University of Kentucky

UKnowledge

Theses and Dissertations--Computer Science

Computer Science

2018

A recurrent neural network architecture for biomedical event trigger classification

Jeevith Bopaiah
University of Kentucky, jeevith.bopaiah@gmail.com
Author ORCID Identifier:
https://orcid.org/0000-0001-8046-6289
Digital Object Identifier: https://doi.org/10.13023/etd.2018.369

Right click to open a feedback form in a new tab to let us know how this document benefits you.

Recommended Citation

A recurrent neural network architecture for biomedical event trigger classification

---
THESIS

A thesis submitted in partial
fulfillment of the requirements for
the degree of Master of Science in
the College of Engineering at the
University of Kentucky

By
Jeevith Bopaiah
Lexington, Kentucky

Director: Dr. Ramakanth Kavuluru
Lexington, Kentucky 2018

ABSTRACT OF THESIS

A recurrent neural network architecture for biomedical event trigger classification

A "biomedical event" is a broad term used to describe the roles and interactions between entities (such as proteins, genes and cells) in a biological system. The task of biomedical event extraction aims at identifying and extracting these events from unstructured texts. An important component in the early stage of the task is biomedical trigger classification which involves identifying and classifying words/phrases that indicate an event. In this thesis, we present our work on biomedical trigger classification developed using the *multi-level event extraction* dataset. We restrict the scope of our classification to 19 biomedical event types grouped under four broad categories - Anatomical, Molecular, General and Planned. While most of the existing approaches are based on traditional machine learning algorithms which require extensive feature engineering, our model relies on neural networks to implicitly learn important features directly from the text. We use natural language processing techniques to transform the text into vectorized inputs that can be used in a neural network architecture. As per our knowledge, this is the first time neural attention strategies are being explored in the area of biomedical trigger classification. Our best results were obtained from an ensemble of 50 models which produced a micro F-score of 79.82%, an improvement of 1.3% over the previous best score.

KEYWORDS: LSTM, word embeddings, biomedical triggers, attention layer

Author's signature: Jeevith Bopaiah

Date: September 3, 2018

A recurrent neural network architecture for biomedical event trigger classification

By
Jeevith Bopaiah

Director of Thesis: <u>Ramakanth Kavuluru</u>

Director of Graduate Studies: <u>Mirosław Truszczyński</u>

Date: <u>September 3, 2018</u>

Dedicated to my family and friends.

ACKNOWLEDGMENTS

In the last few years, I have met some wonderful people who have had a profound impact on my personal and professional life. My journey as a master's student would not have been the same without these people.

First of all, I would like to express my gratitude to Dr. Ramakanth Kavuluru for his guidance and support throughout the program. His constructive feedbacks on my research work have helped me improve my conceptual understanding and technical writing abilities. He always encouraged me to strive for the best and took an active interest in my job search. I am really fortunate to have had him as the thesis advisor for my master's program.

I am grateful to Dr. Sally Ellingson for giving me the opportunity to work on one of her research projects. This project gave me a platform to hone my data engineering skills. My first work assignment at the university was with the Research Information Services department. A big thank you to William Clark for his support and guidance during the assignment.

My sincere thanks to the committee members Dr. Jerzy Jaromczyk and Dr. Licong Cui for providing valuable feedback which helped me in improving this thesis. A special thanks to Dr. Jerzy Jaromczyk for being instrumental in making this journey a reality. A special mention to my mentor and guide, Vijaykant Nadadur. He has been a role model in many ways and it was his encouragement that brought me here to the master's program.

I would like to thank my friends, Arjun Nagaraj and Sagar Rao for always being there for me when I needed them. Lastly, a big thank you to my parents for their constant love and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1 Introduction

Natural language processing (NLP) is an interdisciplinary area of research that lies at the intersection of linguistics and computer science [1]. On a macro level it attempts to build applications and tools that can understand and interpret language to perform useful tasks. Unlike programming languages and other symbolic languages useful for specialized domain specific communication, natural language involves complex syntactic and semantic structure and a loosely defined grammar to enable creative use of the language. This variability in linguistic structure and flexibility of its associated grammar makes processing natural language more difficult compared to artificial languages. With the proliferation of unstructured content on digital platforms, there is a need for NLP systems that can manage huge volumes of textual data and derive actionable insights from them. Recent advances in computing resources, growing potential of unstructured data and its widespread use in various domains has not only encouraged traditional NLP research but also provided a strong foundation for domain specific NLP systems.

1.1 What is Biomedical Natural Language Processing?

A vast majority of biomedical knowledge ranging from scientific research papers to patient related data are expressed in natural language text. Physicians and other medical researchers read through these verbose textual data to mine important biomedical facts and findings. With the increasing growth in biomedical literature, manual exploration of these articles became a herculean task. The need for an automated or machine assisted processing of the texts was increasingly felt by the biomedical community which led to the development of a subfield in NLP known as biomedical NLP (BioNLP). BioNLP [2, Chapter 2] is the application of NLP techniques to derive meaningful and actionable information from biomedical text such as electronic medical records and biomedical literature. Domain specific applications are somewhat different from traditional language processing tasks. Even though there is a considerable overlap between traditional and biomedical NLP, there are several tasks that are specific to biomedical field which require different strategies. Most of these tasks are driven by the needs of the physicians and medical practitioners who also happen to be the primary users of these systems. Besides the tasks that have end user applications, there also exists certain NLP tasks that form intermediary com-

ponents in a larger system. One such example is word sense disambiguation [3, 4] which resolves ambiguities in the contextual meaning of the word, especially when the same word assumes multiple meanings. The flexibility of a language allows the use of different linguistic expressions to refer to the same biomedical concept. To reduce the variability in expression and provide a universal representation of a concept, concept normalization [2, Chapter 6] task was introduced to map each concept in the text to a unique concept identifier. Named entity recognition [2, Chapter 3] (NER) is also a well known language processing task with an objective to identify entities (such as proteins, genes and cells) in unstructured texts. The entity names/types allow us to generate a higher level and more generalized representation of the text. A natural extension of NER is the extraction of relationships between the entities [2, Chapter 4], [5]. These relations describe the function and behavior of the entities which are of interest to knowledge base curators. This knowledge base of biomedical facts can assist healthcare professionals in making an informed clinical decision on their subjects.

Biomedical event extraction [6, 7] is a more complex type of relation extraction. Unlike relation extraction, whose focus is on binary relations, event extraction encompasses extracting n-ary and even nested relations. Another area that is much appreciated by the medical community is automatic text summarization [2, Chapter 8]. Text summarization aims to extract the important details of a biomedical article or clinical notes of a patient and present it in a concise and coherent manner. Depending on their purpose, the summaries can either be generic or targeted. While a generic summary presents the user with all the relevant facts, a targeted summary contains the details regarding a specific information need. Especially, doctors find this tool useful as the patient information from different sources can be summarized and presented in the form of facts, graphs and charts. Biomedical question answering [2, Chapter 9] task was introduced with an aim to assist researchers and clinical practitioners in their search for specific biomedical facts. It was popularized by the BioASQ Challenge [8] which involved retrieving relevant articles for each question from a corpus of several thousand documents. Subsequently, depending on the specific requirements, the system can be designed to generate either factoid answers, answer snippets, and list (of factoids) or summary type answers. Another important contribution of NLP in healthcare is in clinical decision support (CDS) systems. A CDS system draws the attention of health professionals towards patient specific events and also recommends suitable interventions. NLP techniques are used to process clinical narratives and monitor the likelihood of occurrence of an event. Some examples

include early identification of sepsis patients [9] and detection of post-surgical site infections [10]. The immense potential of NLP in biomedicine comes with its own set of challenges, most important being data annotation. Most of the tasks mentioned here require annotated data to employ supervised statistical NLP techniques. Data annotation is an expensive, time-consuming process and it requires biomedical experts. To make effective use of the limited annotated data and facilitate research in bioNLP, several shared tasks are organized by the biomedical community. These shared tasks have played an important role in advancing the existing techniques and developing new state of the art systems for improving the quality of healthcare.

