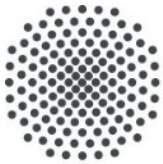


University of Stuttgart

Institute for Natural Language Processing (IMS)

Pfaffenwaldring 5B
70569 Stuttgart

**Master Thesis**

# Detecting Ambiguity in Conversational Systems

Avik Banerjee

| | |
|---|---|
| **Study Program :** | M.Sc. Computer Science |
| **Examiner :** | Prof. Dr. Ngoc Thang Vu |
| **Advisor :** | Daniel Ortega |
| **Start Date :** | 01.01.2021 |
| **End Date :** | 01.08.2021 |

# Abstract

The question of detection of user search queries has been explored by many authors. With the advent of speech based search interfaces, narrowing down the scope of search based on user intent becomes even more important. A prominent part of determining the user's goals is first detecting whether the query is ambiguous, based on which, clarifying questions can be posed. Previous works have mostly attempted to classify user intent into pre-defined categories that may not be suitable for open-domain settings. This thesis explores multiple methods to detect the level of ambiguity of the first query input by the user. Two principal approaches are presented in this work, both of which depend on information provided by documents retrieved from the search operation. The first approach creates a graph based on the similarities between the documents and the second approach generates a graph from the concepts covered in those documents. The graphs are then processed by a graph convolutional network and classified into four levels of ambiguity. The models are tested on data provided by the ClariQ challenge and are found to depend on the documents taken into scope as well as the distribution of the documents in the search results. The best results obtained by the models have been shown to improve over traditional sentence classification approaches and have been compared to the top ranked entries in the challenge. Additionally, ways to improve the datasets and the models have been proposed.

# Acknowledgement

Throughout the research for this thesis, I have received support and assistance from multiple sources.

I would like to acknowledge the support and guidance of Prof. Dr. Ngoc Thang Vu of the Institute for Natural Language Processing at the University of Stuttgart for his guidance and criticisms which helped validate and refine my research questions and methodology.

I would like to acknowledge the supervision and patient support I received at every step of the research from my thesis supervisor, Daniel Ortega.

I would also like to extend my appreciation to the Institute for Natural Language Processing at the University of Stuttgart for providing the infrastructure necessary for conducting the research.

Finally, I would like to thank my friends and family for their constant moral support throughout the duration of the research.

# Contents

# Chapter 1

# Introduction

The World Wide Web is a treasure trove of information [1]. Among trillions of web pages available there and millions added everyday, we can find facts and figures that interest us. Earlier, searching the web involved sifting through directories and catalogues maintained by multiple organizations and research groups. Then came the first popular search engine, Yahoo! Search in 1994, followed by Google and the multitude of search options we have available today. However, all popular search engines mainly relied on text queries as their inputs. Earlier search and ranking algorithms presented search results based on similarity of the metadata to the keywords in the query. Such ranking algorithms, though simpler to implement, can tend to be inaccurate and lead to the user searching through pages of search results to find their result of choice. Such similarity based algorithms were then replaced by Google's fabled PageRank algorithm which took into account not just similarity but also the quality of a document, measured through the number of links to that document. Such algorithms have then developed to include more complex factors such as the user's behaviour on the Internet, their location and tendency to click on certain links and other factors that the developers decide upon. Such approaches have greatly reduced the user's effort to find the right page. Conventional search services can only present the user with a list of results, rather than giving a deterministic answer.

With the advent of natural language processing, users have now attained a way to interact with devices using only their voice. As concluded by [2], speech interaction has proven to be the most enjoyable and convenient form of interaction in conversational systems. Such natural language processing based systems have led to the rise of popularity in devices that require the least amount of

visual or haptic input. Such devices mostly lack sufficient output space for a diversified list of search results for the user to choose from. Devices such as smart speakers, that lack any form of visual output, cannot present the user with more than one result for their queries. In these scenarios, the information presented by the device has to be as accurate and exact to the user's query as possible. Traditional search approaches cannot narrow down the search field sufficiently to do that. Even with BERT based models in use today to better understand the query, there are sufficient points of ambiguity that need to be addressed.

