations are allowed to be missing. This is a natural phenomenon in text, where not all events are mentioned or extracted. These are called null observations and are represented with a special symbol λ. Second, it is assumed that the states of the HMM can be ordered such that all transitions take place only in that order. These are called Left-to-Right HMMs in the literature [68, 4]. Self-transitions of states are permitted and represent "spurious" observations or events with multi-time step durations. While this work can be generalized

to arbitrary HMMs, it was observed that the Left-to-Right HMMs suffice to model scripts in this corpora.

Formally, an HMM is a 4-tuple (Q, T, O, Ω), where Q is a set of states, $T(q'|q)$ is the probability of transition from q to q', O is a set of possible non-null observations, and $\Omega(o|q)$ is the probability of observing o when in state q[1], where $o \in O \cup \{\lambda\}$, and q_n is the terminal state. An HMM is Left-to-Right if the states of the HMM can be ordered from q_0 thru q_n such that $T(q_j|q_i)$ is non-zero only if $i \leq j$. It is assumed that the target HMM is Left-to-Right. Each state is indexed according to a topological ordering of the transition graph. An HMM is a generative model of a distribution over sequences of observations. For convenience w.l.o.g. it is assumed that each time it is "run" to generate a sample, the HMM starts in the same initial state q_0, and goes through a sequence of transitions according to T until it reaches the same final state q_n, while emitting an observation in $O \cup \{\lambda\}$ in each state according to Ω. The initial state q_0 and the final state q_n respectively emit the distinguished observation symbols, "<" and ">" in O, which are emitted by no other state. The concatenation of observations in successive states consitutes a sample of the distribution represented by the HMM. Because the null observations are removed from the generated observations, the length of the output string may be smaller than the number of state transitions. It could also be larger than the number of *distinct* state transitions, since observations are allowed to be generated on the self transitions. Thus spurious and missing observations model insertions and deletions in the outputs of HMMs without introducing special states as in profile HMMs [41].

In this chapter, following problem is addressed: given a set of narrative texts, each of which describes a stereotypical event sequence drawn from a fixed but unknown distribution, learn the structure and parameters of a Left-to-Right HMM model that best captures the distribution of the event sequences. The algorithm is evaluated on natural datasets by how well the learned HMM can predict observations removed from the test sequences.

[1] Ω can be straightforwardly generalized to depend on both of the states in a state transition.

4.3 HMM-Script Learning

At the top level, the algorithm is input a set of documents D, where each document is a sequence of natural language sentences that describes the same stereotypical activity. The output of the algorithm is a Left-to-Right HMM that represents that activity.

This approach has four main components, which are described in the next four subsections: event detection, parameter estimation, structure learning, and structure scoring. The event detection step clusters the input sentences into event types and replaces the sentences with the corresponding cluster labels. This process assumes that each sentence contains a single event. After extraction, the event sequences are iteratively merged with the current HMM in batches of size r starting with an empty HMM. Structure Learning then merges pairs of states (nodes) and removes state transitions (edges) by greedy hill climbing guided by the improvement in approximate posterior probability of the HMM. Once the hill climbing converges to a local optimum, the maxmimum likelihood HMM parameters are re-estimated using the EM procedure based on all the data seen so far. Then the next batch of r sequences are processed. These steps will now be described in more detail.

4.3.1 Event Detection

Given a set of sequences of sentences, the event detection algorithm clusters them into events and arranges them into a tree structured HMM. For this step, it is assumed that each sentence has a simple structure that consists of a single verb and an object. The further simplifying assumption is made that the sequences of sentences in all documents describe the events in temporal order. Although this assumption is often violated in natural documents, this problem is ignored to focus on script learning. There have been some approaches in previous work that specifically address the problem of inferring temporal order of events from texts though that problem is not explored here [69].

While chapter 2 directly addresses the problem of event detection, a simpler approach

was taken for several reasons. First, despite being natural text, the OMICS data is much simpler in comparison to newswire articles, weblog posts, and the other documents types of ACE2005. A neural network model could have been employed; however the short, relatively consistent sentences of OMICS do not warrant such a complex method. Second, the ACE2005 corpus is much larger than the OMICS dataset and ACE2005 is relatively small compared to the datasets typically used in conjunction with neural networks. For that reason, any neural network would likely overfit the data. Finally, ACE2005 provides a clear ontology with 33 event types with annotation, whereas OMICS does not. An event detection system for OMICS needs to be unsupervised, whereas the methods studied in Chapter 2 are all supervised algorithms.

Given the above assumptions, a simple agglomerative clustering algorithm that uses a semantic similarity function $Sim(S_1, S_2)$ [71] is applied over sentence pairs (S_1, S_2) and is parameterized as $w_1 PS(V_1, V_2) + w_2 PS(O_1, O_2)$, where V_i is the verb, O_i is the object in the sentence S_i, and $PS(w, v)$ is the path similarity metric from Wordnet [59]. It is applied to the first verb (preferring verbs that are not stop words) and to the objects from each pair of sentences. The constants w_1 and w_2 are tuning parameters that adjust the relative importance of each component. Like the sequence alignment approach [71], it was found that a high weight on the verb similarity was important to finding meaningful clusters of events. The most frequent verb in each cluster is extracted to name the event type that corresponds to that cluster.

The initial configuration of the HMM is a Prefix Tree Acceptor, which is constructed by starting with a single event sequence and then adding sequences by branching the tree at the first place the new sequence differs from it [25, 75]. By repeating this process, an HMM that fully enumerates the data is constructed.

