INDIAN STATISTICAL INSTITUTE
KOLKATA

MTECH FINAL THESIS

Event Extraction from Bio-Medical Documents

Author:
Krishanu NAYAK
Roll : MTCS-1515

Supervisor:
Prof. Utpal GARAIN
CVPR Unit, ISI Kolkata

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Technology (Computer Science)

in the

Computer Vision and Pattern Recognition Unit
Indian Statistical Institute, Kolkata

Dedicated to my family and friends

CERTIFICATE

This is to certify that the dissertation entitled **"Event Extraction from Bio-Medical Documents"** submitted by **Krishanu Nayak (Roll : MTCS-1515)** to Indian Statistical Institute, Kolkata, in partial fulfillment for the award of the degree of **Masters of Technology in Computer Science** is a bona-fide record of work carried out by him under my supervision and guidance. The dissertation has fulfilled all the requirements as per the regulations of this institute and, in my opinion, has reached the standard needed for submission.

Prof. Utpal Garain
Computer Vision & Pattern Recognition Unit,
Indian Statistical Institute,
Kolkata - 700108

Acknowledgements

First and foremost, I would like to convey my gratitude to Prof. Utpal Garain, who gave me an opportunity to complete my MTech Final Year Thesis entitled **Event Extraction from Bio-Medical Documents**, under his supervision at a very crucial moment of my career. Without his guidance, inspiration and motivation it would not have been possible to maintain the perseverance towards the project.

As this has been an amazingly new experience and my very first probable contribution to research works, I would like to dedicate this to my family, without whom I would have been lost in every part of life.

Finally, I would like to thank Avishek da, Akshay and Onkar, the invaluable trio of the NLP Lab of ISI Kolkata, who believed in me when no one else did, who constantly motivated me, and whose inputs greatly improved the outcomes of this project.

Abstract

With the ever increasing quantity of scientific literature in the Bio-medical domain, it has become a crucial task to build such intelligent automated systems, that can understand the deep hidden meaning underlying texts, and can adapt to new knowledge based discoveries in the literature. Natural Language Processing is the best approach towards this task. In this project, we try to build such a system to extract knowledge from scholary articles of Cancer Genetics for the Event Extraction shared task organized by the Bio-NLP community in 2013. With the help of distributed vector representations of words to capture the semantic informations of words, and the Recurrent Neural Networks to capture the contextual informations of sequential data, we propose a pipelined system of two models for the Cancer Genetics Event Extraction task.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Text Mining (or Text Data Mining, or Textual Analytics, or Information Extraction from Texts) is the process of automatically extracting high-quality, structured information from Natural Language texts, by the means of various Pattern Recognition techniques, to describe the content, structure and behavior underlying the texts.

Over the years various techniques such as Bayes, Naive Bayes, Nearest Neighbors, Support Vector Machines (SVM), Maximum Entropy, Expectation Maximization (EM), Hidden Markov Model (HMM), etc., have been used successfully for such information extraction tasks, because of their simplicity and the simple feature representation for the texts. Although neural network models were already present at the time, they were easily superseded by the aforementioned techniques, because neural nets need powerful computation devices, and some form of numerical representation of the texts.

As the technology has evolved and computation devices such as Processors and Graphical Processing Units (GPUs) became powerful, application of neural nets resurfaced. Along with that, the distributed vector representation of texts that can capture syntactic and semantic information inherent in them, helped the neural nets to improve information extraction tasks from texts, profoundly. We have seen the strengths of different complex neural net models (currently known as Deep Neural Nets) such as, Recurrent Neural Networks (RNN), Autoencoders, Restricted Boltzman Machines (RBM), Deep Belief Networks (DBN), Convolution Neural Networks (CNN), etc., for various Natural Language Processing (NLP) related tasks, for example, Sentiment Analysis, Handwritten Digit Recognition, Machine Translation, Language Modeling, Question Answering, Chat-Bots, etc.

As each day hundreds of scholary articles are proposed in the Bio-Medical domain, it is necessary to improve information retrieval techniques so that the search queries can return the most relevant results. One most important step to achieve that is to improve information extraction techniques. Hence our goal is to try to build such an information extraction system using the power of deep neural networks.

Since 2009, the BioNLP community has been organizing Shared Task (ST) events in order to advance the methods and resources to automate the information extraction from biomedical literature. In this thesis, we are going to explore Cancer Genetics (CG) shared task introduced in BioNLP 2013.

1.2 Organization of the Thesis

Chapter 2 is used to present the Cancer Genetics task in detail, starting from the definitions of Cancer Genetics, Entities, Triggers, Arguments and Events, to describing and breaking down the Event Extraction task into components, and finally stating why we need NLP techniques for this task.

Chapter 3 is used to present briefly the techniques that have been used in most approaches, and a state of the art model for the CG shared task in BioNLP ST 2013.

Chapter 4 is used to describe the relevant theoretic backgrounds needed to understand our models, in-depth description of our proposed models, their results and benchmarking scores, and finally reviewing the pros and cons of our models.

Chapter 5, the final chapter, is used to comment over our overall thesis, and future plans for this task.

Chapter 2

Cancer Genetics Task

2.1 Cancer Genetics

Genes are found in DNA in each cell that forms our body. They control how the cell functions, for example, how quickly it grows, how often it divides, and how long it lives. Each gene has its own instruction to formulate correct protein that performs specific functions for the cells. Providing wrongful instructions to genes, the DNA then mutated, creates nonstop harmful abnormal proteins resulting unlimited cell productions, or no protein at all resulting no repair in the cellular damages. These mutations happen often, but the human body can correct them most of the time. Cancer is a genetic disorder which usually happens after multiple such harmful mutations that could not be corrected by the human body. And the study of the cause and effects of such mutations in genes leading to destructive behavior of cells is called Cancer Genetics.

2.2 Cancer Genetics Task from BioNLP 2013 ST

2.2.1 Background

The BioNLP ST, organized for the third time in 2013, presented open challenges as Natural Language Processing tasks over biomedical scientific literature, where the organizers provided task definitions, Gold standard data for model development and evaluation, and tools for model assessment and comparisons. The biological questions addressed by the BioNLP ST 2013 belong to the molecular biology domain and related fields. Similar to the previous two editions, this one is also targeted towards progressing the complex text-bound Event Extraction (EE) tasks. The BioNLP ST '13 includes six tasks:

1. Genia Event Extraction (GE)
2. Cancer Genetics (CG)
3. Pathway Curation (PC)
4. Gene Regulation Ontology (GRO)
5. Gene Regulation Network in Bacteria (GRN)

6. Bacteria Biotopes (BB)

All these tasks share common event-based representation and file formats, so that one system can be reused across tasks easily. For our project we decided to work with Cancer Genetics (CG) task.

2.2.2 Cancer Genetics task

The CG task aims at automatic extraction of information from the literature on Cancer, a complex group of genetic diseases that is well studied research topic worldwide.

> *The CG task concerns the extraction of events relevant to cancer, covering molecular foundations, cellular, tissue, and organ-level effects, and organism-level outcomes.*[Nédellec et al., 2013]

> *The CG task is differentiated from previous event extraction tasks in the BioNLP ST series in addressing a wide range of pathological processes and multiple levels of biological organizations, ranging from the molecular through the cellular and organ levels up to whole organisms*[Pyysalo et al., 2013]

The CG task is defined over a set of 18 entity and 40 event types based on community standard ontologies. The CG task corpus consists of 600 PubMed abstracts annotated for over 17,000 events. The objective is to capture these events and associated arguments from the given CG corpus. These are thoroughly discussed in the following sections.

2.2.3 Definitions

2.2.3.1 Entity

A *Biological Entity*, or simply an *Entity*, can be defined as an independent yet complete representation of an instance of a biological organization (or biological entity), of various hierarchical levels. Each such *entity* participates to form a biological class, that depicts the level of hierarchy it belongs to. For example:

- *Tumor* is an entity of type *Cancer*
- *TIMP-3* is an entity of type *Gene or Gene Product*
- *Capillary* is an entity of type *Tissue*

The 18 types of *entities* defined with reference to the domain standard databases and ontologies for the Cancer Genetics Shared Task, can be seen in Figure 2.1. The grayed out labels inside the 'Type' column represent higher level organizations, presented in the table only for references, but they are not part of the annotations.