There have been multiple attempts to determine the ambiguity of search queries. All such attempts have mainly aimed at classifying the intent of the query into predetermined classes or into broad categories based on the functional aspects of the queries, such as informational, navigational or transactional [3]. Most of the attempts till date have utilised existing knowledge of query ambiguity in narrow domains. Work on ambiguity detection in open-domain conversations has been limited.

This thesis aims to address the issue of detecting the ambiguity in the initial user query in open-domain conversations by presenting and comparing two approaches. Both the approaches make use of graph convolutional networks [4] that make use of graphical information derived from the search results of the queries. The principal motivation behind the use of graphical information processing came from the fact that the ambiguity of a query can be defined in terms of the concepts covered by, and the similarities between the documents retrieved by the search operation, as explored by [5]. The mutual similarity score between those documents can play an important role in determining whether documents retrieved for the same query are closely related or cover entirely divergent subjects indicated by similar query terms. A graph convolutional network can process such interrelationship between those documents when expressed in the form of a graph, complete with node features, that represent some aspect of the content of the documents and an adjacency matrix that can describe the relationship between those documents, mainly in terms of their similarity. For the first approach, the node features consist of the vector representation of the documents, with the vectprizer weights fine-tuned as a part of the model training, and the edge weights determined by the cosine similarity between the document vectors. The second approach builds on the work done in [6] to determine similarity between news articles. In this approach, concepts derived from the documents are encoded and serve as the graph nodes, whereas the edge weights are determined by the similarity between documents associated with each concept.

This document is structured as follows:

- *Chapter 2* explores the existing literature relevant to the problem handled in this work.
- *Chapter 3* introduces all relevant background concepts, including the basic natural language understanding algorithms used.
- *Chapter 4* describes the resources used, including the tools and datasets required for this work.
- *Chapter 5* describes the proposed algorithms, the experiments performed and the results achieved.
- *Chapter 6* concludes on the explorations and proposes ideas for further improvement.

# Chapter 2

# Related Work

This chapter explores the existing literature pertaining to the detection of user intent and ambiguity therein, in various restricted as well as open settings.

The questions of detecting ambiguity in open-domain queries has been explored comparatively less in literature. An important approach for ambiguity detection involves the detection of the intent of the query. Broder, in his document "A taxonomy of web search" [3], introduced three broad categories of query intent: informational - for queries aimed at retrieving information, navigational - for queries aimed at navigating to a web page and transactional - intended at performing some activity facilitated by the web such as shopping or downloading a file. This taxonomy was further refined by Rose and Levinson [7] into navigational, informational and resource queries, with sub-categories of each. They used queries from the AltaVista search engine to develop a framework for categorizing queries. The sub-categories for the Informational queries include closed and open-ended queries that make room for ambiguity detection. For resolving conflicts in the assignment of a particular category, the authors have resorted to the user's behaviour with the search results. Such functional categorization of queries, however, does not provide any information about the level of ambiguity and depends on prior knowledge of the user's behaviour, which is not suitable for real time application. Examples of Rose and Levinson's taxonomy and samples from the AltaVista queries can be seen in Figure 2.1.

Building on the refined query taxonomy developed by Rose and Levinson [7], Lee et. al. [8] explored the possibility of developing an automatic user goal identification system based on user behaviour and the nature of web pages retrieved in the search results. They classified queries into navigational and informational queries, based on the premise that navigational queries will have users visiting only a particular intended website for the majority of the time and informational