1.2 Biomedical Event Extraction

Over the last decade, scientific community has witnessed huge investments in biomedical research leading to improved diagnostic and therapeutic procedures. Consequently, this growing impetus in biomedical research has created a wealth of information in biomedical literature hosted in the PubMed search engine by the NIH. Among other research findings and experiments, these scientific articles also contain information on various biological events, which describe the behavior of an entity in the presence of other biomolecular entities or external substances such as chemical compounds. These events help physicians, clinical practitioners, and basic scientists understand the molecular interactions and keep themselves abreast with the new developments in biomedicine. Although scientific literature constitutes a rich source of information on various biological processes and events, its effective utilization is hampered by the need to scour through the entire literature to locate relevant biomedical events. It is estimated that more than 7.3 million journal articles were added to PubMed repository in the period 2004-2013 which was an increase of 48.9% since 2003 [11]. As of now, PubMed hosts over 27 million biomedical citations (titles, abstracts, and other metadata).

Biomedical event extraction (BEE) is the process of extracting evidences of molecular interactions from scientific articles. A biomedical event is comprised of three parts: event trigger word, entities participating in the event, and role of each participating entity. NLP is used to process the text and extract the features that can be used by a machine learning algorithm to identify event structures in a piece of text. BEE was introduced for the first time through the BioNLP-ST'09 [12] community shared task, where the entities participating in the event were restricted to protein and genes. In subsequent years, the scope of event extraction was increased

through shared tasks BioNLP-ST'11 [13] and BioNLP-ST'13 [14] to include chemical and cellular component entities participating in 19 different event types. The availability of annotated data for public use has encouraged many researchers to develop event extraction systems pushing the boundaries of NLP and machine learning techniques. Some of the popular approaches to this task include Bui et al. [15], Björne et al. [6], and Li et al. [16]. Bui et al. [15] used a simple rule based approach to extract events from PubMed abstracts. They used the training data to generate linguistic patterns around trigger words and these patterns were applied on the test set to identify the presence/absence of an event and also to classify the event structure into one of the predefined event types. Björne et al. [6] employed machine learning techniques to train a multi class support vector machine (SVM). The features to the classifier included token features, sentence features, dependency parses, trigger features and external features. The features were converted to a vector of binary values indicating the presence or absence of the feature. Li et al. [16] used a neural network architecture composed of bidirectional long short-term memory (LSTM) units with dynamic extended tree of the sentence as its input. Computational techniques used by the prior works fall into three broad categories — rule based strategies which rely on well defined rules to identify the event patterns, machine learning techniques which involve learning the event patterns through the features provided to the model, and deep neural network architectures which avoid feature engineering and learn to identify the features directly from the text. Due to the complexity of BEE and the number of components (triggers, entities, arguments) involved, prior works can be broadly categorized into pipeline based approaches and joint model approaches. Pipeline based approaches model the individual components in a sequential manner while joint modeling approaches extract the entire event structure in a single input–output pass. Previous studies [17], have found that pipeline based strategies perform better than joint modeling approaches in BEE task. Therefore, in this thesis we focus on the first component of the pipeline, biomedical trigger classification.

1.3 Biomedical Event Trigger Classification

Biomedical triggers are words/phrases (usually a verb or nominalized verb) in a sentence that indicate an event or process occurring at the molecular level as discussed in the sentence. These triggers can be broadly classified into a finite set of event types. For example, triggers such as 'respond', 'associated', 'controlled', 'mediated', can be classified under *"Regulation"* event type. Trigger classification can be defined

4

as identifying and classifying the triggers into their relevant event types. Although there are a wide variety of biological event types occurring at the molecular level, we focus on the most frequently occurring event types as annotated in multi-level event extraction (MLEE) dataset [18].

(a) Trigger word

(b) Non trigger word
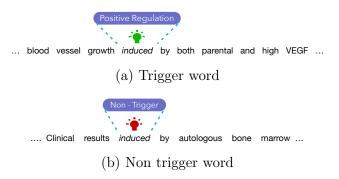
Figure 1.1: Data Samples

Identifying and classifying a trigger is a non-trivial task as the same word/phrase can act as a trigger and as a non-trigger depending on the context in which it is used. As illustrated in Figure 1.1, the word "induced" plays the role of a trigger in Figure 1.1a while it acts as a non-trigger in Figure 1.1b. This shows that contextual information plays a key role in trigger classification task. While the existing methods attribute higher importance to NLP techniques by generating extensive features, our method focuses on learning the right set of features directly from the text. We explore neural architectures that are capable of modeling variable length sequences to capture the contextual information. We present the dataset details in Chapter 3 and our neural model architecture in Chapter 4 and subsequently discuss the benefits of its various components in Chapter 5.

Chapter 2 Related Work

With the growing popularity of pipeline based approaches for biomedical event extraction, several teams are focusing on improving the performance of individual components of the BEE pipeline. Here, we outline the prior efforts related to biomedical trigger classification which are predominantly based on traditional machine learning techniques. Zhou et al. [19] combine word embeddings with syntactic and dependency features using a multiple kernel SVM. Nie et al. [20] avoid feature engineering and employ a multi layer feed forward network whose input is a one-hot encoding of the trigger candidate. Syntactic and semantic information is incorporated by using word embeddings to initialize the hidden layer of feed forward network. Wang et al. [21] use a feed forward network whose input is generated by concatenating word vectors to the left and right of the trigger candidate. The words are represented as low dimensional vectors using dependency based word embeddings. Rahul et al. [22] were the first to exploit sequential information by using LSTMs in a sequence-to-sequence architecture. He et al. [23] followed a two stage approach of trigger identification and classification. Both the stages involved extensive feature engineering followed by SVM classifiers. Jiang et al. [24] used noise contrastive estimation for negative instance filtering and applied a multi-layer perceptron (MLP) network with the trigger candidate as input while ignoring the context information.

It was surprising to note that vanilla MLP network with word embeddings [24] outperformed LSTM based architecture [22] by 5.3% in a task that is heavily dependent on contextual information. On further investigating some of these methods namely, Jiang et al. [24], He et al. [23] and Rahul et al. [22], we found some critical inconsistencies in their implementation and evaluation strategies. A detailed explanation of each one of them is as follows:

- Jiang et al. [24]: To establish some context, PubMed abstracts in the MLEE dataset serve as input and the event file containing trigger spans, trigger words and event types serve as the output. In the preprocessing step, they exploit the prior knowledge of a word/phrase being a trigger and apply differential treatment for triggers and non-triggers. More specifically, they use the word spans of the actual triggers from the event file (output file) to extract all trigger words from the abstract and map them to their corresponding word embeddings. For the remaining non-triggers where the word frequency (in the corpus) is less

than 1000, they replace the word embedding with "UNK" embedding. Since the same word can act as a trigger and non-trigger depending on the context, this can lead to a scenario where the same word has two different embedding if one is a trigger and the other, a non-trigger with frequency less than 1000. This allows the model to clearly distinguish instances when the same word acts as trigger and non-trigger. We have informed the authors regarding our concerns and they have graciously acknowledged the errors and are working towards resolving them.

- He et al. [23]: They use a two-stage approach to trigger classification. The first stage is trigger recognition, a binary classification of triggers and non-triggers. The second stage is event type classification, where the triggers identified in the first stage are classified into one of the 19 event types. In any two-stage classification approach where the input to second stage is derived from first stage, the final recall can never be greater than the recall of first stage. A detailed mathematical proof supporting our claim is provided below.

Proof. Consider a two-stage approach to the event trigger classification task. The first stage is a binary classification problem that aims to classify an instance as trigger or non-trigger. The second stage is a multi-class classification task that aims to classify the triggers (predicted in the first stage) into one of the 19 event types. For the purpose of this claim, we shall focus on the recall metric of the two stages.