2.2.3.2 Event

A *Biological Event*, or simply an *Event*, can be described as a change of state of *Biological Entities*, or some other *Biological Events*. The interacting *entities* and *events* are said to be the arguments of the parent *event*. Furthermore, the type and the number of arguments depend on the parent *event* type. Hence, an *event* is formally constructed by two features:

Type	Scope	Reference
Anatomical entity	structural organization of organism	CARO
Material anatomical entity	anatomical entities (ASs) with mass	CARO
Anatomical structure	material ASs with structure	CARO
Organism	organism mentions	taxonomy DBs
Organism subdivision	fiat parts of multicellular organism	CARO
Anatomical system	ASs of multiple organs	CARO
Organ	ASs of multiple multi-tissue structs.	CARO
Multi-tissue structure	AS of multiple tissues	CARO
Tissue	ASs of similar cells and ECM	CARO
Developing anatomical structure	ASs varying in granularity due to development	CARO
Cell	ASs of cell compartment, surrounded by PM	CL
Cellular component	ASs that are parts of cells	GO-CC
Organism substance	gaseous, liquid or semisolid material ASs	CARO
Immaterial anatomical entity	anatomical entities without mass	CARO
Molecular entity		
Gene or gene product	genes, RNA and proteins	gene/protein DBs
Simple chemical	simple, non-repetitive chemical entities	ChEBI
Protein domain or region	parts of protein molecules	
Amino acid	amino acid residues	ChEBI
DNA domain or region	short, specifically identified spans of DNA	
Pathological formation	pathological material organism parts	
Cancer	cancerous pathological formations	

Figure 2.1: 18 Entity Types as defined in the BioNLP ST '13

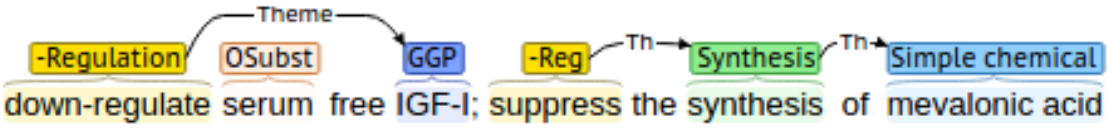
- **Trigger Word(s):** A word, or a sequence of words, that expresses the existence of an *event*, is called a *trigger*. The *trigger* itself decides which type of *event* is happening, and said to belong to that *event type*.

  Let us denote an *event* by superscripted form as $\boldsymbol{TriggerWord}^{EventType}$ and an *entity* by subscripted form as $\boldsymbol{EntityWord}_{EntityType}$.

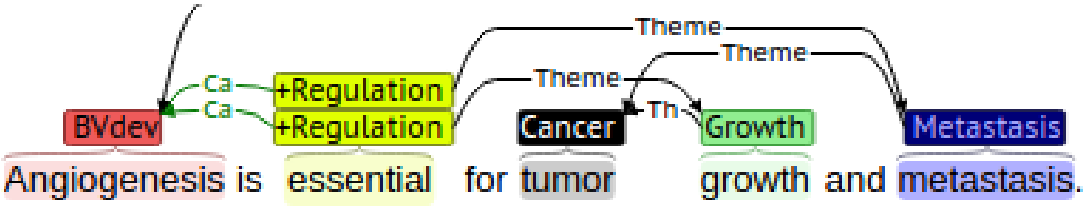
  Figure 2.2 contains multiple such *events*. Let's look at a few examples. In Figure 2.2a, $\boldsymbol{IGF\text{-}I}_{GGP}$ and $\boldsymbol{mevalonic\ acid}_{Simple\ chemical}$ are the *entities*, and $\boldsymbol{down\ regulate}^{-Regulation}$, $\boldsymbol{suppress}^{-Reg}$, and $\boldsymbol{synthesis}^{Synthesis}$ are the *events*.

- **Argument(s):** The *entities* and *events* participating in the *event* being considered, are called its *arguments*. Each of these *arguments* also said to perform different roles in the construction of the *event*, which are defined by the *event type*, and partly the type of the *argument* itself. The number of role types are small and generic, but they are defined some moderate task specific way. The following roles are defined in the CG task.

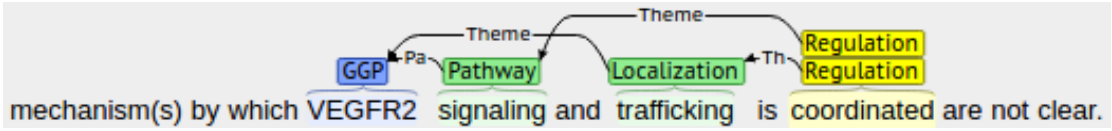
  - **Theme:** An *entity* or an *event* that is primarily affected by the *event*. In Figure 2.2a, an *entity* $\boldsymbol{IGF\text{-}I}_{GGP}$ is an *argument* of type *Theme* of an *event* $\boldsymbol{down\text{-}regulate}^{-Regulation}$, and an *event* $\boldsymbol{synthesis}^{Synthesis}$ is an *argument* of type *Theme* of an another *event* $\boldsymbol{suppress}^{-Reg}$.



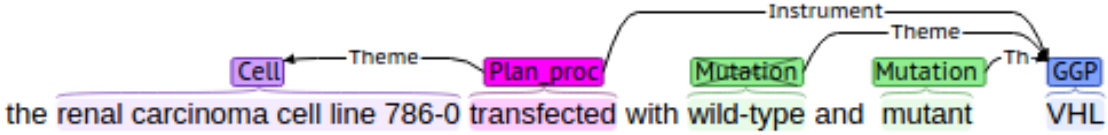
(a) Example for Theme



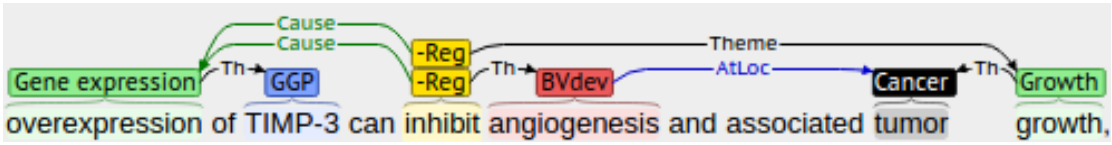
(b) Example for Cause



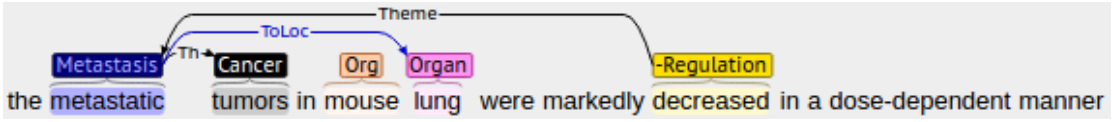
(c) Example for Participant



(d) Example for Instrument



(e) Example for AtLoc



(f) Example for ToLoc

Figure 2.2: Examples of different event compositions

- **Cause:** An *entity* or an *event* which is the reason of the *event*. In Figure 2.2b, an *event* $\boldsymbol{Angiogenesis}^{BVdev}$ is an *argument* of type *Cause* of an *event* $\boldsymbol{essential}^{+Regulation}$.
- **Participant:** An *entity* that is participating in the *event*, but its role can not be described from the context. In Figure 2.2c, an *entity* $\boldsymbol{VEGFR2}_{GGP}$ is an *argument* of type *Participant* of an *event* $\boldsymbol{signalling}^{Pathway}$.
- **Instrument:** An *entity* that is used to carry out the *event*. In Figure 2.2d, an *entity* $\boldsymbol{VHL}_{GGP}$ is an *argument* of type *Instrument* of an *event* $\boldsymbol{transfected}^{Plan\_proc}$.
- **Site:** A part of a *Theme entity* that is specifically affected by the *event*.

- **AtLoc:** Location where the *event* takes place. In Figure 2.2e, an *entity* $\boldsymbol{tumor}_{Cancer}$ is an *argument* of type *AtLoc* of an *event* $\boldsymbol{angiogenesis}^{BVdev}$.
- **FromLoc:** The source of a movement is described by an *event* involving a change of location.
- **ToLoc:** Direction or end-point of a movement described by an *event* involving a change of location. In Figure 2.2f, an *entity* $\boldsymbol{lung}_{Organ}$ is an *argument* of type *ToLoc* of an *event* $\boldsymbol{metastatic}^{Metastasis}$.