4.3.2 Parameter Estimation with EM

In this section, the proposed parameter estimation methods are described. While parameter estimation in this kind of HMM was treated earlier in the literature [68, 4], this work provides a more principled approach to estimate the state-dependent probability of λ transitions from data without introducing special insert and delete states [41]. It is assumed that the structure of the Left-to-Right HMM is fixed based on the preceding structure learning step, which is described in Section 4.3.3.

The main difficulty in HMM parameter estimation is that the states of the HMM are not observed. The Expectation-Maximization (EM) procedure (also called the Baum-Welch algorithm in HMMs) alternates between estimating the hidden states in the event sequences by running the Forward-Backward algorithm (the Expectation step) and finding the maximum likelihood estimates (the Maximization step) of the transition and observation parameters of the HMM [6]. Unfortunately, because of the λ-transitions, the state transitions of the proposed HMM are not necessarily aligned with the observations. Hence it is necessary to explicitly maintain two indices, the time index t and the observation index i. $\alpha_{q_j}(t, i)$ is defined to be the joint probability that the HMM is in state q_j at time t and has made the observations $\vec{o}_{0,i}$. This is computed by the forward pass of the algorithm using the following recursion. Equations 4.1 and 4.2 represent the base case of the recursion, while Equation 4.3 represents the case for null observations. Note that the observation index i of the recursive call is not advanced unlike in the second half of Equation 4.3 where it is advanced for a normal observation. The fact that the HMM is Left-to-Right is exploited by only considering transitions to j from states with indices $k \leq j$. The time index t is incremented starting 0,

and the observation index i varies from 0 thru m.

$$\alpha_{q_0}(0,0) = 1 \tag{4.1}$$

$$\forall j > 0, \alpha_{q_j}(0,0) = 0 \tag{4.2}$$

$$\alpha_{q_j}(t,i) = \sum_{0 \leq k \leq j} T(q_j|q_k)\{\Omega(\lambda|q_j)\alpha_{q_k}(t-1,i) \tag{4.3}$$

$$+ \Omega(o_i|q_j)\alpha_{q_k}(t-1,i-1)\} \tag{4.4}$$

The backward part of the standard Forward-Backward algorithm starts from the last time step τ and reasons backwards. Unfortunately in this setting, τ—the true number of state transitions— is not known, as some of the observations are missing. Hence, $\beta_{q_j}(t,i)$ is defined as the conditional probability of observing $\vec{o}_{i+1,m}$ in the remaining t steps given that the current state is q_j. This allows for t to be incremented starting from 0 as recursion proceeds, rather than decrementing it from τ.

$$\beta_{q_n}(0,m) = 1 \tag{4.5}$$

$$\forall j < n, \beta_{q_j}(0,m) = 0 \tag{4.6}$$

$$\beta_{q_j}(t,i) = \sum_{j \leq k} T(q_k|q_j)\{\Omega(\lambda|q_k)\beta_{q_k}(t-1,i) \tag{4.7}$$

$$+ \Omega(o_{i+1}|q_k)\beta_{q_k}(t-1,i+1)\} \tag{4.8}$$

Equation 4.9 calculates the probability of the observation sequence $z = P(\vec{o})$, which is computed by marginalizing $\alpha_q(t,m)$ over time t and state q and setting the second index i to the length of the observation sequence m. The quantity z serves as the normalizing factor

for the last three equations.

$$z = P(\vec{o}) = \sum_{q \in Q} \sum_{t} \alpha_q(t, m) \tag{4.9}$$

$$\gamma_q(t, i) = P(q|\vec{o}) = z^{-1} \sum_{\tau} \alpha_q(t, i)\beta_q(\tau - t, i) \tag{4.10}$$

$$\delta_{q,q'\uparrow\lambda}(t) = P(q \to q', \lambda|\vec{o}) = z^{-1}T(q'|q)\Omega(\lambda|q') \tag{4.11}$$

$$\sum_{\tau} \sum_{i} \{\alpha_q(t, i)\beta_{q'}(\tau - t - 1, i)\} \tag{4.12}$$

$$\forall o \in \Omega, \delta_{q,q'\uparrow o}(t) = P(q \to q', o|\vec{o}) \tag{4.13}$$

$$= z^{-1}T(q'|q)\Omega(o|q') \tag{4.14}$$

$$\sum_{\tau} \sum_{i} \{\alpha_q(t, i)I(o_{i+1} = o)\beta_{q'}(\tau - t - 1, i + 1)\} \tag{4.15}$$

In equation 4.10, the joint distribution of the state and observation index γ at time t is computed by convolution, i.e. multiplying the α and β that correspond to the same time step and the same state and marginalizing out the length of the state-sequence τ. Convolution is necessary, as the length of the state-sequence τ is a random variable equal to the sum of the corresponding time indices of α and β.

Equation 4.11 computes the joint probability of a state-transition associated with a null observation by first multiplying the state transition probability by the null observation probability given the state transition and the appropriate α and β values. It then marginalizes out the observation index i. Again, a convolution with respect to τ needs to be computed to take into account the variation over the total number of state transitions. Equation 4.13 calculates the same probability for a non-null observation o. This equation is similar to Equation 4.11 with two differences. First, it is ensured that the observation is consistent with o by multiplying the product with the indicator function $I(o_{i+1} = o)$ which is 1 if $o_{i+1} = o$ and 0 otherwise. Second, the observation index i is advanced in the β function.