In the first stage (binary classification), let TP_b and FN_b be the true positives and false negatives for the binary classification task. The number of actual triggers present in the test set is given by $TP_b + FN_b$. Recall can be expressed in terms of true positives and false negatives as shown in equation 2.1.

$$Recall_b = \frac{TP_b}{TP_b + FN_b} \tag{2.1}$$

Here, FN_b number of actual triggers are predicted as non-triggers and hence, they are not available for the second stage. In the second stage (multi-class classification), the total number of candidates available for classification is the sum of TP_b and FP_b. Out of the total number of candidates available, the number of actual triggers present is TP_b. Since, this is a multi-class classification,

7

we use micro averaged measures for evaluation. The final micro averaged recall
is given by

$$Recall_m = \frac{TP_m^0 + TP_m^1 + ... + TP_m^n}{(TP_m^0 + TP_m^1 + ... + TP_m^n) + (FN_m^0 + FN_m^1 + ...FN_m^n) + FN_b} \quad (2.2)$$

where, n is the total number of classes and m denotes the multi-class setting.
An additional term FN_b in the denominator is carried over from the first stage.
Since the total number of actual triggers available (in second stage) for classi-
fication is TP_b, we can conclude,

$$(TP_m^0 + TP_m^1 + ... + TP_m^n) \leq TP_b$$

Let us assume the best case in which the multi-class classifier got all the triggers
right, i.e., all the triggers were classified into their correct event types. Then
we have,

$$(TP_m^0 + TP_m^1 + ... + TP_m^n) = TP_b \quad \& \quad (FN_m^0 + FN_m^1 + ...FN_m^n) = 0$$

Thus, equation (2.2) reduces to equation (2.1) which implies $Recall_m \leq Recall_b$.
In other words, the final recall is always less than or equal to the recall of the
first stage. □

Contrary to what we showed above, He et al. [23] report the recall of first stage
as 79.05% and that of the final stage as 79.16% which is 0.1% higher than the
first stage. This reveals the inconsistencies in the performance evaluation of
their model. Another important finding that further validates our concerns is
that the precision and recall values listed do not lead to the F-score reported
in their paper [23, 2nd row in Table 1]. We tried contacting them to seek
clarification on our concerns but we did not get any responses.

- Rahul et al. [22]: We illustrate the issues in their approach by describing the
  steps followed during model training process. Their neural network model is
  trained on the training set for a predetermined number of epochs. Each epoch

can be regarded as a snapshot of the model at a different stage in the training process. At the end of each epoch, the snapshot was used to make predictions on the test set. These predictions were compared with the ground truth to obtain the performance of each snapshot and the performance of the best snapshot was reported as their final score. This is equivalent to choosing the best model configuration based on its performance on the test set, which ought to be avoided when comparing performances on a public dataset. In most practical applications, the test set labels are unknown and choosing the best epoch based on the test set would not be possible.

The issues highlighted above give an unfair advantage to these models and result in inflated performance scores. Hence, for fair comparison, we refrain from using their performance scores in the rest of the paper.

While all the prior works exploit linguistic phenomenon and statistical learning techniques to develop the predictive model, most of them do not address the variable nature of the model. While it is true that performance variations are small across multiple runs, the error patterns exhibit large variations. To counter this behavior, we explore ensembling strategies (explained in Section 5.2.1) which improve the performance of the overall system and also ensure its consistency.

Chapter 3 Dataset

We use the MLEE dataset [18] for this task. It consists of 262 PubMed abstracts divided into train and test sets with 175 and 87 abstracts respectively. As this dataset is compiled for the task of event extraction, PubMed abstracts are annotated for event triggers, associated entities and role of the entities in an event. This dataset is limited to 19 types of events spanning four broad categories - Anatomical, Molecular, General and Planned. Table 3.1 shows the distribution of various event types in train and test splits. From the entity information annotated in this dataset, we have 14 types of entities that can be associated with 19 event types. Table 3.2 highlights the various entity types present in the dataset.

Table 3.1: Distribution of Event Types

Category	Event Type	Train	Test
Anatomical	Cell Proliferation	82	43
	Development	202	98
	Blood Vessel Development	540	305
	Growth	107	56
	Death	57	36
	Breakdown	44	23
	Remodeling	22	10
Molecular	Synthesis	13	4
	Gene Expression	210	132
	Transcription	16	7
	Catabolism	20	4
	Phosphorylation	26	3
	Dephosphorylation	2	1
General	Localization	282	133
	Binding	102	56
	Regulation	362	178
	Positive Regulation	654	312
	Negative Regulation	450	233
Planned	Planned Process	407	175

Table 3.2: Distribution of Entity Types

Category	Entity Type	Train	Test
Organism	Organism	485	237
Anatomy	Organism Subdivision	27	22
	Anatomical System	10	8
	Organ	123	53
	Multi-tissue Structure	348	166
	Tissue	304	122
	Cell	866	332
	Cellular Component	105	40
	Developing Anatomical Structure	4	2
	Organism Substance	82	60
	Immaterial Anatomical Entity	11	4
	Pathological Formation	533	357
Molecule	Drug Or Compound	637	307
	Gene Or Gene Product	1961	1001

Although events in general can be represented with a single or multi-word trigger, most of the events in this dataset are characterized by single word triggers and less than 3.0% constitute multi-word triggers. While constructing trigger candidates, we include only single word triggers and treat all multi-word triggers as false negatives. Candidate generation for multi-word triggers is expensive in terms of the number of candidate triggers as it includes all possible consecutive word sequences present in the dataset, which dramatically increases the search space for actual triggers. Also, this large increase in the number of candidates cannot be justified, given the relatively small number of the actual multi-word triggers present in the training data.

Chapter 4 Methods

Event trigger classification consists of two subtasks – trigger detection and trigger type prediction. Trigger detection deals with distinguishing triggers from non-triggers while trigger type prediction predicts the event types for the triggers identified in the first subtask. Instead of approaching these two subtasks individually, we combine them into one task by introducing an additional class for non-triggers. As a result of this modification, we now have a multi-class classification problem with 19 event types and one non-trigger type (which plays the role of trigger detection). Multi-class models can be constructed using two different approaches, namely, extension to binary and transformation to binary classification. In the former, binary classification algorithms are modified to predict more than two classes while in the latter, multi-class classification task is decomposed into multiple binary classification problems and predictions from all the classifiers are used to determine the class of an instance. We adopt the extension to binary classification approach as it avoids a situation where large number of negative instances dominate the final outcomes (as is the case with the second approach). Before discussing specifics of our model, we will present a few fundamental concepts that are essential to develop a better understanding of our architecture.

Neural networks, more specifically, recurrent neural networks (RNNs) and their variants form the core component of our model. RNNs [25] are a class of neural networks known for their ability to capture sequential information from the data. Vanilla RNNs were once a popular choice for several NLP tasks such as language modeling [25, 26, 27], machine translation [28, 29] and sequence labelling [30]. They are composed of basic units called cells that recur over different inputs in the sequence to form a chain like structure. They process the input in a sequential manner, aggregating the past information and using it to inform the current processing step. This allows the RNNs to process variable length sequences which became their strong selling point in NLP tasks. However, it was observed that as the sequence gets longer, RNN's information retaining ability decreases and they are unable to retain clues from the distant past which may be relevant for processing the future time steps. With longer sequences, they also suffer from vanishing/exploding gradients [31]. These issues led to the development of several variants of RNN, a popular one being LSTM. LSTMs [32] are capable of remembering long-term dependencies and are commonly used in place of vanilla RNNs. An LSTM cell consists of three gates which regulate

the flow of information within the network. The *forget gate* decides what elements of the past are not needed and can be forgotten, the *input gate* controls what information of the input should be updated to the cell state, and the *output gate* decides what portion of the cell state should be output at the current time step. LSTMs, feed forward networks [33, Chapter 4] and external features form the basic constituents of our model and they are combined in a unique fashion to arrive at the architecture described in Section 4.2.

4.1 Preprocessing

Since this task is concerned with classifying triggers in biomedical literature, the input is in the form of PubMed abstracts. Along with event type information, these abstracts are also annotated for the entities and their semantic types. We believe that entity types hold important clues to event classification and to leverage this relationship, we replace all the entities in the abstract by their corresponding semantic types (as annotated in the data). Next, we decompose the abstract into sentences and words by using sentence segmenter and word tokenizer respectively. Each tokenized word is considered to be a trigger candidate. By this definition, the number of trigger candidates are ten times more than the actual triggers as observed in the training data. In order to reduce the search space for triggers, we adopt a two stage filtering approach to reduce the non-triggers in the trigger candidate list. In the first stage we completely ignore stop words and entities (i.e., words in entity spans) and treat all other words as triggers candidates. The second stage uses the unified medical language system (UMLS [34]) to eliminate non-trivial, domain neutral non-triggers. Using UMLS semantic types, we retrieve type information for each candidate word in the vocabulary of our corpus. Based on the semantic type information of the actual triggers in the training set (obtained through API calls to NLM's UMLS Metathesaurus server), we have identified 33 semantic types (shown in Table 4.1) as valid types.