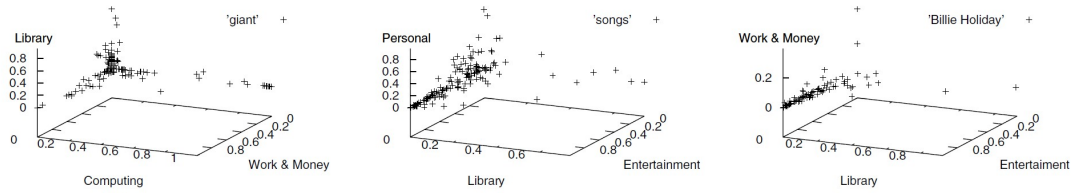
| SEARCH GOAL | DESCRIPTION | EXAMPLES |
|---|---|---|
| **1. Navigational** | My goal is to go to specific known website that I already have in mind. The only reason I'm searching is that it's more convenient than typing the URL, or perhaps I don't know the URL. | aloha airlines<br>duke university hospital<br>kelly blue book |
| **2. Informational** | My goal is to learn something by reading or viewing web pages | |
| 2.1 Directed | I want to learn something in particular about my topic | |
| 2.1.1 Closed | I want to get an answer to a question that has a single, unambiguous answer. | what is a supercharger<br>2004 election dates |
| 2.1.2 Open | I want to get an answer to an open-ended question, or one with unconstrained depth. | baseball death and injury<br>why are metals shiny |
| 2.2 Undirected | I want to learn anything/everything about my topic. A query for topic X might be interpreted as "tell me about X." | color blindness<br>jfk jr |
| 2.3 Advice | I want to get advice, ideas, suggestions, or instructions. | help quitting smoking<br>walking with weights |
| 2.4 Locate | My goal is to find out whether/where some real world service or product can be obtained | pella windows<br>phone card |
| 2.5 List | My goal is to get a list of plausible suggested web sites (I.e. the search result list itself), each of which might be candidates for helping me achieve some underlying, unspecified goal | travel<br>amsterdam universities<br>florida newspapers |
| **3. Resource** | My goal is to obtain a resource (not information) available on web pages | |
| 3.1 Download | My goal is to download a resource that must be on my computer or other device to be useful | kazaa lite<br>mame roms |
| 3.2 Entertainment | My goal is to be entertained simply by viewing items available on the result page | xxx porno movie free<br>live camera in l.a. |
| 3.3 Interact | My goal is to interact with a resource using another program/service available on the web site I find | weather<br>measure converter |
| 3.4 Obtain | My goal is to obtain a resource that does not require a computer to use. I may print it out, but I can also just look at it on the screen. I'm not obtaining it to learn some information, but because I want to use the resource itself. | free jack o lantern patterns<br>ellis island lesson plans<br>house document no. 587 |

Figure 2.1: Examples of AltaVista queries and their intents [7]. The authors further refined Broder's taxonomy [3]. The open category within the directed subcategory and the undirected queries mainly point to ambiguous informational queries, thus enabling detection of some level of ambiguity in user queries. Following their nomenclature, all the subcategories can have open and closed queries, thereby allowing for ambiguity detection for all types of queries. However, the level of ambiguity cannot be determined.

queries will have users visiting multiple websites from the search results to obtain the maximum amount of information on the topic. To obtain these data, they observed the distribution of user clicks on search results for individual queries. If the clicks had an even distribution over multiple search results, then it was probably an informational query, whereas if the clicks showed a bias towards one particular website, then the query was probably informational. Another aspect they took into account was the distribution of links that had the same anchor text as the query. The text for an informational query should not link to one particular website for the majority of the samples, whereas the same for a navigational query should always lead the user to one particular website. This approach, though automated, has limited applicability for open-domain conversations since the system will require a prior knowledge of click-through rates and distribution of search results for each query posed by the user.

Jansen et al. [9] utilised the taxonomy developed by Rose and Levinson [7] to automatically classify queries as informational, navigational or transactional based on web search logs from a search engine. They used a rule based approach to compare a set of features associated with each query, such as the user identifier, the search terms in the query and the type of content searched by the user, to a pre-defined set of class-based conditions that determined the category a particular query would belong to. These class-based conditions were derived from a qualitative analysis of available search logs and deriving features for manually classified queries. Though this procedure does not make use of user click-through data, it uses manually defined rules that may not be applicable to open-domain queries in a dynamic environment like the internet. Tamine et al. [10] combined user queries with context in order to determine intent. However, this approach relies on previous knowledge of similar queries with the same intent.