Since the equations above are applied to each individual observation sequence, α, β, γ, and δ all have an implicit index s which denotes the observation sequence and has been

omitted in the above equations. The sequence will be made explicit below in calculating the expected counts of state visits, state transitions, and state transition observation triples.

$$\forall q \in Q, C(q) = \sum_{s,t,i} \gamma_q(s,t,i) \tag{4.16}$$

$$\forall q, q' \in Q, C(q \to q') = \sum_{s,t,o \in \Omega \bigcup \{\lambda\}} \delta_{q,q' \uparrow o}(s,t) \tag{4.17}$$

$$\forall q, q' \in Q, o \in \Omega \bigcup \{\lambda\}, \tag{4.18}$$

$$C(q, q' \uparrow o) = \sum_{s,t} \delta_{q,q' \uparrow o}(s,t) \tag{4.19}$$

Equation 4.16 counts the total expected number of visits to each state in the data. Equation 4.17 estimates the expected number of transitions between each state pair. Finally, Equation 4.18 computes the expected number of observations and state-transitions including null transitions. This concludes the E-step of the EM procedure.

The M-step of the EM procedure consists of Maximum Aposteriori (MAP) estimation of the transition and observation distributions assuming an uninformative Dirichlet prior. This amounts to adding a pseudocount of 1 to each of the next states and observation symbols. The observation distributions for the initial and final states q_0 and q_n are fixed to be the Kronecker delta distributions at their true values.

$$\hat{T}(q'|q) = \frac{C(q \to q') + 1}{[C(q) + \sum_{p' \in Q} 1]} \tag{4.20}$$

$$\hat{\Omega}(o|q') = \frac{\sum_q C(q, q' \uparrow o) + 1}{\sum_{o'} \{\sum_q C(q, q' \uparrow o')\} + 1} \tag{4.21}$$

The E-step and the M-step are repeated until convergence of the parameter estimates.

4.3.3 Structure Learning

The proposed structure learning algorithm, SEM-HMM, will now be described. The algorithm is inspired by Bayesian Model Merging (BMM) [79] and Structural EM (SEM) [29] and adapts them to learning HMMs with missing observations. SEM-HMM performs a greedy hill climbing search through the space of acyclic HMM structures. It iteratively proposes changes to the structure either by merging states or by deleting edges. It evaluates each change and makes the merge with the best score. An exact implementation of this method is expensive, because, each time a structure change is considered, the MAP parameters of the structure given the data must be re-estimated. One of the key insights of both SEM and BMM is that this expensive re-estimation can be avoided in factored models by incrementally computing the changes to various expected counts using only local information. While this calculation is only approximate, it is highly efficient.

During the structure search, the algorithm considers every possible structure change, i.e. merging of pairs of states and deletion of state-transitions, checks that the change does not create cycles, evaluates it according to the scoring function, and selects the best scoring structure. This is repeated until the structure can no longer be improved (see Algorithm 1).

Algorithm 1 Script induction via SEM-HMM

```
procedure LEARN(Model M, Data D, Changes S)
    while NotConverged do
        M = AcyclicityFilter (S(M))
        M* = argmax_{M'∈M} P(M'|D)
        if P(M*|D) ≤ P(M|D) then
            return M
        else
            M = M*
        end if
    end while
end procedure
```

The Merge States operator creates a new state from the union of a state pair's transition and observation distributions. It must assign transition and observation distributions to the new merged state. To be exact, the parameter estimation needs to be redone for the changed structure. To compute the impact of several proposed changes efficiently, it is assumed that

all probabilistic state-transitions and trajectories for the observed sequences remain the same as before except in the changed parts of the structure. In this work, the assumption will be called "locality of change," which allows for the addition of the corresponding expected counts from the states being merged as shown below.

$$C(r) = C(p) + C(q) \tag{4.22}$$

$$C(r \to s) = C(p \to s) + C(q \to s) \tag{4.23}$$

$$C(s \to r) = C(s \to p) + C(s \to q) \tag{4.24}$$

$$C(r, s \uparrow o) = C(p, s \uparrow o) + C(q, s \uparrow o) \tag{4.25}$$

$$C(s, r \uparrow o) = C(s, p \uparrow o) + C(s, q \uparrow o) \tag{4.26}$$

The second kind of structure change that is considered is edge deletion. It consists of removing a transition between two states and redistributing its evidence along the other paths between the same states. Again, making the locality of change assumption, only the parameters of the transition and observation distributions that occur in the paths between the two states need to be recomputed. The parameters affected by deleting an edge (q_s, q_e) are re-estimated by effectively redistributing the expected transitions from q_s to q_e, $C(q_s \to q_e)$, among other edges between q_s and q_e based on the parameters of the current model.

This is done efficiently using a procedure similar to the Forward-Backward algorithm under the null observation sequence. Algorithm 4.3.3 takes the current model M, an edge $(q_s \to q_e)$, and the expected count of the number of transitions from q_s to q_e, $N = C(q_s \to q_e)$, as inputs. It updates the counts of the other transitions to compensate for removing the edge between q_s and q_e. It initializes the α of q_s and the β of q_e with 1 and the rest of the αs and βs to 0. It makes two passes through the HMM, first in the topological order of the nodes in the graph and the second in the reverse topological order. In the first, "forward" pass from q_s to q_e, it calculates the α value of each node q_i that represents the probability that a sequence that passes through q_s also passes through q_i while emitting no observation. In the

second, "backward" pass, it computes the β value of a node q_i that represents the probability that a sequence that passes through q_i emits no observation and later passes through q_e. The product of $\alpha(q_i)$ and $\beta(q_i)$ gives the probability that q_i is passed through when going from q_s to q_t and emits no observation. Multiplying it by the expected number of transitions N gives the expected number of additional counts which are added to $C(q_i)$ to compensate for the deleted transition $(q_s \rightarrow q_e)$. After the distribution of the evidence, all the transition and observation probabilities are re-estimated for the nodes and edges affected by the edge deletion.