2.3 Task Breakdown

The Event Extraction task can be partitioned into two consecutive subtasks:

1. **Trigger Words Detection:** The first part of the two deals with finding the trigger words from a given text. This also involves finding the associated event types for those trigger words. This essentially is a *sequence labeling* problem.

2. **Arguments Extraction:** The next part is to extract the arguments (entities and events), in association with the type of the arguments (theme, cause etc.), for the trigger words identified in the first part.

For example, in Figure 2.2f, the first task is to identify the trigger words **metastactic** and **decreased**, with their associated event types *Metastasis* and *Negative Regulation*, respectively. The next part is to identify arguments of $\boldsymbol{metastactic}^{Metastasis}$, which are entities $\boldsymbol{tumors}_{Cancer}$ of argument type *Theme* and $\boldsymbol{lung}_{Organ}$ of argument type *ToLoc*; and the argument of $\boldsymbol{decreased}^{NegativeRegulation}$, which is an event $\boldsymbol{metastatic}^{Metastasis}$ of argument type *theme*. Then the compositions of events become:

Event 1: $\boldsymbol{metastactic}^{Metastasis} \xrightarrow[\text{Theme}]{} \boldsymbol{tumors}_{Cancer}$

Event 2: $\boldsymbol{metastactic}^{Metastasis} \xrightarrow[\text{ToLoc}]{} \boldsymbol{lung}_{Organ}$

Event 3: $\boldsymbol{decreased}^{NegativeRegulation} \xrightarrow[\text{Theme}]{} \boldsymbol{metastatic}^{Metastasis}$

2.4 The Need of NLP Techniques for EE Task

With the existence of millions of proteins, genes, organisms, species, we are nowhere near to understand the relationships or interactions amongst them completely. Even the scientific literatures already available in these domains, can not possibly be comprehended all, to make rules for knowledge based information extraction tasks. Hence, along with knowledge representations (e.g. *named entities*), some generic ways (e.g. different *entity types*, *event types* and *argument roles*) are defined to represent the possible relationships. Therefore, no better method than Natural Language Processing can be thought of for deep, automatic understanding and capturing of those relationships.

2.5 Summary

In this chapter, we covered the BioNLP ST 2013 platform, the in-depth setting of Cancer Genetics (CG) task, and why we have chosen NLP to apprehend the task. In the subsequent chapters we will discuss the related works for the CG task, and present our methods and results for the same.

Chapter 3

Related Works

3.1 Participants

There were six teams, including one from ISI, who took participation in the BioNLP ST 2013 for Cancer Genetics Task. The Turku Event Extraction System (TEES 2.1)[Björne and Salakoski, 2013], presented by University of Turku, which was first proposed in BioNLP ST 2009, and extended in both BioNLP ST 2011 and 2013 to adapt to new corpora and derive task specific event rules. Like 2011, in 2013 also TEES stayed unbeaten, with recall 48.76%, precision 64.17%, and fscore 55.41%. The results for full task (primary evaluation criteria), of top three teams are shown in Table 3.1.

Teams	**Institute**	**Recall**	**Precision**	**F-Score**
TEES-2.1	University of Turku, Finland	48.76	**64.17**	**55.41**
NaCtTeM	University of Manchester, UK	**48.89**	55.82	52.09
NCBI	NCBI, US	38.28	58.84	46.38

Table 3.1: Top three team results in BioNLP ST 2013

3.2 Discussion

3.2.1 Introduction

There were various approaches for the extraction tasks, including Support Vector Machines (SVM) based pipeline architectures, a joint pattern matching approach, a rule based approach, and a parsing based approach. The organizers provided different supporting resources, that is, the training, development and test data were preprocessed by BioC lemmatizer, Genia sentence splitter, Genia Treebank tokenizer, Stanford Parser, etc. The task setting explicitly allowed the use of any external resources. In the following section we will look into the methodologies used by the TEES 2.1 system.

3.2.2 Turku Event Extraction System (TEES) 2.1

TEES[Björne and Salakoski, 2013] is a Support Vector Machine (SVM) based system for the extraction of events and relations from natural language texts. The generalizations

included in the version 2.1 made TEES achieve highest ranks in four out of eight tasks in BioNLP ST 2013.

3.2.2.1 Main Idea behind TEES

The basic goal of TEES is to extract text-bound graphs from Natural Language articles using machine learning techniques. It represents each event in a unified graph format where triggers and entities are nodes, and arguments are labeled directed edges connecting those nodes.

3.2.2.2 Preprocessing

TEES converts the shared task data (*a1, a2 and txt files*) into interaction XML formats. This generalization was required to incorporate various corpora provided for multiple shared tasks in the BioNLP ST 2013. The tokenization was performed by standard tokenizer, parse trees were generated by McCCJ parser, which are then converted into a collapsed Stanford dependency scheme.

3.2.2.3 Event Extraction

TEES follows three primary processing steps to detect events.

1. In the first step, event triggers are identified by classifying each non-entity words into one of the event classes or as a negative.
2. In the second step, for each pair, constructed by taking an entity and a predicted trigger, an argument edge candidate is generated and classified into one of the argument classes or as a negative.
3. In the final unmerging step, for each valid set of outgoing argument edges, an unmerging example is generated and classified as a true event or not, separating overlapping events into structurally valid ones. This unmerging step removes structurally valid but possibly negatives from consideration, increasing support scores.

TEES uses $SVM^{multiclass}$ Support Vector Machine with a linear kernel as the classifier in all machine learning steps.

In the previous versions of TEES, task specific rules were used for event extraction, for example, number of arguments and argument types for each event were manually defined. In TEES 2.1 these rules and constraints are learned automatically. The performance of TEES 2.1 on CG task can be seen in Table 3.1.

3.2.2.4 Shortcomings

As reported in the TEES 2.1 paper, representing the *site* arguments have been problematic, due to the structure of *site* arguments. As the *site* arguments are part of the primary arguments (*themes*), TEES requires another step after the predictions of primary arguments. This creates problems when there are multiple such primary arguments. In those cases TEES uses the nearest primary argument to be the *site* of the *event*. This schemes although bypasses the problem, does not resolve the ambiguity.

3.3 Summary

Although it is necessary to describe the investigations of all the other related works in the context of event extraction, it is beyond the scope of this thesis to present them here, due to the paucity of space, and the distinctive nature of the methods we are going to propose. Hence, we presented in a very concise manner, the Turku Event Extraction System, its methodologies, tools and techniques, and its pitfalls.

Chapter 4

Proposed Work

4.1 Theoretic Background

In this section, initially we are going to describe the techniques with their theories we were going to embed, then we will describe our model, and the results.

4.1.1 Word Embedding: Word2Vec

4.1.1.1 Word Embedding

In the context of machine learning, most of the algorithms, as well as all neural network models, require features to be represented in numerical forms; they just would not work on strings of plain text. In a nutshell, word embedding is just representing texts as some kind of numbers, so they can be fed into our algorithms or neural net models. Word embeddings became one of the most exciting area of research.

4.1.1.2 One-Hot Vectors

One most basic form of word embedding is One Hot Encoding method. In this approach, each word is denoted as a sparse vector whose dimensions are represented by a word. That is, if we have 50 words in our vocabulary, then each of the 50 words will be represented as a 50 dimensional vector. Now say, *'dog'* is in the vocabulary and its index is 9, then in its vectorized representation, the 9^{th} bit is set to 1, all other 49 dimensions are marked as 0's. Although, this technique makes each word uniquely identifiable and input ready for neural net models, it does not have significant value in the context of Natural Language Processing. Because, this method does not capture the syntactic and semantic information inherent in the word itself, and the context it appears in.

4.1.1.3 Distributional Vectors

Although in a unanimous form a word itself has a meaning, it is actually defined by the context, that is, the words that surround it. Hence, it is arguably but safe to assume that words that appear within a boundary of similar contexts, or similar neighboring

words, inherits similar meaning (e.g. *elevator* and *lift* will both appear next to *down*, *up*, *building*, *floor*, and *stairs*).

This knowledge raised an idea to represent words by their contexts. Suppose we represent each word by a $|V|$ dimensional vector, where V is the set of unique words. Now, unlike one hot encoding we don't mark the index of the word, rather we mark the indices of the words that surround it. Then we find all the occurrences of a certain word (e.g. *elevator*) from a big corpus. Then we decide a neighboring window K, all the words inside this region of the word (e.g. *elevator*) are considered as its neighbors. For example, if the corpus has the sentence *"The left elevator goes down to the second floor"*, with a window of size 5, (e.g. *elevator*) then has the following neighbors: *the*, *left*, *goes*, *down*. In the 'counting methods' technique the vector of *elevator* is then modified by adding the number of occurrences of *the*, *left*, *goes*, and *down* at their respective indices. When this process is finished, each word is represented by a vector having the frequencies of all its neighbors. Usually it is then normalized to obtain a probabilistic distribution, that is, how likely each of the words supposed to appear as a neighbor of some other word.