If a trigger candidate does not belong to at least one of these valid types, we remove the candidate from the trigger candidate list. The combined two stage filtering process reduces the non-triggers by 53.0% while loosing only 1.0% of the actual triggers as observed on the training data. Table 4.2 shows the actual numbers before and after the filtering process.

Table 4.1: Valid Semantic Types

Biologic Function	Organism Function
Organ or Tissue Function	Cell Function
Molecular Function	Physiologic Function
Genetic Function	Pathologic Function
Disease or Syndrome	Cell or Molecular Dysfunction
Event	Activity
Laboratory Procedure	Diagnostic Procedure
Therapeutic or Preventive Procedure	Research Activity
Molecular Biology Research Technique	Regulatory Activity
Phenomenon or Process	Human-caused Phenomenon or Process
Natural Phenomenon or Process	Functional Concept
Idea or Concept	Temporal Concept
Qualitative Concept	Quantitative Concept
Spatial Concept	Chemical Viewed Functionally
Sign or Symptom	Anatomical Abnormality
Neoplastic Process	Social Behavior
Finding	

Table 4.2: Data summary before and after candidate filtering process

Class	Train		Test	
	Before	After	Before	After
Triggers	3,465	3,437	1,749	1,730
Non Triggers	26,068	12,310	13,544	6,365
Total .	29,533	15,747	15,293	8,095

4.2 Bidirectional LSTM with attention

None of the existing architectures for biomedical trigger classification have explicitly considered the varying degree of influence of the words surrounding the trigger candidate. We believe that differential treatment of contextual words is likely to improve the performance of the model. To test our hypothesis, we propose a novel architecture (shown in Figure 4.1) that uses an attention mechanism to learn the degree of importance of words in the context. It has six components: embedding layer, stacked bidirectional LSTM (BiLSTM) layer, attention layer and three feed forward layers.
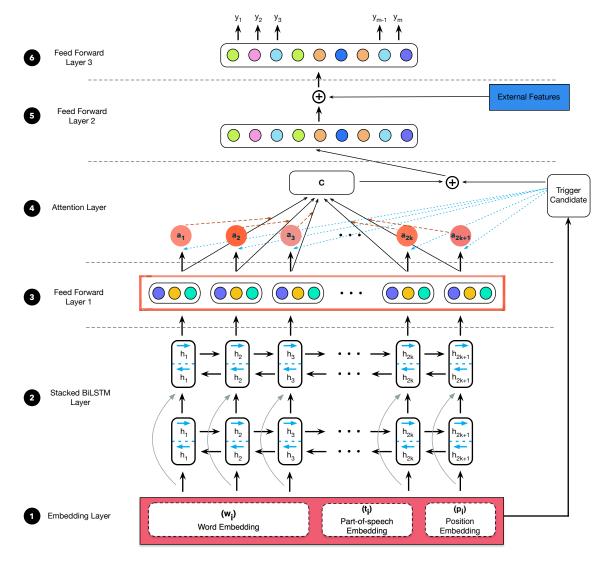
Figure 4.1: Event Trigger Classification Framework

4.2.1 Input embeddings

The input to neural network consists of two parts - X and q, where X is the trigger context and q is the trigger candidate. The trigger context is obtained from the sentence containing the trigger candidate by using a window of k words to the left and right of the trigger candidate. In other words, $X = \{x_{t-k}, ..., x_{t-1}, x_t, x_{t+1}, ..., x_{t+k}\}$, where $x_{t \pm i}$ are k words surrounding the trigger candidate on either side. Each word in the input is represented as e_1 dimensional vector using word embedding table. Word embeddings [35, 36] are pre-trained dense vectors that capture syntactic and semantic meaning of the word. Depending on whether x_i is a word or entity type, the vector representation is obtained from either word embedding table or randomly initialized

15

type embeddings. Similarly, q is also mapped to its corresponding vector in the word embedding table and it is used directly in layer ❹ of Figure 4.1. Along with word embeddings, we also use e_2 dimensional part-of-speech (POS) and e_3 dimensional position embeddings for the trigger context. These embeddings are derived from randomly initialized weights which are updated during the model training process. These three embeddings are concatenated to form each input in sequence X, such that $X \in \mathbb{R}^{(2k+1)\times(e_1+e_2+e_3)}$. In other words, each x_i is constructed as

$$x_i = w_i||t_i||p_i$$

where w_i is the word embedding, t_i and p_i are POS and position embedding respectively and $||$ is the concatenation operator.

4.2.2 Stacked BiLSTM layer

The first component of the input pair – trigger context is passed through a two layered stacked BiLSTM. Essentially, each BiLSTM is composed of two LSTMs, one processing the input from left to right and the other processing the input from right to left. It generates two output vectors for each word and these vectors are concatenated with each other as shown in equations 4.1 – 4.3.

$$\overrightarrow{h_i} = \text{Forward-LSTM}(x_i), \tag{4.1}$$

$$\overleftarrow{h_i} = \text{Backward-LSTM}(x_i), \tag{4.2}$$

$$h_i = \overrightarrow{h_i}||\overleftarrow{h_i}, \text{ for } i = 1, ..., 2k+1 \tag{4.3}$$

where $\overrightarrow{h_i}$ and $\overleftarrow{h_i}$ are the output vectors of forward and backward LSTMs respectively and $||$ is the concatenation operation. The output of first BiLSTM layer with d_1 hidden units is concatenated with input X before feeding it to the second BiLSTM layer with d_2 hidden units. Each output, h_i from the second BiLSTM can be arranged in a matrix H such that $H \in \mathbb{R}^{(2k+1)\times 2d_2}$. Next, the matrix H is passed to a feed forward network in layer ❸ of Figure 4.1, which is composed of l_1 densely interconnected units called neurons. These neurons learn the mapping $s_i = f(h_i W_i + b_i)$, where b_i is the bias unit, s_i is the output, $W_i \in \mathbb{R}^{2d_2 \times l_1}$ is a learnable parameter of the network and f is a rectified linear unit (ReLU) function. Since we have $2k+1$ BiLSTM outputs, we would need $2k+1$ independent feed forward networks. The outputs of all the feed forward networks can be represented as a matrix, $S = [s_1, s_2, s_3, ..., s_{2k+1}]$ such that $S \in \mathbb{R}^{(2k+1)\times l_1}$.

16

4.2.3 Attention layer

In NLP, attention was first used in neural machine translation [37] where the decoder attends to different parts of the source sentence to generate the next target word. Recently, it has been applied to a variety of other NLP tasks such as textual entailment [38], question-answering [39] and text summarization [40]. Depending on the task at hand, a variety of attention strategies are available such as self attention, additive attention and multiplicative attention. Here, we use a multiplicative attention strategy inspired from Luong et al. [41], which involves using the trigger candidate to learn a weight matrix, $W_a \in \mathbb{R}^{e_1 \times l_1}$ and generating a weighted product for each word in the context. Mathematically, this translates to

$$a = softmax(qW_aS^T),$$
$$c = aS$$

where $a \in \mathbb{R}^{1 \times (2k+1)}$ is the attention vector and $c \in \mathbb{R}^{1 \times l_1}$ is the context vector. The context vector is concatenated with the trigger candidate, q and fed to a feed forward network (at layer ❺) with l_2 hidden units. The operations at this layer are governed by $h_\alpha = f([c||q]W_\alpha + b_\alpha)$, where b_α is the bias unit, $h_\alpha \in \mathbb{R}^{1 \times l_2}$ and $W_\alpha \in \mathbb{R}^{(l_1+e_1) \times l_2}$ are the output and weights of the feed forward network and f is a ReLU function.

4.2.4 Feature Engineering

Besides the features learned by the neural network, we also provide explicitly engineered features to the final layer of the network. A brief description of these features are given below:

- For each instance, we obtain the part-of-speech (POS) tag of the trigger candidate and encode it as a r-dimensional one hot feature vector denoted as g_1.

- As already discussed in Chapter 1, an event is described by the entities associated with it. This suggests a correlation between the event and entity types which can be exploited by constructing a table, $M \in \mathbb{R}^{(u+1) \times v}$, where $u = 14$ is the number of entity types and $v = 19$ is the number of event types. The extra row in M corresponds to "no-entity", a situation where an event does not have any entities as arguments. Table M is constructed using the entity-event correspondences present in the training data. Once the table is constructed,

each input instance can be mapped onto a specific row in M by determining the entity type that is closest to the trigger candidate in its three-hop dependency path. Thus, the feature vector for each instance is a v-dimensional row vector denoted as g_2.