Algorithm 2 Forward-Backward algorithm to delete an edge and re-distribute the expected counts.

procedure DELETEEDGE(Model M, edge $(q_s \rightarrow q_e)$, count N)
  $\forall i s.t. s \leq i \leq e, \alpha(q_i) = \beta(q_i) = 0$
  $\alpha(q_s) = \beta(q_e) = 1$
  for $i = s + 1$ **to** e **do**
    for all $q_p \in Parents(q_i)$ **do**
      $\alpha(q_p \rightarrow q_i) = \alpha(q_p) T(q_i|q_p) \Omega(\lambda|q_i)$
      $\alpha(q_i) = \alpha(q_i) + \alpha(q_p \rightarrow q_i)$
    end for
  end for
  for $i = e - 1$ **downto** s **do**
    for all $q_c \in Children(q_i)$ **do**
      $\beta(q_i \rightarrow q_c) = \beta(q_c) T(q_c|q_i) \Omega(\lambda|q_c)$
      $C(q_i \rightarrow q_c) = C(q_i \rightarrow q_c) + \alpha(q_i \rightarrow q_c) \beta(q_i \rightarrow q_c) N$
      $C(q_i) = C(q_i) + C(q_i \rightarrow q_c)$
      $\beta(q_i) = \beta(q_i) + \beta(q_i \rightarrow q_c)$
    end for
  end for
end procedure

In principle, one could continue making incremental structural changes and parameter updates and never run EM again. This is exactly what is done in Bayesian Model Merging (BMM) [79]. However, a series of structural changes followed by approximate incremental parameter updates could lead to bad local optima. Hence, after merging each batch of r sequences into the HMM, we re-run EM for parameter estimation on all sequences seen thus far.

4.3.4 Structure Scoring

This section describes how to score the structures produced and select the best one. A Bayesian scoring function is employed, which is the posterior probability of the model given the data, denoted $P(M|D)$. The score is decomposed via Bayes Rule (i.e. $P(M|D) \propto P(M)P(D|M)$)), and the denominator is omitted since it is invariant with regards to the model.

Since each observation sequence is independent of the others, the data likelihood $P(D|M) = \Pi_{\vec{o} \in D} P(\vec{o})$ is calculated using the Forward-Backward algorithm and Equation 4.9 in Section 4.3.2. Because the initial model fully enumerates the data, any merge can only reduce the data likelihood. Hence, the model prior $P(M)$ must be designed to encourage generalization via state merges and edge deletions (described in Section 4.3.3). A prior with three components is employed: the first two components are syntactic and penalize the number of states $|Q|$ and the number of non-zero transitions $|T|$ respectively. The third component penalizes the number of frequently-observed semantic constraint violations $|C|$. In particular, the prior probability of the model $P(M) = \frac{1}{Z} \exp(-(\kappa_q|Q| + \kappa_t|T| + \kappa_c|C|))$. The κ parameters assign weights to each component in the prior.

The semantic constraints are learned from the event sequences for use in the model prior. The constraints take the simple form "X never follows Y." They are learned by generating all possible such rules using pairwise permutations of event types, and evaluating them on the training data. In particular, the number of times each rule is violated is counted and a z-test is performed to determine if the violation rate is lower than a predetermined error rate. Those rules that pass the hypothesis test with a threshold of 0.01 are included. When evaluating a model, these contraints are considered violated if the model could generate a sequence of observations that violates the constraint.

In addition to incrementally computing the transition and observation counts, $C(r \rightarrow s)$ and $C(r, s \uparrow o)$, the likelihood, $P(D|M)$, can be incrementally updated with structure changes

as well. Note that the likelihood can be expressed as the following:

$$P(D|M) = \prod_{q,r \in Q} \prod_{o \in O} T(r|q)^{C(q \to r)} \Omega(o|r)^{C(q, r \uparrow o)} \tag{4.27}$$

when the state transitions are observed. Since the state transitions are not actually observed, the above expression is approximated by replacing the observed counts with expected counts. Further, the locality of change assumption allows for easy calculation of the effect of changed expected counts and parameters on the likelihood by dividing it by the old products and multiplying by the new products. This version of the algorithm is called SEM-HMM-Approx.

4.4 Experiments and Results

The experimental results on SEM-HMM and SEM-HMM-Approx are now presented. The evaluation task is to predict missing events from an observed sequence of events. For comparison, four baselines were also evaluated. The "Frequency" baseline predicts the most frequent event in the training set that is not found in the observed test sequence. The "Conditional" baseline predicts the next event based on what most frequently follows the prior event. A third baseline, referred to as "BMM," is a version of the proposed algorithm that does not use EM for parameter estimation and instead only incrementally updates the parameters starting from the raw document counts. Further, it learns a standard HMM, that is, without λ transitions. This is very similar to the Bayesian Model Merging approach for HMMs [79]. The fourth baseline is the same as above, but uses the EM algorithm for parameter estimation without λ transitions. It is referred to as "BMM + EM."