If we can assume that these distributional vectors are capturing the meaning of words, then by some vector similarity measure it will be possible to capture similar words. Let's say we take cosine similarity measure, and we apply on all possible word pairs - we can expect *elevator* and *lift* to yield a higher similarity score than *elevator* and, say, *cat*. As we will see, this assumption turned out to be true, and a revolutionary development towards the NLP tasks where semantic similarity is very much needed to be captured.

Although the idea of distributed representation of words was lucrative, in the real world obtaining it from a large corpus was computationally infeasible. Because. storing each of these $|V|$ words in a $|V|$-dimensional vector results in a $|V|^2$ matrix - this is quite large, and performing operations on all words were computationally heavy.

These obstacles were overcome by the Neural Net models. The idea was to represent each word in a relatively small vector space (typically 50 to 1000) where each dimension of that space will represent a feature of the word, rather than the word itself, and the target is to bring words with similar features closer and words with dissimilar features pushed afar. This approach was first presented by [Bengio et al., 2003], but gained extreme popularity with Word2Vec [Mikolov et al., 2013b] (discussed in Section 4.1.1.4).

Now unlike the counting method, each vector is learned rather than frequency calculated from the corpus. In the most basic sense, the algorithm goes as follows: each vector is randomly initialized, and by going through a large corpus at each step, the vectors of both the target word and the vectors of the neighboring words are updated in a way to bring them closer. After going through many such steps, the vectors become meaningful, yielding similar vectors to similar words.

The lower dimensional representation still captures the essence of distributed representation with advantages of lower computation cost. In addition, it turned out that different vector arithmetics can be applied to these vectors to understand more about the features each of the word holds in its vectorized representation.

4.1.1.4 Word2Vec

Word2vec presented by [Mikolov et al., 2013a] is an efficient model that finds the semantic distributional vector representations given a raw text. It uses a simple Multilayer Perceptron neural net model with a single hidden layer to learn those representations. There are two types of Word2vec models based on the targets, the Continuous Bag-of-Words model (CBOW) and the Skip-Gram model. CBOW predicts target words (e.g. *'mat'*) from source context words (*'the cat sits on the'*), while the skip-gram predicts source context-words from the target words. As it was later proven than for a large set of data, skip-gram tends to perform better than the CBOW technique, we focus in the skip-gram model in the remaining section.

4.1.1.4.1 Skip-gram The model is a simple neural network which contains a single hidden layer, and is used to perform a certain task, but after that, in a conventional sense, the network is not used for the task it has been trained for. Instead, the goal is actually just to learn the weights of the hidden layer - we'll see that these weights are actually the "word vectors" that we're trying to learn.

We're going to train the neural network to do the following. Given a specific word in the middle of a sentence (the input word), look at the words nearby (there is actually a parameter called window size, usually set to 5, meaning 5 words behind and 5 words ahead, that defines the neighborhood) and pick one at random. The network is going to tell us the probability for every word in our vocabulary of being the "nearby word" that we chose. The output probabilities are going to relate to how likely it is to find each vocabulary word nearby our input word. For example, if we gave the trained network the input word *"Soviet"*, the output probabilities are going to be much higher for words like *"Union"* and *"Russia"* than for unrelated words like *"watermelon* and *"kangaroo"*.

We have to first find a way to represent the words of our vocabulary. Let's say, we have a vocabulary of 10,000 words. We are going to represent a word like *'ants'* as one hot encoded vector like we did in Section 4.1.1.2. It will have 10,000 components, one for every word in our vocabulary, and set the corresponding bit for the word *'ants'* to 1. Now look at the architecture of the Word2Vec model shown in Figure 4.1[1]. The output of the network is a single vector of the size of our vocabulary, here it is 10,000, containing probabilities of each word appearing as a neighbor of the input word *'ants'*.

Finally the rows of the weight matrix will be our word vectors. So the end goal of all of this is really just to learn this hidden layer weight matrix – the output layer we'll just toss when we're done!

The 1 x 300 word vector for *'ants'* then gets fed to the output layer. The output layer is a softmax regression classifier. Each output neuron (one per word in our vocabulary!) will produce an output between 0 and 1, and the sum of all these output values will add up to 1. Specifically, each output neuron has a weight vector which it multiplies against the word vector from the hidden layer, then it applies the function $\exp(x)$ to the result. Finally, in order to get the outputs to sum up to 1, we divide this result by the sum of the results from all 10,000 output nodes.

The skip-gram neural network contains a huge number of weights. For our example with 300 features and a vocabulary of 10,000 words, that's 3M weights in the hidden

[1] `http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/`

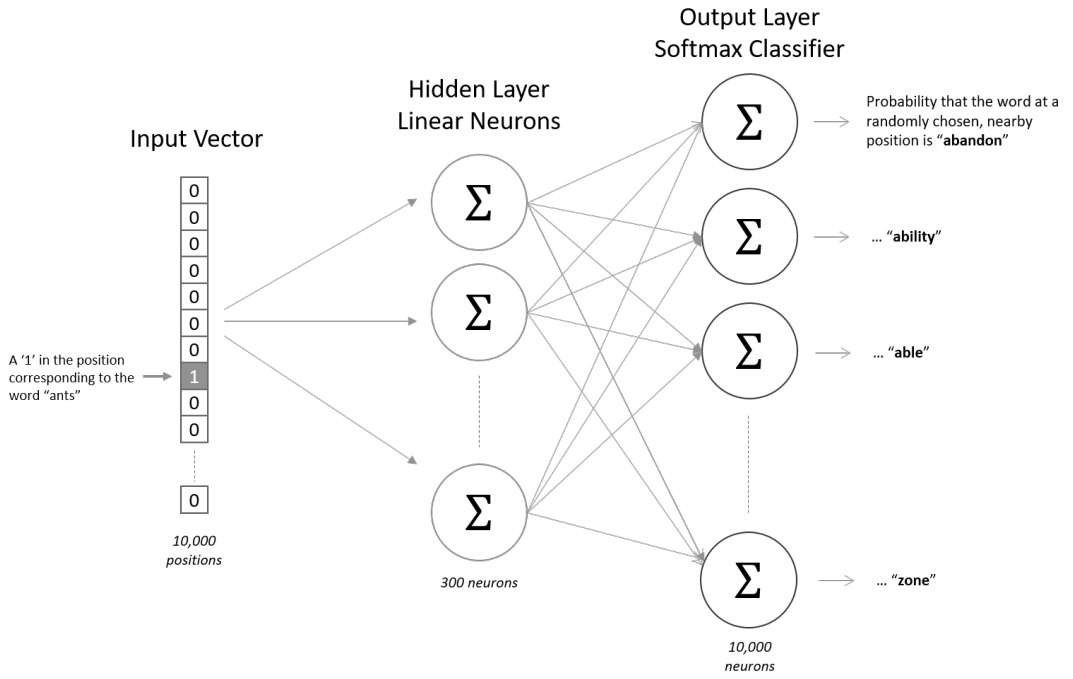


Figure 4.1: Architecture of Word2Vec.

layer and output layer each. Training this on a large dataset would be prohibitive, so the Word2vec authors introduced a number of tweaks to make training feasible. They are *Sub-sampling*, *Negative Sampling* and *Hierarchical Softmax*. But we will leave those techniques from describing, which can be found in [Goldberg and Levy, 2014].

4.1.1.4.2 Reasoning with word vectors If two different words have very similar "contexts" (that is, what words are likely to appear around them), then our model needs to output very similar results for these two words. And one way for the network to output similar context predictions for these two words is if the word vectors are similar. So, if two words have similar contexts, then our network is motivated to learn similar word vectors for these two words!