The approaches described in the previous paragraphs were the initial attempts at determining the actual intent of a query. However, all of them aimed at classifying queries into fixed categories, rather than determine the actual goal of the user. Since most of them rely on previous information about similar queries and about user behaviour on search results, they cannot be applied to open-domain coversational systems where the user's query is completely unpredictable and may not have any precedent. Additionally such attempts at query classification do not aid in detecting the level of ambiguity in the query. Although the taxonomy developed by Rose and Levinson [7] had a provision for classifying a query as ambiguous if it could not be placed into any of the classes, it does not describe any procedure to detect ambiguity within the individual categories.




(a) Type A: giant (b) Type B: songs (c) Type C: Billie Holiday

Figure 2.2: Clusters of documents retrieved for 3 queries [5]. the graphs show distribution of documents based on their probability of belonging to the predefined categories such as "Work & Money", "Computing" and "Library".

There have been multiple attempts at determining whether a query is ambiguous and quantifying the degree of ambiguity. Ruihua et al. [5] made one of the first attempts to determine the ambiguity of a query based on the documents retrieved by the search engine. The documents were projected on a three-dimensional space and features were derived to represent the distribution of the documents. Figure 2.2 shows an example of the clustering of documents retrieved for 3 queries. An SVM was used to classify queries as ambiguous based on the assumption that documents relevant to ambiguous queries would belong to multiple clusters.

Triennes et al. [11] envisioned the possibility of detecting unclear questions in a question-answering forum setting without the need to monitor user responses after the question had been posted. The authors made use of a database of similar questions that existed on the platform and had been classified as clear or unclear. A question was detected as unclear if clarifying questions had been posed to the author of that question and if the answer to the clarifying question added any new information. This approach, however, is limited by the presence of similar questions in the database.

Ammicht et al. [12] developed a novel system for detecting and resolving position ambiguity in natural language input. The system parses the user's input and prepares a notepad tree based on important information, such as the departure and arrival cities, in the case of a flight booking system, as shown in Figure 2.3. Then an application tree is prepared to keep track of values that can be derived from the inputs. The system, in effect, parses the inputs into a set of trees with attribute-value pairs and detects ambiguity in the information provided for each attribute in the notepad tree. Such a system is effective in determining the ambiguous nature of user inputs when the system knows what to expect from the

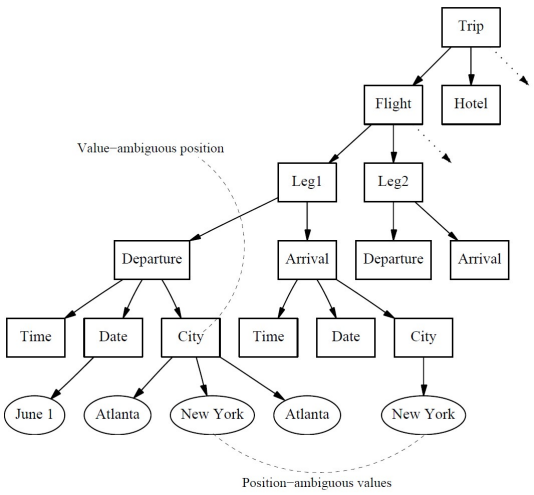



Figure 2.3: A notepad tree generated from user input to a flight reservation system. As seen in the penultimate level, the departure and arrival cities are unclear from what the user said. [12]

user, such as instructions pertaining to a particular domain. The system can be finetuned to any domain, but is not efficient for open-domain queries that may not follow any particular structure.