The Open Minds Indoor Common Sense (OMICS) corpus was developed by the Honda Research Institute and is based upon the Open Mind Common Sense project [32]. It describes 175 common household tasks with each task having 14 to 122 narratives describing, in short sentences, the necessary steps to complete it. Each narrative consists of temporally ordered, simple sentences from a single author that describe a plan to accomplish a task. Examples

Batch Size r	2	5	10
SEM-HMM	42.2%	45.1%	**46.0%**
SEM-HMM Approx.	43.3%	43.5%	**44.3%**
BMM + EM	41.1%	41.2%	42.1%
BMM	41.0%	39.5%	39.1%
Conditional Frequency			36.2% 27.3%

Table 4.1: The average accuracy on the OMICS domains

Example 1	Example 2
Hear the doorbell.	Listen for the doorbell.
Walk to the door.	Go towards the door.
Open the door.	Open the door.
Allow the people in.	Greet the vistor.
Close the door.	See what the visitor wants.
	Say goodbye to the visitor.
	Close the door.

Table 4.2: Examples from the OMICS "Answer the Doorbell" task with event triggers underlined

from the "Answer the Doorbell" task can be found in Table 2. The OMICS corpus has 9044 individual narratives, and its short and relatively consistent language lends itself to relatively easy event detection.

The 84 domains with at least 50 narratives and 3 event types were used for evaluation. For each domain, forty percent of the narratives were withheld for testing, each with one randomly-chosen event omitted. The model was evaluated on the proportion of correctly predicted events given the remaining sequence. On average each domain has 21.7 event types with a standard deviation of 4.6. The average narrative length across domains is 3.8 with standard deviation of 1.7. This implies that only a fraction of the event types are present in any given narrative. There is a high degree of omission of events, and there are many different ways of accomplishing each task. Hence, the prediction task is reasonably difficult, as evidenced by the performance of simple baselines. Neither the frequency of events nor simple temporal structure is enough to accurately fill in the gaps. It indicates that more

sophisticated modeling such as SEM-HMM is needed.

The average accuracy across the 84 domains for each method is found in Table 4.4. On average the proposed method significantly out-performed all the baselines, with the average improvement in accuracy across OMICS tasks between SEM-HMM and each baseline being statistically significant at a .01 level across all pairs and on sizes of $r = 5$ and $r = 10$ using one-sided paired t-tests. For $r = 2$, improvement was not statistically greater than zero. The results improve with batch size r until $r = 10$ for SEM-HMM and BMM+EM, but they decrease with batch size for BMM without EM. Both of the methods that use EM depend on statistics to be robust and hence need a larger r value to be accurate. However for BMM, a smaller r size means it reconciles a couple of documents with the current model in each iteration, which ultimately helps guide the structure search. The accuracy for "SEM-HMM Approx." is close to the exact version at each batch level, while only taking half the time on average.

4.5 Conclusions

In this chapter, the first formal treatment of scripts as HMMs with missing observations was proposed. We adapted the HMM inference and parameter estimation procedures to scripts and developed a new structure learning algorithm, SEM-HMM, based on the EM procedure. It improves upon BMM by allowing for missing observations and by incorporating maximum likelihood parameter estimation via EM. The proposed algorithm is shown to be effective in learning scripts from documents and performs better than other baselines on sequence prediction tasks. Thanks to the assumption of missing observations, the graphical structure of the scripts is usually sparse and intuitive.

There are several avenues for future work. First, the inclusion of entities as arguments to events is an important extension. This will enable representing the roles of people, places, and things within the context of a script. Understanding entities and the roles they occupy

can be used for additional inferences, in particular when references to actors or participants are implied or omitted.

Another consideration is to use scripts themselves as knowledge to guide event detection and event sequence extraction. The three chapters of this thesis form a pipeline, producing scripts. Including scripts as input for the pipeline could refine the entire process. This could be iteratively repeated until no further improvements are found.

Finally, instead of viewing scripts as a final product of this pipeline, scripts could be created as a byproduct. Scripts are often viewed as a representation of common sense, that is, knowledge that does not need to be explicitly stated because it is assumed by the text. When performing event sequence extraction, it is reasonable to believe that some temporal relationships are stated and others are assumed. Scripts could be used as a latent variable in the model, such that a relationship is either explicitly stated, is implied by the script, or does not exist. An event sequence extraction model could choose between extracting an explicit relationship or one implied by a script. Scripts would be learned indirectly based on their utility in assisting with event sequence extraction.

Chapter 5: Conclusion & Future Work

This thesis explored the task of narrative understanding and its three primary components: event detection, relation extraction, and script learning. Chapter 2 detailed a new event detection model named DAG-GRU that incorporates syntactic relationships into its structure. It identified the on-average improvement of document-level models over sentence-level models, in particular, the strength of the EntNet memory network. The model along with replications of other models were evaluated with rigor, accounting for variation due to the initial parameter values of neural networks and due to the variety of documents contained in the standard dataset. Overall, the robust evaluation demonstrated that the variation observed was often greater than the differences in the state-of-the-art. Finally, an analysis of the predictions between two different models surprisingly showed that despite the different network architectures, they largely made the same predictions and errors.

In Chapter 3, a new framework called graph transforming networks (GTNN) was proposed to be a general method for relation extraction. It represents the task as a series of graph transformations, each transformation corresponding to a related sub-task. Each transformation is performed by a neural network, the modularity of the framework allows for various models to be easily instantiated. The framework was evaluated on both event sequence extraction as well as ACE2005's entity relation extraction task.