As it turned out, the learned vectors for the words capture a deep and meaningful relationships amongst them. And not only that, vector arithmetics applied over those representations extracted more informations than that we had imagined. For example, if we denote the vector for word i as x_i, and focus on the singular/plural relation, we observe that x_{apple}–$x_{apples} \approx x_{car}$–$x_{cars}$, x_{family}–$x_{families} \approx x_{car}$–$x_{cars}$, and so on. As a result [Mikolov et al., 2013c] showed that using simple algebraic computations more informations can be obtained. For example, vector(*"King"*) – vector(*"Man"*) + vector(*"Woman"*) results in a vector that is closest to the vector representation of the word *"Queen"*. SemEval 2012 task of measuring relation similarity came up with more surprising sets of results. And to add more, Word2vec captured not only semantic but also syntactic regularities.

Here are some more results shown in Figure 4.2 achieved using the same technique:

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Figure 4.2: Relationship pairs in a word embedding. From [Mikolov et al., 2013a]

4.1.2 Recurrent Neural Networks

We discuss the basics of Recurrent Neural Networks (RNNs) which are deep learning models that are becoming increasingly popular. We don't intend to get too heavily into the math and proofs behind why these work but are aiming for a more abstract understanding.

Recurrent Neural Networks were created in the 1980's but have just been recently gaining popularity from advances to the networks designs and increased computational power from graphic processing units. They're especially useful with sequential data because each neuron or unit can use its internal memory to maintain information about the previous input. This is great because in cases of language, *"I had washed my house"* is much more different than *"I had my house washed"*. This allows the network to gain a deeper understanding of the statement. This is important to note because reading through a sentence even as a human, we pick up the context of each word from the words before it.

4.1.2.0.1 Architecture A typical RNN is a network that can feed itself over time, that is, it has a feedback loop connected to itself as input. This feedback loop makes the RNN act like a *'memory'* which remembers information (theoretically) about what has been seen (or calculated) so far. Figure 4.3 shows how a typical RNN looks like.

Figure 4.3 also shows a RNN being unfolded into a full network. Unrolling the network reveals the complete time-steps as a feed-forward multilayer network. For example, if the sequence is a sentence of 5 words (or 5 time-steps), the network would be unrolled into a 5-layer neural network, one layer for each word (or time-step). The formulas that govern the computation happening in a RNN are as follows:

- x_t is the input at time step t. For example, x_1 could be a one-hot vector corresponding to the second word of a sentence.
- s_t is the hidden state at time step t. It's the *'memory'* of the network. s_t is calculated based on the previous hidden state and the input at the current state:

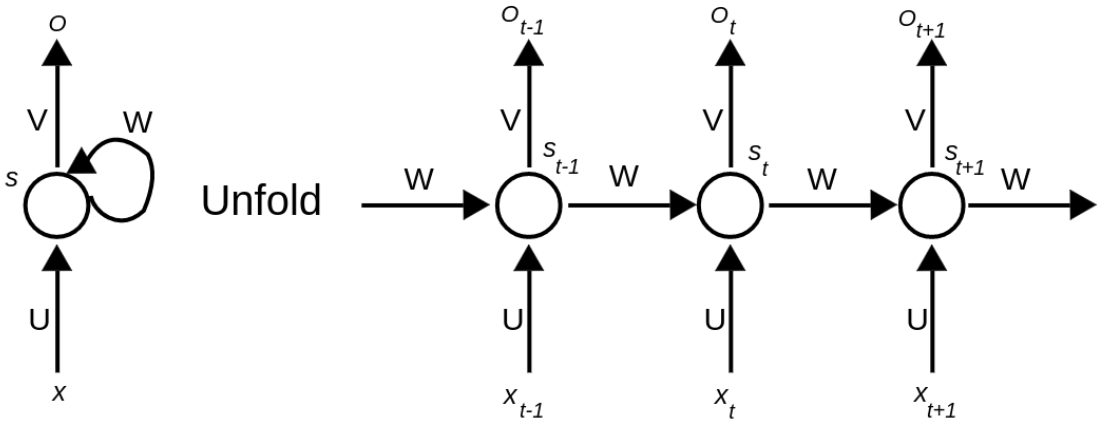


Figure 4.3: A Recurrent Neural Network and the unfolding in time

$s_t = f(Ux_t + Ws_{t-1})$. The function f usually is a nonlinearity such as $tanh$ or $ReLU$. s_{-1} which is required to calculate the first hidden state, is typically initialized to all zeros.

- o_t is the output at step t. For example, if we wanted to predict the next word in a sentence it would be a vector of probabilities across our vocabulary. $o_t = softmax(Vs_t)$.

Few things to notice here:

- If we assume the hidden state s_t as the memory of the network, then it can be stated that s_t captures all the informations of the previous time steps from s_0 to s_{t-1}. At step t the output o_t is calculated based on the memory or state s_t at time t. Although theoretically true, in reality a hidden state can not remember information from far too time steps ago.
- When we unraveled the network, we observed a chain of layers connected with synaptic weights, forming a deep layered network. But unlike the traditional deep nets all those weights are same (U, V, W above) across the time-steps. This helps target only one goal with different inputs, and it also reduces the total number of trainable parameters.
- Although the Figure 4.3 shows outputs at each time steps, this is not necessarily true for all RNNs. The behavior of a RNN can be changed so that it also can output at specific intervals or at the final time-step. Depending on the types of problems we want to solve this was needed. For example, when doing Sentiment Analysis, we do not want output after processing each word, rather after a sentence or a paragraph.

4.1.2.0.2 Strengths of RNN RNNs have shown great success in so many NLP tasks. A few of them are stated below:

- **Language Modeling and Generating Text:** When we write a text, we can always notice that the choice of words depends on the words we have previously written. A language model precisely does that by finding the words that are highly probable for that current position. Applications of these can be seen in current smart-phones where a keyboard predicts the next character or even next word while typing. Some papers implementing Language models are [Mikolov et al., 2010], [Mikolov et al., 2011], and [Sutskever et al., 2011].
- **Machine Translation:** Machine Translation takes an input of a sequence of words in a given language and translates it into another language. While earlier models started with word by word translation and then rearranging them, RNN captures information hidden in the sequence by completely reading them, then through an intermediate representation tries to predict the arranged sequence of words in the target language. Some research papers regarding machine translation using RNN are [Liu et al., 2014], [Sutskever et al., 2014], and [Auli et al., 2013].
- **Speech Recognition:** Today RNNs are also trained to find the phonetic segments and textual representation of them from an audio signal. Research paper regarding speech recognition is [Graves and Jaitly, 2014].

4.1.2.0.3 Problems with RNN Earlier we saw that a RNN is nothing but a chain of layers where layers are states at different time-steps, connected by shared weights. Hence, it is easy to understand that the training of such network is same as the Multi-layer Perceptron networks. The only difference is that at each layer the gradient is dependent upon all the gradients of the previous layers. The technique of calculating the gradients of all the previous layers (time-steps) and summing them up to get the gradient of the current layer is called Backpropagation Through Time (BPTT).

Theoretically it seems that RNNs can capture long term dependencies, but in reality vanilla RNNs suffer from vanishing and exploding gradient problems that restrict them from remembering long distance relationships. This led to the development of some enhanced mechanisms to deal with those problems. Some of them are briefly stated below:

4.1.2.1 LSTM

LSTMs are nothing but RNNs with a few tweaks in the sleeve. Rather than using Sigmoid activation function, that is the main reason of the vanishing gradient problem, LSTM uses a different activation function (ReLu). A few internal circuits are added, which are called memory gates or forget gates, that decide what to remember and what to forget at each time step. It turns out that LSTM can in reality capture long-term dependencies.

4.1.2.2 GRU

A slightly more dramatic variation which resulted a more simpler network with very less number of parameters than LSTM is the Gated Recurrent Unit, or GRU, introduced by [Cho et al., 2014]. It combines the forget and input gates into a single “update gate”,

and also merges cell state and hidden state to reduce the number of parameters. This model is gaining popularity each day.

4.1.2.3 Bidirectional RNN

Bidirectional RNNs (or LSTM or GRU) are based on the idea that the output at time t may not only depend on the previous elements in the sequence, but also elements yet to be seen. For example, to predict a missing word in a sequence it is advisable to look at both the left and the right neighboring words. Bidirectional RNNs are nothing but just two RNNs stacked on top of each other, where the first one receives the original sequence and the latter receives the reverse sequence. The output is usually concatenation of both those RNNs.

4.2 Our Work

4.2.1 Introduction

Similar to most of the successful approaches to the Event Extraction tasks, we decided to follow the Pipeline method, where we will predict the triggers first, and based on those predictions we will extract arguments for those triggers. Hence, we propose two subtasks:

1. Trigger detection
2. Arguments extraction

We will present these two subtasks separately, along with respective models, parameters, and scores at the word level, finally the scores obtained by using BioNLP ST 2013 **Evaluation Script**.