- For each instance, we generate z-dimensional dependency encoding of the candidate trigger denoted as g_3. We consider the dependency tags of those relations in which the trigger candidate is either a governor or dependent.

The above features are concatenated with the previous output, h_α and passed into the final layer in Figure 4.1. Softmax activation function is applied over the outputs from the final layer to generate a probability distribution over the event types. The operations at the final layer is given by $p = softmax\big(W_o[g_1||g_2||g_3||h_\alpha] + b_o\big)$, where g_1, g_2, g_3 are the three explicitly engineered feature vectors, $W_o \in \mathbb{R}^{(r+v+z+l_2) \times 20}$ is a learnable network parameter, b_o is the bias unit and $p \in \mathbb{R}^{20}$ is a vector of probability estimates for the event types. Using back propagation [42], the weights of the different network components are updated recursively to minimize the classification errors. This involves minimizing the objective function

$$J(\theta) = -\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{m} y_{ij} \log p_{ij},$$

where θ is a learnable parameter of all the neural network components in our architecture, n and m are the total numbers of training instances and classes respectively, y_{ij} is the ground truth and p_{ij} is the predicted probability of ith input belonging to j^{th} class. The magnitude of errors from the objective function determines the magnitude of updates propagated back to the network. We use optimization technique to minimize the objective function by driving the weight updates in the direction of local or global minima.

Chapter 5 Experiments and Results

We conducted experiments using MLEE dataset to assess the performance of our model. The training data was divided into training set and validation set. The model was trained on the training set and fine tuned on validation set.

5.1 Experiments

After exploring several combination of hyperparameters such as hidden units, dropouts [43], initializations, embeddings and optimizers, we present below the experimental details that produced the best performance.

- **Word Embeddings**: We used 200 dimensional word embeddings published by Pyysalo et al. [44]. The embeddings were trained on PubMed and PubMed Central (PMC) articles using word2vec model. The entity types did not have any pre-trained word embedding and hence, these were randomly initialized with truncated normal distribution with mean of 0.0 and standard deviation of 0.05

- **Input Layer**: The trigger context was constructed by setting the value of $k = 4$. This yields a sequence of 9 words (four words each to the left and right of the trigger candidate and the trigger candidate itself). We used sequence padding for instances going beyond sentence boundaries. Each word in the trigger context is encoded by 3 types of embeddings: $e_1 = 200$ dimensional word embedding concatenated with $e_2 = 30$ dimensional POS embedding and $e_3 = 20$ dimensional position embedding to yield an input word vector of size 250.

- **BiLSTM Layer**: We used a two layer stacked BiLSTMs with $d_1 = d_2 = 512$ hidden units. The weights of each layer were initialized using truncated normal distribution with mean value of 0.0 and standard deviation of 0.05. A non-linear ReLU transformation was applied to the outputs of BiLSTM network. The dropout was set to 0.5 in both layers.

- **Feed Forward Layers**: These layers were initialized with truncated normal distribution with $l_1 = l_2 = 512$ hidden units. Similar to BiLSTMs, non-linearity was induced by applying ReLU transformation and dropout was set to 0.5.

- **Explicit features**: We used 3 feature vectors at the final layer of our network. The POS of the candidate trigger was encoded in a vector of size, r=6, the event type probabilities were encoded in a vector of size, v=19 and the dependency tags of trigger candidate were encoded in a vector of size, z=30. Combining these three features, we obtain a vector of size 55.

- **Output layer**: It consisted of 20 output units corresponding to 20 classes in our task. We omit dropout at this layer and unlike the previous feed forward layers, we used softmax function to generate prediction probabilities.

- **Model Training**: The training data is split into 85% training set and 15% validation set. We followed a batch gradient descent strategy with a batch size of 128. Adam [45] was used for optimization with a learning rate of 0.001. We trained the model on training set for 20 epochs. At the end of each epoch, we tested it on the validation set. The epoch that had the lowest validation loss was used to make predictions on the blind test set.

From the above details, the architectural design and experimental setup might appear too specific to this task. These choices were made after carefully tuning the hyperparameters on a validation set. The hyperparameters such as hidden units, dropouts and number of hidden layers were chosen by gradually increasing their respective values until no further improvements in performance were observed. Thus, after several permutations and combinations of the network components and their values, we arrived at the architecture shown in Figure 4.1

5.2   Results

The performance of our model is evaluated using micro averaged measures of precision, recall and F-score. While computing micro measures, we omit the non-trigger class as it is not an event type and it was added for convenience during the preprocessing step. We shall discuss the merits of our model by comparing it with some of the prior works as shown in Table 5.1. Nie et al. [20] achieve the best performance in this task. Hence, the performance improvements of our model are discussed with respect to their scores. Our best performing model is an ensemble of 50 models and it clocks an improvement of 1.3% (in F-score). Interestingly, it yields an improvement of 9.23% in precision while loosing 7.3% in recall. The benefits of ensembling and its various strategies are discussed in Section 5.2.1

Table 5.1: Comparison of our model against prior works

	Precision	Recall	F-score
Wang et al. [21]	0.7356	0.8362	0.7827
Zhou et al. [19]	0.7556	0.8129	0.7832
Nie et al. [20]	0.7330	**0.8462**	0.7856
Our ensemble	**0.8253**	0.7728	**0.7982**

5.2.1 Model Ensembles

In any parametric learning approach, the final results vary across different runs even when the hyperparameters are kept constant. These variations are caused by random initializations of network weights resulting in different optimization paths leading to different local minima. Owing to the presence of multiple solutions, the error patterns and instances also vary across multiple runs. In order to reduce this variability and improve the consistency of our results, we explore model ensembling [46] which essentially provides different strategies to combine predictions from multiple models creating a synergistic effect. Of the ensembling strategies [47] available, we use bagging approach with model averaging. In model averaging, we average the class probabilities of multiple models and class predictions are based on the averaged probabilities. We generate 50 ensembles by running ten models using different train-validation splits and choosing five best epochs in each model. Each train-validation split is created by randomly choosing 85% as train instance and the remaining 15% as validation instances. We made a conscious choice of retaining five best epochs from each model as we found that error patterns exhibited large variations around the vicinity of local minima.

5.2.2 Ablation Study

In this chapter, we saw the performance gains of our model against prior efforts. Now, we will try to understand the reasons behind these improvements. In ablation study, we remove the various components of the model to interpret the incremental gains in performance. This uncovers the contribution of individual components by comparing the model performance with and without the component. We have identified six such components for ablation experiments as shown in Table 5.2. In all the experiments, we have used individual model instead of the ensemble. Row 1 displays the score of

the individual model with all the components as shown in Figure 4.1. In rows 2 - 7, "−" operator indicates the removal of the component to its right. In row 3, the output of the LSTM is directly used by the attention layer bypassing the feed forward network at layer ❸ in Figure 4.1. In row 4, we remove the feed forward network at layer ❺ of Figure 4.1 and directly pass the concatenated vectors along with external features to layer ❻.

Table 5.2: Component level performance analysis

	Precision	Recall	F-score
Individual model	0.8286	0.7507	0.7877
Individual model − position embedding	0.8346	0.7253	0.7761
Individual model − layer ❸	0.8152	0.7341	0.7725
Individual model − layer ❺	0.7692	0.7518	0.7604
Individual model − external features	0.8171	0.7485	0.7813
Individual model − attention	0.8086	0.7429	0.7744
Individual model − negative filtering	0.7887	0.7739	0.7813

All the performance gains are measured with respect to the F-scores. In row 6, it can be seen that removal of the attention layer decreases the performance by 1.3%. This confirms our conjecture that the words in trigger context exert varying degree of influence on the trigger candidate and learning the degree of influence is imperative to the classification task. The benefit of adding feed forward networks at layers ❸ and ❺ of Figure 4.1 can be noticed through the performance improvement of 1.5% and 2.7% respectively. Even though position embeddings might seem trivial in this task, it contributes to a modest increase of 1.2%. Negative filtering generates an improvement of 0.6%, indicating a promising avenue for further exploration. Ensembling is also one of the contributors to the performance (refer Table 5.1), with an improvement of 1.0% over the individual model.