Finally, in Chapter 4 a new variation of Hidden Markov Models that allows for omitted events was discussed along with a procedure for inducing the HMMs from event sequences. The formulation of scripts as HMMs is novel, as is the algorithm for learning the HMMs which is based on Structural Expectation Maximization and Bayesian Model Merging. The result was scripts (learned from natural texts) that were effective in inferring events omitted from event sequences.

5.1 Future Work

Currently, the GTNN model lags the state-of-the-art in entity relation extraction. One way GTNN differs from most, if not all, relation extraction models is by modeling segments of the document rather than relation-mention pairs. The standard procedure is to take the context, i.e., a sentence fragment, between two entities and build a representation of it with a neural network e.g. LSTM. GTNN builds a representation for each entity mention using the entire document as context. However, the relation mention strategy may have an advantage, because the representations it builds are for pairs of entity mentions. In the standard approach, each representation is used for a single relation extraction decision, rather than an entity mention representation in GTNN which is used in multiple decisions.

For the DAG-GRU model, the obvious next step is to apply it to the full event extraction task rather than only to the event detection task. Event extraction involves identifying entity mentions that participate in the event and have specified role type. This additional step is called argument linking. An event extraction model that jointly detects events and links arguments based on DAG-GRU would be a natural extension, since there is evidence that syntactic information is useful for identifying events [28].

One more avenue of extension is to convert the three tasks presented into a single joint model. This would allow for script-level knowledge and context to be used while detecting events and constructing event sequences. Knowing the co-occurrence of events as well as the probability of omissions is likely helpful for event detection and event sequence extraction.

Finally, all stages of narrative understanding, including a joint model could benefit from additional data. The ACE2005 dataset is relatively small, and given the nuance of the tasks, additional data is likely required for substantial further progress. Though several recent approaches to event detection have utilized additional data, the extended datasets are still relatively small and have not produced large improvements. Adapting the tasks to semi-supervised, unsupervised, or transfer learning problems are some potential ways of extending the data.

Bibliography

[1] David Ahn. The stages of event extraction. In *Proceedings of the Workshop on Annotating and Reasoning about Time and Events*, pages 1–8. Association for Computational Linguistics, 2006.

[2] Jun Araki, Zhengzhong Liu, Eduard H Hovy, and Teruko Mitamura. Detecting subevent structure for event coreference resolution. In *LREC*, pages 4553–4558, 2014.

[3] Jun Araki and Teruko Mitamura. Joint event trigger identification and event coreference resolution with structured perceptron. In *EMNLP*, pages 2074–2080, 2015.

[4] L. R. Bahl, F. Jelinek, and R. L. Mercer. A maximum likelihood approach to continuous speech recognition. *IEEE Transactions in Pattern Analysis and Machine Intelligence (PAMI)*, 5(2):179–190, 1983.

[5] Collin F Baker, Charles J Fillmore, and John B Lowe. The berkeley framenet project. In *Proceedings of the 17th International Conference on Computational linguistics-Volume 1*, pages 86–90. Association for Computational Linguistics, 1998.

[6] Leonard E Baum, Ted Petrie, George Soules, and Norman Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The Annals of Mathematical Statistics*, 41(1):164–171, 1970.

[7] Steven Bethard, Leon Derczynski, Guergana Savova, James Pustejovsky, and Marc Verhagen. Semeval-2015 task 6: Clinical tempeval. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 806–814. Association for Computational Linguistics Denver, Colorado, 2015.

[8] Steven Bethard, Guergana Savova, Wei-Te Chen, Leon Derczynski, James Pustejovsky, and Marc Verhagen. Semeval-2016 task 12: Clinical tempeval. *Proceedings of SemEval*, pages 1052–1062, 2016.

[9] Nathanael Chambers. Event schema induction with a probabilistic entity-driven model. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1797–1807, 2013.

[10] Nathanael Chambers and Dan Jurafsky. Unsupervised learning of narrative event chains. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 789–797, 2008.

[11] Chen Chen and V Incent NG. Joint modeling for chinese event extraction with rich linguistic features. In *In COLING*. Citeseer, 2012.

[12] Yubo Chen, Liheng Xu, Kang Liu, Daojian Zeng, and Jun Zhao. Event extraction via dynamic multi-pooling convolutional neural networks. In *ACL (1)*, pages 167–176, 2015.

[13] Zheng Chen and Heng Ji. Language specific issue and feature exploration in chinese event extraction. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pages 209–212. Association for Computational Linguistics, 2009.

[14] Fei Cheng and Yusuke Miyao. Classifying temporal relations by bidirectional lstm over dependency paths. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 1–6, 2017.

[15] Timothy Chklovski and Patrick Pantel. Verbocean: Mining the web for fine-grained semantic verb relations. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, 2004.

[16] Prafulla Kumar Choubey and Ruihong Huang. A sequential model for classifying temporal relations between intra-sentence events. In *the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1796–1802, 2017.

[17] Prafulla Kumar Choubey and Ruihong Huang. Improving event coreference resolution by modeling correlations between event coreference chains and document topic structures. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 485–495, 2018.

[18] Fenia Christopoulou, Makoto Miwa, and Sophia Ananiadou. A walk-based model on entity graphs for relation extraction. In *the 56th Annual Meeting of the Association for Computational Linguistics (Short Papers)*, pages 81–88. Association for Computational Linguistics, 2018.