4.2.2 Preprocessing

Before discussing the architectures for both of the subtasks, we need to discuss the preprocessing we made. These are:

- Sentence and word tokenization are already performed by Standford Parser for all of the Training, Development and Test data, and shared as external resources by the BioNLP community. We have used those preprocessed data.
- As both the entities and the triggers can contain more than one word, we needed a way to represent those phrases, so that we can give each word separately as an input to and get an output from our networks, without ambiguity. To do that, we decided to replace each *entity type* and *event type* with added prefixes of **B-** and **I-**, as in ***B**egining* word and ***I**nside* word respectively. This technique helped predict triggers with phrases, and also reduced complexity at the time of post-processing.
- For the argument extraction task, in our proposed model we needed indices of words in a sentence, so that the predictions will tell us at what indices the arguments for a given trigger are. In the cases of multiple words for entities, we mark them with same index.

Let's look at an example for clarification for both of the above: From Table 4.1, it can

Low	**metastatic**	**and**	**G-418**	**resistant**	**H-3**	**cells**	**were**	**paired**	**cultured**	**with**	**BALB/c3T3**	**fibroblasts**
	Metastasis		Simple_Chemical		Cell			Planned_process			Cell	
	B-Metastasis		**B-Simple_Chemical**		**B-Cell**	**I-Cell**		**B-Planned_process**	**I-Planned_process**		**B-Cell**	**I-Cell**
1	2	3	4	5	6	7	8	9	10	11	12	13
1	2	3	4	5	**6**	**6**	7	*8*	*9*	10	**11**	**11**

Table 4.1: Preprocessing steps

be seen that **H-3 cells** is a phrase with entity type *Cell*, and hence in the preprocessing step **H-3** is tagged with *B-Cell* and **cells** is tagged with *I-Cell*, and same goes with the other entity and trigger phrases. But in the cases of single words, the tags are prefixed with *B-*. Hence, **metastactic** is tagged with *B-Metastatic*, and **G-418** is tagged with *B-Simple_Chemical*.

For the indexing, in the Table 4.1, it can be seen that the indices of only the entities with phrases are modified. Hence, indices of **H-3** and **cells**, and indices of **BALB/c3T3** and **fibroblasts**, are kept same (in this case 6 and 11 respectively). But we do not modify the indices of the trigger phrases, because, while in the first step we are going to predict the triggers, merging of trigger words happen at the post-processing stage, hence modified indices for triggers are not available beforehand.

4.2.3 Obtaining Word2vec

BioNLP community provides pre-trained Word2vec models[2], using texts obtained from PubMed[3], PMC[4], and their combination with a recent English Wikipedia dump. There

Word2vec models	**Size**
PMC-w2v.bin	1.90 GB
PubMed-and-PMC-ri.tar.gz	2.31 GB
PubMed-and-PMC-w2v.bin	3.09 GB
PubMed-w2v.bin	1.78 GB
wikipedia-pubmed-and-PMC-w2v.bin	4.11 GB

Table 4.2: Pre-trained Word2vec models provided by BioNLP community

are five such models as shown in Table 4.2, out of which we used **PubMed-w2v.bin** to generate Word2vec vectors for our task. The features length provided in these Word2vec models are 200. For the unknown words which were not found in the PubMed Word2vec model, we generated 200 length random vectors, and made dictionaries for reference use.

[2] `http://evexdb.org/pmresources/vec-space-models/`

[3] PubMed comprises more than 27 million citations for biomedical literature from MEDLINE, life science journals, and online books.

[4] PubMed Central®(PMC) is a free full-text archive of biomedical and life sciences journal literature at the U.S. National Institutes of Health's National Library of Medicine (NIH/NLM).

4.2.4 Trigger Detection

4.2.4.1 Model

Similar to Parts-of-speech tagging, the trigger detection task is also a sequence labeling problem, where we would like to classify each word of a given sentence to one of the event types, or a negative. The architecture of the Trigger Detection model is shown in Figure 4.4. The four layers of the model are described below:

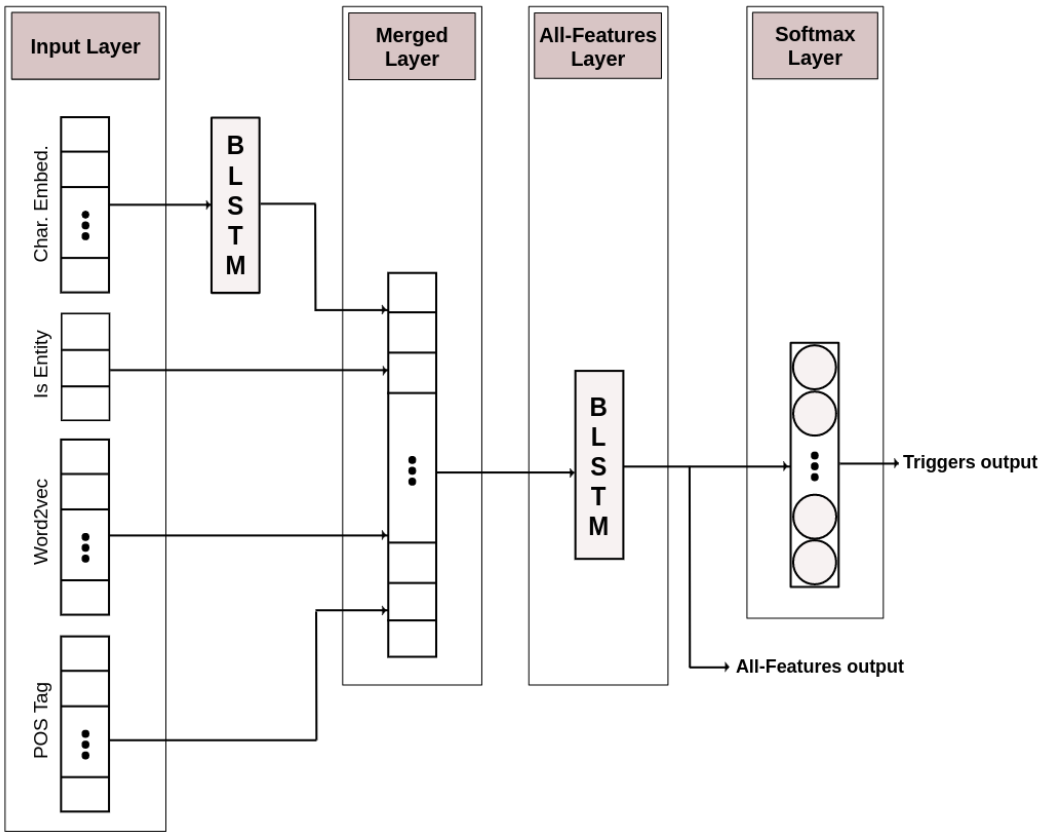


Figure 4.4: Trigger Detection Model

Input Layer: Similar to Parts-of-Speech task, we need character features to predict the triggers. Thus, the first input of this layer is *Character Embedding* followed by a BLSTM capturing character features. As an entity can not be a trigger word, it is beneficial to let the network know whether the given word is an entity or not. Therefore, we give *Is Entity* information (i.e. given a word is it an entity or not) as the second input. To capture the semantic information we give *Word2vec* as the third input. Finally, as only some specific POS tagged words (e.g. mostly nouns and verbs) are found to be triggers, we provide *Parts of Speech* tags as the final input in the *Input Layers*.

Merged Layer: This layer simply concatenates *Character Features* output from the first BLSTM, and *Is Entity*, *Word2vec*, and *Parts of Speech* information from the *Input layer*.

All-features Layer: This layer contains one BLSTM layer which takes the input from the *Merged Layer* and produces *All-Features* output.

Softmax Layer: This is the classification layer which takes *All-features* output as input for the actual classification task, and maps each word to one of the event types, or negative.

4.2.4.2 Parameters

The parameters of the network and their respective vector sizes are shown in Table 4.3. The shape of the input and output matrices are shown in Table 4.4 (*None* is replaced at

Character Vector size	28
Maximum no. of characters in a word	20
Maximum no. of words in a sentence	50
No. of nodes in the first BLSTM	64
No. of nodes in the Softmax layer	86
Is Entity Vector size	3
Word2vec vector size	200
POS tag vector size	39
No. of nodes in the second BLSTM	70

Table 4.3: Trigger Detection model parameters

the training and testing time, by the number of samples, i.e. in this case by the number of sentences in the training and testing data, respectively).