Hashemi et al. [13] developed a method to classify query intent using convolutional neural networks, based on the works of Collobert et al. [14] and Kim [15]. The query was first encoded using word2vec and then passed through a neural network with CNN layers to obtain a query vector representation. The query vector is then passed through a dense layer that serves as the intent classifier. The intents were obtained by manually classifying a dataset of queries. This method does not make use of search logs or results to identify intent but cannot be trained in the absence of pre-defined intent classes, which may be difficult to obtain for open-domain conversations.

# Chapter 3

# Background

This section explains the relevant background information required for sufficient understanding of the work in this thesis. It also builds up the groundwork and explains relevant work that have inspired the explored algorithms.

## 3.1 Artificial Intelligence

John McCarthy defines Artificial Intelligence as the "science and engineering of making intelligent machines, especially intelligent computer programs" [16]. The concept of AI entails the task of making a computer understand and mimic the process of human intelligence [17].

The birth of artificial intelligence can be attributed to Alan Turing's "Computing Machinery and Intelligence" [18] where he raised the question "Can machines think?". The "Turing Test" developed by him remains an important part of the history of artificial intelligence as it utilizes ideas around linguistics to test whether an interrogator can distinguish between a computer and a human respondent.

The most important aims of AI involve developing systems that think and act rationally like humans. It combines computer science and data to develop robust solutions to problems [17].

## 3.2 Machine Learning

Machine Learning is a subfield of artificial intelligence where data and algorithms are used to imitate the learning process of humans. Machine learning

lies at the intersection of computer science, statistics and artificial intelligence, where statistical methods are used on available data to develop algorithms that can output predictions and classifications. Machine learning plays an important role in detecting patterns in datasets that can help in decision making. Machine learning is a core component of all approaches towards achieving general artificial intelligence [19].

In the last two decades, the applications of machine learning have encompassed almost every application of computer science. Machine learning has been used successfully to derive information and patterns from huge sets of statistical data, which would not have been possible otherwise. Additionally machine learning algorithms can be designed to be multimodal, deriving inputs from a large variety of file formats and media, such as text, audio, video and images. These data can then be used to generate new sets of data, such as new text or speech or can be used to perform regression on the inputs or classification of the inputs. The essence of machine learning lies in the analysis of input data in order to derive a pattern that fits the data and then use that pattern to derive new data points.

Machine learning has been used extensively in natural language processing, automatic speech recognition, computer vision and other problems that cannot be solved in a reasonable amount of time or at all by traditional rule-based algorithms. Machine learning algorithms usually encompass four paradigms: supervised learning, semi-supervised learning, unsupervised learning and reinforcement learning. In the context of this thesis, supervised and unsupervised learning algorithms are of relevance and have been covered in the following section. The other paradigms are out of the scope of the current research.

### 3.2.1 Supervised Learning

Supervised learning can be defined as the task of learning a function that can map from inputs to outputs based on example input-output pairs [20]. A supervised learning algorithm is trained using a labelled dataset containing input-output pairs. The performance of the algorithm is then tested using a set of unlabelled data points. Supervised learning algorithms are used in many applications of machine learning, such as speech recognition, optical character recognition and object detection from images, among others. The inputs are converted into vectors, known as feature vectors before they are presented to the algorithm. The outputs or labels (also known as *supervisory signal*) are usually discrete or continuous values. In case of discrete labels, the task is known as a classification task and in case the labels are continuous in nature, the task is referred to as

regression.

In mathematical terms, the *training set* consists of a set of input-output pairs

$$S_{training} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\} \tag{3.1}$$

where $x_i$ is the feature vector corresponding to the $i$th input and $y_i$ is the $i$th target output or class. The objective of the supervised learning algorithm is to learn the mapping

$$f : X \to Y \tag{3.2}$$

where $x_i \in X, y_i \in Y$ for $i = 1, 2, 3, \ldots, N$.

The algorithm learns a function $h$ that can approximate the mapping $f$ and can also generalize over the entire training set $S_{training}$.

$$y_i^* = h(x_i) \tag{3.3}$$

The difference between the prediction $y_i^*$ and the target $y_i$ is termed the loss and determined by the loss function. The discrepancy between the actual output and that predicted by the algorithm can be defined in terms of mean squared error, cross-entropy loss and hinge loss, among others. The loss function is selected based on the type of output and the type of task at hand. Categorical cross-entropy is more suited as the loss function for a classification problem whereas mean squared error can be used for a regression problem.