5.3   Analysis

On examining the instances where the model failed to predict the correct type, we found that a majority of the errors were due to actual triggers being classified as non-triggers or vice versa with very few inter-event type errors. More specifically, around 90.5% of the total errors were due to trigger/non-trigger type misclassification and the

remaining 9.5% were due to inter-event type errors. Here, we will focus our analysis on the former type of errors. Among these errors, false negatives (triggers being classified as non-trigger) were higher than false positives (non-triggers being classified as triggers). On further investigating each of these misclassified trigger candidates and their context, we discovered that these errors could be broadly grouped into two buckets as shown in Figure 5.1.

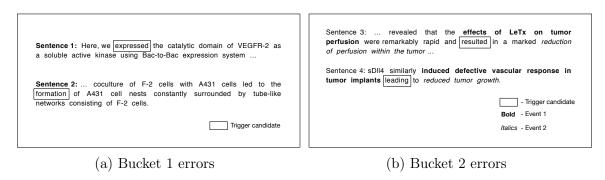(a) Bucket 1 errors (b) Bucket 2 errors

Figure 5.1: Error patterns

The first bucket (Figure 5.1a) contains errors due to incorrect ground truth annotation. For example, consider Sentence 1, the word "expressed" is used to describe the experiment performed on VEGFR-2 and it does not describe an entity/event regulating another entity/event. In Sentence 2, we notice that the word "formation" describes a "development" event. However, it was not annotated as a trigger word in the MLEE corpus. The second bucket (Figure 5.1b) corresponds to those candidates with complex event structure spanning across words/phrases far off from the trigger candidate. Sentence 3 is an example of nested events where the trigger word "resulted" interconnects two independent events, event 1 and event 2 (as shown in Figure 5.1b). Similarly, in Sentence 4, the trigger word "leading" interconnects an event on its left to another event on its right. These errors can be explained by revisiting the trigger context which is restricted to a window of k words on either side of the trigger candidate. Nested events, more often extend beyond this context window and hence they are not available to our model. It would appear that the most obvious solution is to consider the entire sentence as the trigger context. However, when the entire sentence was used as the context, the model failed to capture the nuances in the immediate vicinity of the trigger candidate and it resulted in far more errors than using a context window approach. Therefore, exploring various strategies to acknowledge the presence of other events around the trigger candidate could potentially improve the performance.

Chapter 6 Reproducibility

There has been a growing concern over the reproducibility aspects of scientific research. In particular, this concern is more relevant to the fields of natural language processing, machine learning and deep learning where the number of system variables is large and only the right combination of these variables can reproduce the desired result. In order to facilitate reproducibility of our methods and allow the biomedical community to utilize and extend our system, we release all our project related files online[1]. In this chapter, we will discuss the finer details of implementing the system described in Chapter 4. This chapter contains references to various project related scripts and functions (available on github[1]) and it is advised to refer to these files to develop a better understanding of our system. From a software engineering perspective, the different modules of the system are laid out as shown in Figure 6.1.
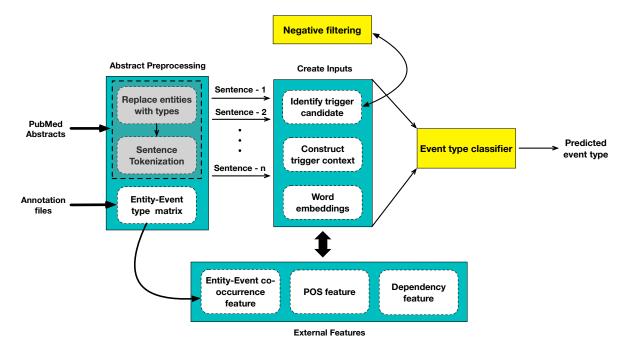
Figure 6.1: System Architecture

Before we begin with the discussion, let us reiterate the goal of this task. Given a PubMed abstract, extract and classify the biomedical event triggers into their biomedical event types.

---

[1] https://github.com/bionlproc/biomedical-trigger-classification

6.1 Abstract Preprocessing

The dataset consists of three types of files - the PubMed abstract, event annotation file and entity annotation file. The event annotation file provides the trigger spans, trigger words and event type information while the entity annotation file provides entity spans and entity type information. In this section, we discuss the classes and methods defined in *preprocess_ abstracts.py* that reads the abstract, replaces the entities with their corresponding types and divides the abstract into individual sentences. The *process_ abstracts* method iterates over each abstract and its corresponding annotation files. It invokes the *extract_triggers_entities* function that extracts the location and type of the entities/trigger words from the annotation files. The location and entity type information are used by *replace_ entities* function to locate the entities in the abstract and replace them with their corresponding entity types. The location and event type of the triggers are used to locate the triggers in the abstract and label them with their corresponding event types. This act of labeling the triggers becomes useful at the time of constructing the inputs, for generating the target class of triggers. Next, the abstract is passed to a sentence tokenizer, which uses regular expressions to determine the end of a sentence. Regular expressions are also used to get rid of non-alphanumerical tokens as they contribute to noise in the trigger classification task.

As we process the abstracts and their annotation files, we also collect information on the entity and event types co-occurring with each other. This information is aggregated across all the abstracts in the training set and is stored as a co-occurrence matrix. At a later stage, this matrix is used to construct one of the external feature vectors that encodes the likelihood of an entity type occurring with different event types.

6.2 Negative Filtering

Negative filtering helps in eliminating some straightforward non-triggers and reducing the size of the trigger candidate candidate space. We adopt two types of negative filtering - rule based filtering and domain based filtering. Domain based filtering leverages the semantic types in UMLS to determine if a trigger candidate is a non-trigger word. The UMLS REST API is used to extract the semantic types of the actual trigger words in the training set. Treating these semantic types as **valid types**, *negative_filtering.py* script is used to determine if the trigger candidate belongs to one of the valid types. The script consists of several functions that perform a sequence of

REST API calls to query the semantic type of each trigger candidate. As the UMLS semantic types are mapped to concept unique identifiers (CUI) and not directly to words, we use *iterate_search* function to retrieve the concept unique identifier (CUI) for the trigger candidate. The CUI is passed as input to *iterate_cui* function which returns the semantic type of the trigger candidate. If the semantic type matches with any of the valid types, it is retained as a valid candidate, otherwise it is excluded from the trigger candidate list.

6.3   External Features

Three types of external features are provided to the output layer of our classification system. These features vary from one trigger candidate to another and they are extracted from the sentence containing the trigger candidate. Each feature is represented as a vector and the Python code to generate these vectors can be found in *external_features.py*. The script is organized as two functions described below.

- *dependency_pos_feature*: This function extracts two linguistic features from the sentence. The first one is known as the dependency feature, which captures the syntactic relationship between the trigger candidate and other words in the sentence while the second feature captures the POS tag of the trigger candidate. Before extracting these features, we first construct the parse tree of the sentence using the Brown BLLIP parser. The parse tree is used to identify those relations where the trigger candidate participates either as a governor or dependent. These relations are encoded as a feature vector of 30 dimensions, each representing one of the 30 predefined dependency relations. This 30-dimensional vector forms the dependency feature of the trigger candidate. All elements of the feature vector are zero except for those which represent the dependency relation of the trigger candidate. For the second feature vector, we extract the part-of-speech information of the trigger candidate and encode it as a six-dimensional one hot vector, each representing one of the six POS tags.

- *entity_event_mapping*: This function leverages the entity type around the trigger candidate to generate a feature vector that represents the likelihood of a particular entity type co-occurring with the event types. Here, the first step is to explore the dependency paths and identify the entity type closest to the trigger candidate. We construct all possible dependency paths that pass through the trigger candidate. These paths are truncated to include up to three de-

pendency hops. Next, we iterate over all the truncated paths to determine the entity type closest to trigger candidate. From the co-occurrence matrix constructed in Section 6.1, we retrieve the row vector corresponding to the entity type closest to the trigger candidate.