[19] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[20] Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 1–8. Association for Computational Linguistics, 2002.

[21] Gerald DeJong. Generalizations based on explanations. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence (IJCAI)*, pages 67–69, 1981.

[22] Gerald DeJong and Raymond Mooney. Explanation-based learning: An alternative view. *Machine learning*, 1(2):145–176, 1986.

[23] Dmitriy Dligach, Timothy Miller, Chen Lin, Steven Bethard, and Guergana Savova. Neural temporal relation extraction. *EACL 2017*, page 746, 2017.

[24] Shaoyang Duan, Ruifang He, and Wenli Zhao. Exploiting document level information to improve event detection via recurrent neural networks. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 352–361, 2017.

[25] Pierre Dupont, Laurent Miclet, and Enrique Vidal. What is the search space of the regular inference? In *Grammatical Inference and Applications*, pages 25–37. Springer, 1994.

[26] Bradley Efron. Bootstrap methods: another look at the jackknife. In *Breakthroughs in statistics*, pages 569–593. Springer, 1992.

[27] Xiaocheng Feng, Lifu Huang, Duyu Tang, Heng Ji, Bing Qin, and Ting Liu. 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)*, volume 2, pages 66–71, 2016.

[28] Baobao Chang Zhifang Sui Feng Qian, Lei Sha. Jointly extracting event triggers and arguments by dependency-bridge rnn and tensor-based argument interaction. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[29] Nir Friedman. The Bayesian structural EM algorithm. In *Proceedings of the Fourteenth conference on Uncertainty in Artificial Intelligence*, pages 129–138. Morgan Kaufmann Publishers Inc., 1998.

[30] Reza Ghaeini, Xiaoli Z Fern, Liang Huang, and Prasad Tadepalli. Event nugget detection with forward-backward recurrent neural networks. In *The 54th Annual Meeting of the Association for Computational Linguistics*, page 369, 2016.

[31] Ralph Grishman, David Westbrook, and Adam Meyers. Nyus english ace 2005 system description. *ACE*, 5, 2005.

[32] Rakesh Gupta and Mykel J Kochenderfer. Common sense data acquisition for indoor mobile robots. In *AAAI*, pages 605–610, 2004.

[33] Mikael Henaff, Jason Weston, Arthur Szlam, Antoine Bordes, and Yann LeCun. Tracking the world state with recurrent entity networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

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

[35] Yu Hong, Jianfeng Zhang, Bin Ma, Jianmin Yao, Guodong Zhou, and Qiaoming Zhu. Using cross-entity inference to improve event extraction. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 1127–1136. Association for Computational Linguistics, 2011.

[36] Heng Ji and Ralph Grishman. Refining event extraction through cross-document inference. In *ACL*, pages 254–262, 2008.

[37] Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley. Molecular graph convolutions: moving beyond fingerprints. *Journal of computer-aided molecular design*, 30(8):595–608, 2016.

[38] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *the 3rd International Conference on Learning Representations (ICLR)*, 2014.

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

[40] Jackie Chi Kit Cheung, Hoifung Poon, and Lucy Vanderwende. Probabilistic frame induction. In *Proceedings of Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL:HLT)*, pages 837–846, 2013.

[41] Anders Krogh, Michael Brown, I. Saira Mian, Kimmen Sjolander, and David Haussler. Hidden markov models in computational biology. *Journal of Molecular Biology*, pages 1501–1531, 1994.

[42] Jey Han Lau and Timothy Baldwin. An empirical evaluation of doc2vec with practical insights into document embedding generation. In *1st Workshop on Representation Learning for NLP*, 2016.

[43] Quoc V Le and Tomas Mikolov. Distributed representations of sentences and documents. In *ICML*, volume 14, pages 1188–1196, 2014.

[44] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[45] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[46] Peifeng Li, Guodong Zhou, Qiaoming Zhu, and Libin Hou. Employing compositional semantics and discourse consistency in chinese event extraction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1006–1016. Association for Computational Linguistics, 2012.

[47] Qi Li, Heng Ji, and Liang Huang. Joint event extraction via structured prediction with global features. In *the 51st Annual Meeting of the Association for Computational Linguistics*, pages 73–82, 2013.

[48] Shasha Liao and Ralph Grishman. 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. Association for Computational Linguistics, 2010.

[49] Jian Liu, Yubo Chen, Kang Liu, and Jun Zhao. Event detection via gated multilingual attention mechanism. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[50] Shulin Liu, Yubo Chen, Shizhu He, Kang Liu, and Jun Zhao. Leveraging framenet to improve automatic event detection. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 2134–2143, 2016.

[51] Shulin Liu, Yubo Chen, Kang Liu, and Jun Zhao. Exploiting argument information to improve event detection via supervised attention mechanisms. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1789–1798, 2017.

[52] Weiyi Lu and Thien Huu Nguyen. Similar but not the same: Word sense disambiguation improves event detection via neural representation matching. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4822–4828, 2018.

[53] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. In *the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, 2015.

[54] Mingbo Ma, Liang Huang, Bing Xiang, and Bowen Zhou. Dependency-based convolutional neural networks for sentence embedding. In *the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Short Papers)*, pages 174–179, 2015.

[55] Sean MacAvaney, Arman Cohan, and Nazli Goharian. Guir at semeval-2017 task 12: A framework for cross-domain clinical temporal information extraction. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 1024–1029, 2017.

[56] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60, 2014.