Char. Embed.	(None, 50, 20, 28)
Is Entity	(None, 50, 4)
Word2vec	(None, 50, 200)
POS tag	(None, 50, 39)
Softmax	(None, 50, 86)

Table 4.4: Shapes of the Trigger Detection model input and output matrices

4.2.4.3 Trigger Detection Results

After training with 50 iterations and batch size 1, during predictions, at the word level, 1978 trigger words matched out of 2596, giving accuracy 76.19%, and 24523 non-trigger words matched out of 25011, giving accuracy 98.05%, with overall trigger accuracy being 97.72%. The benchmarking scores are shown in Table 4.5. When we looked into the

Precision	**Recall**	**F-score**
59.72	64.28	59.52

Table 4.5: Precision, Recall, F-score of trigger detection task

data, we found that in the training samples, there were 10% trigger words, and 90% non-trigger words. This severe class imbalance resulted this low benchmarking scores despite high accuracies.

4.2.5 Arguments Extraction

4.2.5.1 Model

The formulation of Arguments Extraction task is somewhat different than that of the sequence labeling tasks. In this case, for a given trigger in a sentence, we want to predict at which indices in that sentence, the arguments are, i.e. for a given trigger, for each of the argument types, at each word positional index we predict whether it is a positive or a negative argument index. The architecture of the Argument Extraction model is shown in Table 4.5. The layers are described below:

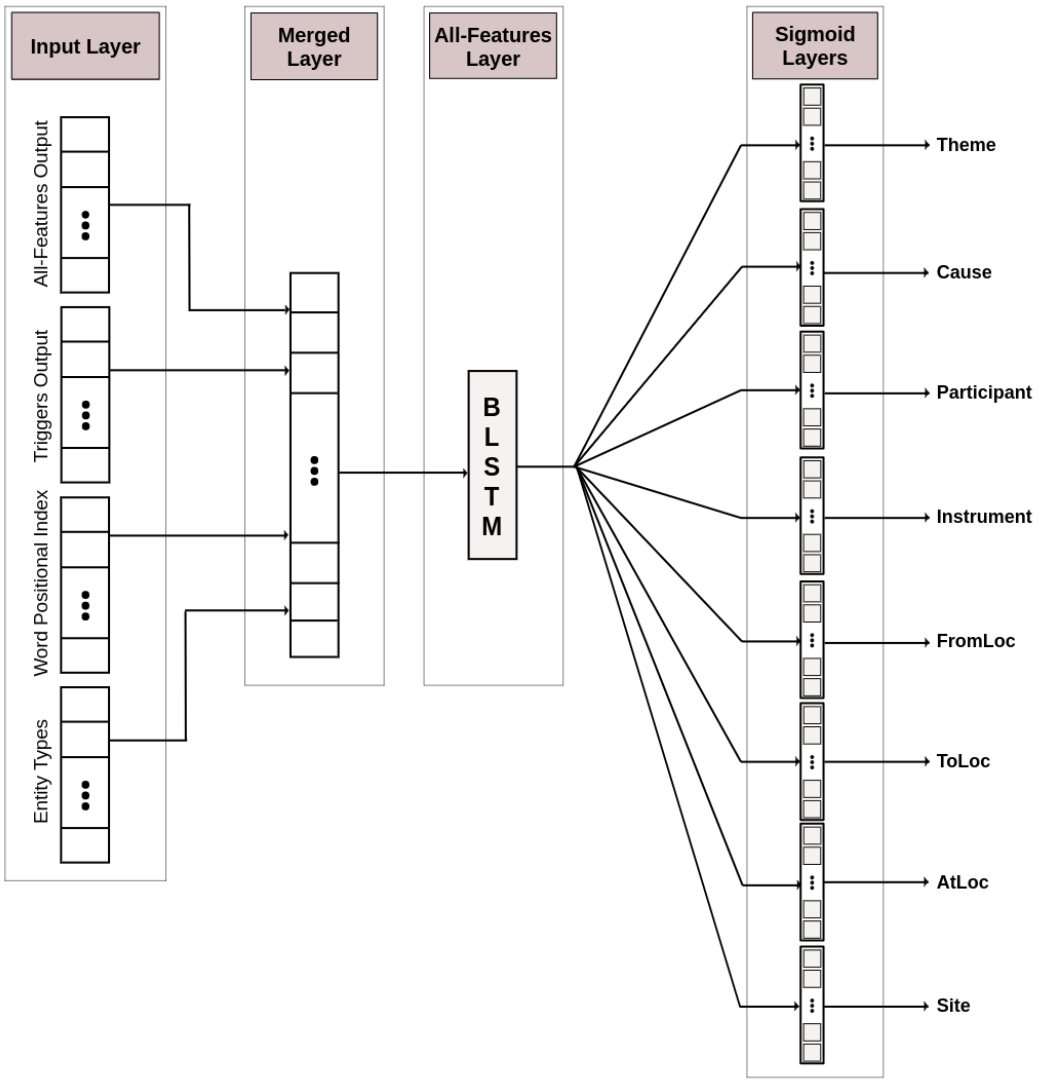


Figure 4.5: Arguments Extraction Model

Input Layer: To predict whether a word is an argument of a trigger, it is also necessary to capture character features, word vector representations, entity informations, and POS tag informations of all the words surrounding the trigger word. As all these informations are already provided in the Trigger Detection model, and has already been trained, it is feasible to use that already trained model to capture the semantics rather than training again. Hence, we take the outputs of the *All-Features layer* of the Trigger Detection model as the first input of the *Input Layer* of Argument Detection model. As the arguments need to be predicted only for trigger words, it is necessary to tell the network whether a given word is a trigger or not. Hence the *Softmax layer's triggers output* of the Trigger Detection model is given as the second input of the *Input Layer* of Argument Detection model. As we are predicting at each positional index whether there is an argument of specific type at that index, it is necessary to provide word positional index along with every word. Hence, *Word Positional Index* is the third input in the *Input Layer*. Finally, as a specific entity type form as specific argument type for a specific trigger type, along with trigger information, it is also necessary to provide entity type informations of each word. Hence, the final input of the *Input Layer* is the *Entity Types* information.

Merged Layer: This layer simply concatenates the input vectors from the *Input Layer*.

All-Features Layer: This layer contains a single BLSTM which takes input from the *Merged Layer* and produces *All-Features output*.

Sigmoid Layers: We have eight different types of arguments, hence we built eight different sigmoid layers. Each of these layers consists nodes denoting all word positional indices. Then at each of those nodes of positional index, we try to predict whether it is a positive or a negative argument. Hence, for a sigmoid layer, the task becomes binary classification at each positional index. Each of these sigmoid layers take inputs separately from *All-Features layer*, but trains the complete network based on each others' predictions, hence making the Argument Extraction a *joint learning* task.

4.2.5.2 Parameters

The parameters of the network and their respective vector sizes are shown in Table 4.6. The shape of the input and output matrices are shown in Table 4.7 (*None* is replaced at

All-Features output vector size	140
Triggers output vector size	86
Word positional index vector size	51
Entity types vector size	34
No. of nodes in the All-Features Layer BLSTM	80
Arguments vector size (for each sigmoid layer)	53

Table 4.6: Arguments Extraction model parameters

the training and testing time, by the number of samples, i.e. in this case by the number of sentences in the training and testing data, respectively).