6.4 Input Data

The input to the classifier consists of a trigger candidate and the context in which it appears. These two inputs are constructed with the help of functions defined in *construct_input.py* file. Earlier in Section 6.1, we saw that the abstracts are converted into a list of sentences. Here, these sentences are passed through a word tokenizer that splits each sentence into its constituent words. Next, we utilize the negative filtering component to determine if a word should be considered as a valid trigger candidate. For all valid trigger candidates, we create their context by extracting four words to the left and four words to the right of trigger candidate. Part-of-speech tagger is used to extract the POS tags of all words in the context. We also store the position of each word with respect to the trigger candidate. Mathematically, the words in the context are represented by three vector embeddings - word embedding derived from the word itself, POS embedding derived from POS tags and position embedding derived from position indices. Since we replace the entities in the abstract by their types, some context words may also be entity types. The entity types have no predefined word embeddings and hence they are represented by vectors with random values drawn from a uniform distribution. Additionally, we use the external features component to obtain three feature vectors for every input instance. Finally, we represent the ground truth label for each valid candidate as a one hot vector of 20 dimensions, each representing one of the 20 classes. Depending on the class to which the input belongs, all elements of the vector are zero except one which has the value of one.

6.5 Classifier

The neural network components of the classifier are implemented using the functions defined in *neural_network_model.py*. A brief description of these functions are as follows.

- *embedding_layer*: This function takes a scalar input and maps it to a vector. The part-of-speech and position indices are mapped to 30 and 20 dimensional

27

vectors respectively. These vectors are updated along with other parameters of the network.

- *stacked_lstm_layer*: This function creates two layered bi-directional LSTMs with 512 hidden units per layer. The input to the first layer is a concatenation of word embeddings, POS embeddings and position embeddings while the input to the second layer includes the output from the first layer along with the initial embeddings.

- *attention_layer*: This functions reduce the dimension of the BiLSTM output vectors (obtained from previous function) and uses the trigger candidate to attend to specific parts of the context words. The vectors from the previous function are passed to a feed forward network with 512 hidden units to obtain context word vectors of a smaller dimension. The trigger candidate attends to these newly created context word vectors to obtain the attention weights. The attention weights are applied to these context word vectors to obtain a single vectoral representation of the entire context.

- *neural_network*: This function organizes the components described above to create the classifier shown in Figure 4.1.

- *construct_model*: The input vectors and their dimensions are defined in this function. The position and part-of-speech embeddings are also created here by invoking the *embedding_layer* function. The type of optimizer and loss function to be used in the training process are also instantiated in this function.

6.6 Training

In this step, we divide the training instances into two parts - 85% training subset and 15% validation set. The inputs from the training subset are passed to our classifier in batches of 128 instances and the parameters of the network are updated once for every batch. When the model has seen all instances from the training subset, we say that the model has been trained for one epoch and the trained model is referred to as model snapshot. We continue training the model for 20 such epochs. At the end of each epoch, we evaluate the performance of the model snapshot using the validation set. The snapshot with the lowest validation error is used to classify the triggers in the test set. While working with model ensembles, instead of selecting the snapshot with lowest validation error, we select top five (out of 20) snapshots with

least validation errors. We train 10 such independent models by shuffling training and validation instances. By following this process, we obtain 50 model ensembles (5 best snapshots × 10 models). The source code for training the ensemble model can be found in *train_ model_ ensemble.py*

6.7   Testing

Once we have the models trained on the training subset, we use *test_ model.py* to evaluate the performance of the classifier on unseen test samples. We provide these samples to the 50 trained models which yield 50 probability estimates of each sample belonging to 20 classes. We average these individual probability outcomes and predict the class that has the highest probability score for that particular test instance. In the *evaluate* function, the predictions and ground truth labels are compared with each other to count the number of true positives, false positives and false negatives for each of the 19 event types. These counts are used in computing the final performance of the system which is given by micro-averaged precision, recall and F-score.

Chapter 7 Conclusion

In this thesis, we developed a predictive model to identify and classify the trigger words into biomedical event types. We explored deep neural network architectures and proposed an attention mechanism that learns to attribute varying degrees of importance to words in the context. We also saw the benefits of adding a minimal yet effective set of features at the final layer of the network. Domain based candidate filtering plays an important role in reducing non-triggers and its contribution to the overall performance cannot be overlooked. The novelty of our architecture lies in the way different components (attention layer, stacked BiLSTMs, feed forward layers) interact with each other to produce a competent model.

7.1 Limitations

Even though our model performs better than the existing methods in terms of F-score, it has the following limitations.

- Our model does not support the classification of multi-word triggers. We have intentionally restricted it to classify triggers at the word level due to the lack of a sizable portion of phrase triggers of length > 1.

- The results of our model highlight an opposite trend in precision-recall measure. We gain 9.23% in precision and loose 7.3% in recall with a net gain of 2 points in the overall trade off. Because of the gain in precision and loss in recall, this system might appear as a product of a straightforward precision-recall trade off. A higher net gain would have been more convincing of the performance of our system.

- The idea of using fixed length context prevents the system from looking at clues beyond the context window. This is especially true for nested events where the clues are interspersed far away from the trigger word.

7.2 Future Work

Some of the future research directions in the area of biomedical event trigger classification are as follows

- Exploring distance supervision techniques to include more trigger words and phrases is likely to improve the performance of the existing system. Extending our model to support multi-word triggers can be beneficial if it can be done without compromising a lot on precision.

- From the results, it is clear that our model produces a higher precision than recall, reversing the existing precision-recall trend seen in prior methods that depend on feature engineering. Expanding on our manually engineered feature set and exploring a hybrid combination of conventional learning methods with our deep learning system could yield positive results.

- While analyzing errors, we noticed that the model failed on those test instances where the entity clues were far apart from the trigger word. Instead of using fixed length context, forming a dynamic context guided by syntactic structure and domain level clues could help in reducing these errors.

- Our UMLS guided filtering mechanism is based on the individual word and does not take into account the context in which it is used. Exploring mechanisms to consider the contextual meaning of the word while filtering the trigger candidates could eliminate far more non-triggers leading to fewer false positives.

Bibliography

[1] P. M. Nadkarni, L. Ohno-Machado, W. W. Chapman, Natural language processing: an introduction, Journal of the American Medical Informatics Association 18 (5) (2011) 544–551.

[2] K. B. Cohen, D. Demner-Fushman, Biomedical natural language processing, Vol. 11, John Benjamins Publishing Company, 2014.

[3] G. K. Savova, A. R. Coden, I. L. Sominsky, R. Johnson, P. V. Ogren, P. C. De Groen, C. G. Chute, Word sense disambiguation across two domains: biomedical literature and clinical notes, Journal of biomedical informatics 41 (6) (2008) 1088–1100.

[4] A. Sabbir, A. Jimeno-Yepes, R. Kavuluru, Knowledge-based biomedical word sense disambiguation with neural concept embeddings, in: Bioinformatics and Bioengineering (BIBE), 2017 IEEE 17th International Conference on, IEEE, 2017, pp. 163–170.

[5] A. Rios, R. Kavuluru, Z. Lu, Generalizing biomedical relation classification with neural adversarial domain adaptation, Bioinformatics 1 (2018) 9.

[6] J. Björne, T. Salakoski, Generalizing biomedical event extraction, in: Proceedings of the BioNLP Shared Task 2011 Workshop, Association for Computational Linguistics, 2011, pp. 183–191.

[7] J. Björne, F. Ginter, S. Pyysalo, J. Tsujii, T. Salakoski, Scaling up biomedical event extraction to the entire pubmed, in: Proceedings of the 2010 workshop on biomedical natural language processing, Association for Computational Linguistics, 2010, pp. 28–36.

[8] G. Tsatsaronis, M. Schroeder, G. Paliouras, Y. Almirantis, I. Androutsopoulos, E. Gaussier, P. Gallinari, T. Artieres, M. R. Alvers, M. Zschunke, et al., Bioasq: A challenge on large-scale biomedical semantic indexing and question answering., in: AAAI fall symposium: Information retrieval and knowledge discovery in biomedical text, 2012.

[9] E. Apostolova, T. Velez, Toward automated early sepsis alerting: Identifying infection patients from nursing notes, BioNLP 2017 (2017) 257–262.

[10] Z. Hu, G. J. Simon, E. G. Arsoniadis, Y. Wang, M. R. Kwaan, G. B. Melton, Automated detection of postoperative surgical site infections using supervised methods with electronic health record data, Studies in health technology and informatics 216 (2015) 706.

[11] K. Z. Vardakas, G. Tsopanakis, A. Poulopoulou, M. E. Falagas, An analysis of factors contributing to pubmed's growth, Journal of Informetrics 9 (3) (2015) 592–617.