[57] Diego Marcheggiani and Ivan Titov. Encoding sentences with graph convolutional networks for semantic role labeling. In *the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1506–1515, 2017.

[58] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Hlt-naacl*, volume 13, pages 746–751, 2013.

[59] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[60] Michael C Mozer. A focused backpropagation algorithm for temporal. *Backpropagation: Theory, architectures, and applications*, page 137, 1995.

[61] Thien Huu Nguyen, Kyunghyun Cho, and Ralph Grishman. Joint event extraction via recurrent neural networks. In *Proceedings of NAACL-HLT*, pages 300–309, 2016.

[62] Thien Huu Nguyen and Ralph Grishman. Event detection and domain adaptation with convolutional neural networks. In *ACL (2)*, pages 365–371, 2015.

[63] 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.

[64] 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.

[65] J. Walker Orr, Prasad Tadepalli, and Xiaoli Fern. Event detection with neural networks: A rigorous empirical evaluation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 999–1004, 2018.

[66] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014.

[67] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Human Language Technologies: The 2018 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2018.

[68] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Readings in Speech Recognition*, pages 267–296, 1990.

[69] Preethi Raghavan, Eric Fosler-Lussier, and Albert M. Lai. Learning to temporally order medical events in clinical text. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 70–74, 2012.

[70] Lance A Ramshaw and Mitchell P Marcus. Text chunking using transformation-based learning. In *Natural language processing using very large corpora*, pages 157–176. Springer, 1999.

[71] Michaela Regneri, Alexander Koller, and Manfred Pinkal. Learning script knowledge with web experiments. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 979–988. Association for Computational Linguistics, 2010.

[72] Frank Rosenblatt. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, DTIC Document, 1961.

[73] Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Tim Lillicrap. A simple neural network module for relational reasoning. In *Advances in neural information processing systems*, pages 4974–4983, 2017.

[74] Roger Schank and Robert P Abelson. Scripts, plans, goals and understanding: An introduction into human knowledge structures, 1977.

[75] Kristie Seymore, Andrew McCallum, and Roni Rosenfeld. Learning hidden Markov model structure for information extraction. In *AAAI Workshop on Machine Learning for Information Extraction*, pages 37–42, 1999.

[76] Tomohide Shibata, Hongkai Li, Tomohiro Sakaguchi, and Sadao Kurohashi. Kyotou at tac kbp 2017 event track: Neural network-based event sequence classification model. In *TAC*, 2017.

[77] Robert Speer and Catherine Havasi. Representing general relational knowledge in conceptnet 5. In *LREC*, pages 3679–3686, 2012.

[78] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[79] Andreas Stolcke and Stephen M. Omohundro. Hidden markov model induction by bayesian model merging. In *Advances in Neural Information Processing Systems (NIPS)*, pages 11–18, 1992.

[80] Naushad UzZaman, Hector Llorens, Leon Derczynski, James Allen, Marc Verhagen, and James Pustejovsky. Semeval-2013 task 1: Tempeval-3: Evaluating time expressions, events, and temporal relations. In *Second Joint Conference on Lexical and Computational Semantics (* SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)*, volume 2, pages 1–9, 2013.

[81] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[82] Marc Verhagen, Robert Gaizauskas, Frank Schilder, Mark Hepple, Graham Katz, and James Pustejovsky. Semeval-2007 task 15: Tempeval temporal relation identification. In *Proceedings of the 4th International Workshop on Semantic Evaluations*, pages 75–80. Association for Computational Linguistics, 2007.

[83] Marc Verhagen, Roser Sauri, Tommaso Caselli, and James Pustejovsky. Semeval-2010 task 13: Tempeval-2. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, pages 57–62. Association for Computational Linguistics, 2010.

[84] Christopher Walker, Stephanie Strassel, Julie Medero, and Kazuaki Maeda. Ace 2005 multilingual training corpus. *Linguistic Data Consortium, Philadelphia*, 2006.

[85] Yu Wang, Jihong Li, Ruibo Wang, and Wingli Yang. Confidence interval for f1 measure of algorithm performance based on blocked 32 cross-validation. *IEEE Transactions on Knowledge and Data Engineering*, 27:651–659, 2015.

[86] Bishan Yang and Tom M. Mitchell. Joint extraction of events and entities within a document context. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 289–299, San Diego, California, June 2016. Association for Computational Linguistics.

[87] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, 2016.

APPENDICES

Appendix A: Event Details

Event Type	**Training**	**Dev.**	**Test**
Acquit	5	0	1
Appeal	30	7	6
Arrest-Jail	78	4	6
Attack	1273	177	93
Be-Born	47	0	3
Charge-Indict	96	2	8
Convict	64	6	6
Declare-Bankruptcy	24	1	2
Demonstrate	65	9	7
Die	524	57	17
Divorce	20	0	9
Elect	162	5	16
End-Org	31	1	5
End-Position	159	31	22
Execute	14	5	2
Extradite	6	0	1
Fine	22	0	6
Injure	127	14	1
Marry	73	0	10
Meet	200	30	50
Merge-Org	14	0	0
Nominate	11	0	1
Pardon	2	0	0
Phone-Write	112	3	8
Release-Parole	46	0	1
Sentence	84	4	11
Start-Org	29	0	18
Start-Position	92	13	13
Sue	60	12	4
Transfer-Money	127	56	14
Transfer-Ownership	89	5	30
Transport	611	62	48
Trial-Hearing	103	1	5

Table A.1: Event Type distribution across de-facto standard ACE2005 partitions for event detection.