All-Features output	(None, 50, 140)
Triggers output	(None, 50, 86)
Word positional index	(None, 50, 51)
Entity types	(None, 50, 34)
Each Sigmoid Layer	(None, 50, 53)

Table 4.7: Shapes of the Arguments Extraction model input and output matrices

4.2.5.3 Arguments Extraction Results

After training with 200 iterations and batch size 1, during predictions, for the 76.19% correctly classified triggers by the Trigger Detection model, 736 Themes matched out of 1836, giving accuracy 40.09%; 127 Cause matched out of 548, giving accuracy 23.18%; 17 Participant matched out of 97, giving accuracy 17.53%; 19 Instrument matched out of 146, giving accuracy 13.01%; 0 FromLoc matched out of 7, giving accuracy 0.0%; 15 ToLoc matched out of 48, giving accuracy 31.25%; 16 AtLoc matched out of 90, giving accuracy 17.78%; and 0 Site matched out of 19, giving accuracy 0.0%. Precision, Recall, F-score measures are shown in Table 4.8. Still these scores are a little misleading when

Argument Type	**Precision**	**Recall**	**F-Score**
Theme	94.76	92.49	93.61
Cause	96.95	95.97	96.46
Participant	97.93	97.78	97.85
Instrument	97.82	97.47	97.65
FromLoc	98.02	98.18	98.10
ToLoc	97.97	98.06	98.02
AtLoc	97.92	97.83	97.88
Site	98.01	97.97	97.99

Table 4.8: Scores of Arguments Extraction task

we look at the accuracies stated earlier. Because, for a sentence with 50 words, say, a trigger can have as much as 5 arguments in that sentence. Then the target vector will contain 45 zeros and 5 ones. Hence, if the predictions match zeros almost all the time, but mismatches ones (i.e. True Positives are always high, False Positives and False Negatives are very low), still these scores will be higher, because of the severely imbalanced labels (0's in this case with respect to 1's).

4.2.6 Events Construction and Benchmarking Results

At the post-processing step, we combine a predicted trigger at the *Trigger Detection* stage with its arguments from *Arguments Extraction* stage to obtain *Events* for that trigger. Then we created *a2* files required for the *Evaluation Script* to obtain the

Benchmarking Results. We present a few results in comparison with TEES 2.1 system in the Table 4.9.

Event Class	**Our System**			**TEES 2.1**		
	recall	**precision**	**fscore**	**recall**	**precision**	**fscore**
Development	48.33	44.62	46.40	72.12	70.75	71.43
Blood_vessel_development	64.64	66.22	63.84	80.51	90.65	85.28
Growth	**78.57**	73.33	75.86	68.06	85.96	75.97
Death	17.07	21.21	18.92	79.66	83.93	81.74
Cell_death	55.88	67.86	61.29	66.94	81.00	73.30
Cell_proliferation	38.89	48.28	43.08	74.60	86.24	80.00
Breakdown	55.56	71.43	62.50	67.35	89.19	76.74
Metastasis	50.00	47.50	48.72	69.23	72.67	70.91
Gene_expression	38.46	48.11	42.75	76.91	79.56	78.21
Pathway	23.26	17.54	20.00	64.57	79.61	71.30
Binding	18.03	25.58	21.15	35.45	62.90	45.35
Regulation	1.73	2.19	1.93	26.53	42.46	32.66
Positive_regulation	6.64	8.56	7.48	40.15	53.54	45.89
Negative_regulation	7.84	9.38	8.54	42.62	54.40	47.79
Overall (All 40 Classes)	21.16	26.52	23.54	48.76	64.17	55.41

Table 4.9: Event Extraction Benchmarking Results

4.3 Discussion

When we look into the benchmarking results for Event Extraction in Table 4.9, we see that although only *Growth* event class giving better recall in our model than in the state of the art TEES 2.1 system, we have achieved comparable results. We found the following reasons for having such variance in the results:

- As the inputs of the Trigger Detection model, in conjunction with Character Embedding and Word2vec, we experimented with different combinations of features to improve the trigger detection results. They are:

  1. Word Positional Index, Entity Types
  2. Is Entity information, Entity Types present around the current word
  3. Is Entity information, POS tag
  4. Is Entity information
  5. Entity Types present around the current word

  We found that Case 1 resulted in the least accuracy, and Case 3 gave a little edge (0.5 to 1.0% better) over all the others.

- Although we have achieved a very good accuracy while detecting triggers, the low f-score (Table 4.5) suggests that it is going to impact the arguments extraction task, which is the problem of such pipeline methods. That is, an incorrectly classified trigger brings ambiguities to the network, and affects the learning of arguments extraction task.
- Due to the massive imbalance in the trigger classes themselves it is expected that the trigger detection task will be hard.
- Although there is enough data to use deep learning methods for trigger detection, upon inspection we found that the number of instances for each argument type, are very small. This is one of the main reasons of poor performance in the Arguments Extraction task.
- As the prediction of arguments are average for *Theme* and *AtLoc* types, but very poor for the others, very less number of accurate events could be constructed. And as the final benchmark happens over the actual construction of events, a very good trigger accuracy could not improve overall Event Extraction performance. That is why although *Death*, *Cell_proliferation*, *Gene_expression*, *Positive_regulation* and *Negative_regulation*, each matched more than 80% of the time, they could not contribute much in the final event extraction result.
- It is also possible that, in the Argument Extraction model the Word Positional Indices might influence the learning in a negative way, that is, the network might be learning, for a given trigger, at which indices an argument type is appearing most of the times and predicting those indices, hence sometimes predicting non-trigger and non-entity words as arguments.

4.4 Summary

In this chapter, we looked into some brief theoretic backgrounds necessary to understand our model, and presented a pipeline model for the Cancer Genetics Event Extraction task. We discussed the architectures of both of our models, and discussed the results. We finally presented the Benchmarking scores of our model for Event Extraction task, and reviewed the benefits and shortcomings of our proposed model.

Chapter 5

Conclusion & Future Work

In this thesis, we looked into the task of Event Extraction from Bio-medical documents on Cancer Genetics from deep learning point of view. We proposed a pipeline model with two subtasks to solve the problem. We noticed that like any other pipeline model, our system also suffered from error propagation. The final benchmarking results revealed that our system fairly competed with the state of the art system for event extraction tasks. Now, as we understand the benefits and shortcomings of our designed system, we need to work on the following:

- Applying Convolution Neural Network in the Arguments Extraction task to know whether it gives better features than the LSTM we used.
- As mentioned in Section 4.3, it is probable that the word positional indices affecting the arguments extraction task, we need to remove this input from the model and improvise.
- We now believe that rather than looking at the complete sentence at once and then finding the arguments positions for a given trigger, it might be a better idea to compare the given trigger with each of the other entities and triggers to learn whether one such pair forms an event or not.
- Finally, as we stated earlier, pipeline models suffer from error propagation, we want to devise a joint model that will learn to predict triggers and their arguments all in one go.

Bibliography

[Auli et al., 2013] Auli, M., Galley, M., Quirk, C., and Zweig, G. (2013). Joint language and translation modeling with recurrent neural networks. In *EMNLP*, volume 3, page 0.

[Bengio et al., 2003] Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.

[Björne and Salakoski, 2013] Björne, J. and Salakoski, T. (2013). Tees 2.1: Automated annotation scheme learning in the bionlp 2013 shared task. In *Proceedings of the BioNLP Shared Task 2013 Workshop*, pages 16–25. Association for Computational Linguistics.

[Cho et al., 2014] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

[Goldberg and Levy, 2014] Goldberg, Y. and Levy, O. (2014). word2vec explained: Deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*.

[Graves and Jaitly, 2014] Graves, A. and Jaitly, N. (2014). Towards end-to-end speech recognition with recurrent neural networks. In *ICML*, volume 14, pages 1764–1772.

[Liu et al., 2014] Liu, S., Yang, N., Li, M., and Zhou, M. (2014). A recursive recurrent neural network for statistical machine translation.

[Mikolov et al., 2013a] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

[Mikolov et al., 2010] Mikolov, T., Karafiát, M., Burget, L., Cernockỳ, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *Interspeech*, volume 2, page 3.

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

[Mikolov et al., 2013b] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

[Mikolov et al., 2013c] Mikolov, T., Yih, W.-t., and Zweig, G. (2013c). Linguistic regularities in continuous space word representations. In *Hlt-naacl*, volume 13, pages 746–751.

[Nédellec et al., 2013] Nédellec, C., Bossy, R., Kim, J.-D., Kim, J.-J., Ohta, T., Pyysalo, S., and Zweigenbaum, P. (2013). Overview of bionlp shared task 2013. In *Proceedings of the BioNLP Shared Task 2013 Workshop*, pages 1–7. Association for Computational Linguistics Sofia, Bulgaria.

[Pyysalo et al., 2013] Pyysalo, S., Ohta, T., and Ananiadou, S. (2013). Overview of the cancer genetics (cg) task of bionlp shared task 2013. In *Proceedings of the BioNLP Shared Task 2013 Workshop*, pages 58–66. Association for Computational Linguistics.

[Sutskever et al., 2011] Sutskever, I., Martens, J., and Hinton, G. E. (2011). Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1017–1024.

[Sutskever et al., 2014] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.