[12] J.-D. Kim, T. Ohta, S. Pyysalo, Y. Kano, J. Tsujii, Overview of bionlp'09 shared task on event extraction, in: Proceedings of the Workshop on Current Trends in Biomedical Natural Language Processing: Shared Task, Association for Computational Linguistics, 2009, pp. 1–9.

[13] J.-D. Kim, S. Pyysalo, T. Ohta, R. Bossy, N. Nguyen, J. Tsujii, Overview of bionlp shared task 2011, in: Proceedings of the BioNLP shared task 2011 workshop, Association for Computational Linguistics, 2011, pp. 1–6.

[14] C. Nédellec, R. Bossy, J.-D. Kim, J.-J. Kim, T. Ohta, S. Pyysalo, P. Zweigenbaum, Overview of bionlp shared task 2013, in: Proceedings of the BioNLP Shared Task 2013 Workshop, 2013, pp. 1–7.

[15] Q.-C. Bui, D. Campos, E. van Mulligen, J. Kors, A fast rule-based approach for biomedical event extraction, in: proceedings of the BioNLP shared task 2013 workshop, 2013, pp. 104–108.

[16] L. Li, J. Zheng, J. Wan, D. Huang, X. Lin, Biomedical event extraction via long short term memory networks along dynamic extended tree, in: Bioinformatics and Biomedicine (BIBM), 2016 IEEE International Conference on, IEEE, 2016, pp. 739–742.

[17] M. Miwa, P. Thompson, I. Korkontzelos, S. Ananiadou, Comparable study of event extraction in newswire and biomedical domains, in: Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers, 2014, pp. 2270–2279.

[18] S. Pyysalo, T. Ohta, M. Miwa, H.-C. Cho, J. Tsujii, S. Ananiadou, Event extraction across multiple levels of biological organization, Bioinformatics 28 (18) (2012) i575–i581.

[19] D. Zhou, D. Zhong, Y. He, Event trigger identification for biomedical events extraction using domain knowledge, Bioinformatics 30 (11) (2014) 1587–1594.

[20] Y. Nie, W. Rong, Y. Zhang, Y. Ouyang, Z. Xiong, Embedding assisted prediction architecture for event trigger identification, Journal of bioinformatics and computational biology 13 (03) (2015) 1541001.

[21] J. Wang, J. Zhang, Y. An, H. Lin, Z. Yang, Y. Zhang, Y. Sun, Biomedical event trigger detection by dependency-based word embedding, BMC medical genomics 9 (2) (2016) 45.

[22] P. V. Rahul, S. K. Sahu, A. Anand, Biomedical event trigger identification using bidirectional recurrent neural network based models, BioNLP 2017 (2017) 316.

[23] X. He, L. Li, Y. Liu, X. Yu, J. Meng, A two-stage biomedical event trigger detection method integrating feature selection and word embeddings, IEEE/ACM transactions on computational biology and bioinformatics.

[24] N. Jiang, W. Rong, Y. Nie, Y.-K. Shen, Z. Xiong, Biological event trigger identification with noise contrastive estimation, IEEE/ACM Transactions on Computational Biology and Bioinformatics.

[25] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, S. Khudanpur, Recurrent neural network based language model, in: Eleventh Annual Conference of the International Speech Communication Association, 2010.

[26] T. Mikolov, S. Kombrink, L. Burget, J. Černockỳ, S. Khudanpur, Extensions of recurrent neural network language model, in: Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on, IEEE, 2011, pp. 5528–5531.

[27] A. Graves, Generating sequences with recurrent neural networks, arXiv preprint arXiv:1308.0850.

[28] S. Liu, N. Yang, M. Li, M. Zhou, A recursive recurrent neural network for statistical machine translation, in: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Vol. 1, 2014, pp. 1491–1500.

[29] M. Auli, M. Galley, C. Quirk, G. Zweig, Joint language and translation modeling with recurrent neural networks, in: Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, 2013, pp. 1044–1054.

[30] J. A. Perez-Ortiz, M. L. Forcada, Part-of-speech tagging with recurrent neural networks, in: Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on, Vol. 3, IEEE, 2001, pp. 1588–1592.

[31] R. Pascanu, T. Mikolov, Y. Bengio, On the difficulty of training recurrent neural networks., Proceedings of the 30th International Conference on Machine Learning 28 (2013) 1310–1318.

[32] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural computation 9 (8) (1997) 1735–1780.

[33] C. M. Bishop, Neural networks for pattern recognition, Oxford university press, 1995.

[34] National Library of Medicine, Unified Medical Language System Reference Manual, http://www.ncbi.nlm.nih.gov/books/NBK9676/ (2009).

[35] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, J. Dean, Distributed representations of words and phrases and their compositionality, in: Advances in Neural Information Processing Systems, 2013, pp. 3111–3119.

[36] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, arXiv preprint arXiv:1301.3781.

[37] D. Bahdanau, K. Cho, Y. Bengio, Neural machine translation by jointly learning to align and translate, International Conference on Learning Representations.

[38] A. Parikh, O. Täckström, D. Das, J. Uszkoreit, A decomposable attention model for natural language inference, in: Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, 2016, pp. 2249–2255.

[39] K. Chen, J. Wang, L.-C. Chen, H. Gao, W. Xu, R. Nevatia, Abc-cnn: An attention based convolutional neural network for visual question answering, arXiv preprint arXiv:1511.05960.

[40] A. M. Rush, S. Chopra, J. Weston, A neural attention model for abstractive sentence summarization, in: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, 2015, pp. 379–389.

[41] T. Luong, H. Pham, C. D. Manning, Effective approaches to attention-based neural machine translation, in: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, 2015, pp. 1412–1421.

[42] R. Hecht-Nielsen, Theory of the backpropagation neural network, in: Neural networks for perception, Elsevier, 1992, pp. 65–93.

[43] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, The Journal of Machine Learning Research 15 (1) (2014) 1929–1958.

[44] S. Pyysalo, F. Ginter, H. Moen, T. Salakoski, S. Ananiadou, Distributional semantics resources for biomedical text processing, Proceedings of 5th International Symposium on Languages in Biology and Medicine (2013) 39–44.

[45] D. Kinga, J. B. Adam, Adam: A method for stochastic optimization, in: International Conference on Learning Representations (ICLR), 2015.

[46] D. Abbott, A. Analytics, Benefits of creating ensembles of classifiers.

[47] D. Opitz, R. Maclin, Popular ensemble methods: An empirical study, Journal of artificial intelligence research 11 (1999) 169–198.

Vita

Name: Jeevith Bopaiah

Education

Bachelor of Engineering in Computer Science,   August 2010 - May 2014
Nitte Meenakshi Institute of Technology, India

Experience

- *Research Intern*, **Oak Ridge National Lab**, Tennessee   Jun 2017 - Jul 2017

- *Full Stack Developer*, **Stride.ai Inc**, India   Nov 2015 - Jun 2016

- *Software Engineer*, **Tesco HSC**, India   Jul 2014 - Oct 2015

- *Tech Intern*, **Tationem**, India   Jan 2014 - May 2014

Awards

- Best Graduate Student (MS) in Computer Science (2018)

Publications

1. Derek Jones, **Jeevith Bopaiah**, Fatemah Alghamedy, Nathan Jacobs, Heidi L. Weiss, W.A. de Jong, Sally R. Ellingson. *Polypharmacology Within the Full Kinome: a Machine Learning Approach.* In AMIA 2018 Informatics Summit

2. Fatemah Alghamedy, **Jeevith Bopaiah**, Derek Jones, Xiaofei Zhang, Heidi L. Weiss, Sally R. Ellingson. *Incorporating Protein Dynamics Through Ensemble Docking in Machine Learning Models to Predict Drug Binding.* In AMIA 2018 Informatics Summit

3. **Jeevith Bopaiah** and Ramakanth Kavuluru. *Precision/Recall Trade-Off Analysis in Abnormal/Normal Heart Sound Classification*, Proceedings of the 5th International Conference on Big Data Analytics (BDA 2017), pp. 179–194.

4. Michael Bowie, Edmon Begoli, Byung Park and **Jeevith Bopaiah**. [2017, October] *Towards the LSTM-based Approach for Detection of Temporally Anomalous Data in Medical Datasets.* In 2017 22nd MIT International Conference on Information Quality (ICIQ), Little Rock, Arkansas, USA.