The supervised learning algorithm is designed based on the suitability for the task at hand. From Bayesian regression to deep neural networks, the wide range of available algorithms reflects the wide range of problems that can be solved with machine learning.

The generalization achieved by the supervised learning algorithm is evaluated on a test set, which is a set of data points completely disjoint from the training dataset.

The main classifying algorithm in both the approaches presented in this thesis make use of the supervised learning paradigm.

### 3.2.2 Unsupervised Learning

Unsupervised learning can be defined as the task of deriving patterns in data without human supervision. Unsupervised learning tasks do not take target output values, only the unlabelled training data are input to the model. Unsupervised learning can derive previously unknown interesting information from available data.

Unsupervised learning algorithms are mostly concerned with clustering data points into groups based on similarity of features. The algorithm builds a rich internal representation of the data points which can then be used to generate imaginative data. The rich representation mainly consists of neuronal predilections and probability densities [21]. The principal goal of an unsupervised learning algorithm lies in estimating the *a priori* probability distribution of data,

$$p(x),\ x \in X,\ X \text{ is the training dataset} \tag{3.4}$$

whereas the goal of a supervised learning algorithm is to derive the *conditional* probability distribution of data,

$$p(x|y),\ x \in X, y \in Y,\ X \text{ is the training dataset},\ Y \text{ is the set of labels} \tag{3.5}$$

Traditional approaches to unsupervised learning include dimension reduction techniques like principal component analysis and clustering algorithms such as K-Means, DBSCAN and others. Dimensionality reduction techniques help reduce the number of features per data point by selecting the most important features such as those with the highest variance as in the case of principal component analysis. Clustering algorithms assign data points to regions or groups by calculating the similarity of each data point to group centroids.

Additionally unsupervised learning can be used in learning latent variable models, that help in anomaly detection and also in generative networks. Neural networks such as autoencoders are used for this purpose. Autoencoders can learn a latent representation of the input data by using the input as the target output and reducing the discrepancy between the latent representation and its origin. This latent representation can be used in detecting anomalous data points and also in generating new data using generative adversarial training that use the Expectation-Maximization algorithm [22].

In this thesis, one of the approaches uses K-Means clustering to derive concept clusters from document keywords.

## 3.3 Neural Networks

Inspired by neurons in the human brain, an artificial neural network is a computational model consisting of units called neurons that respond to inputs over a certain threshold value. Each neuron has a set of inputs and produces a single output. The output is determined by the *propagation function*. The output of one neuron is transmitted to another neuron through connections, each of which is assigned a weight to assess its relative importance. The weights are optimized during the training process. The output of the neuron is made up of the weighted sum of the inputs and a bias term. This output is called the *activation* of the neuron. The activation is then passed through a non-linear *activation function* to generate the output of the neuron.

Neurons are usually arranged in multiple layers. The neurons in each layer interact with neurons in other layers but never with other neurons in the same layer. The layer that determines the final output is the *output layer*. Between the input and output layers, there are multiple *hidden layers*. Usually, neurons in one layer are connected only to neurons in the immediately preceding and succeeding layers, but there are multiple patterns that can be used. *Dense* or *fully connected* layers are those in which each neuron connects to every neuron in the next layer. To reduce the number of connections and the number of neurons, *pooling* can be used, where multiple neurons in one layer connect to one neuron in the next layer.

Neural networks are trained using supervised methods where the output is compared with a pre-determined target and discrepancy is used to modify the weights and biases (the *model parameters*). The process of generating the output with a neural network is called *forward propagation*, whereas the process of adapting the parameters based on the loss is called *back propagation*. Back propagation effectively distributes the loss among the network connections. It calculates the gradient of the loss with respect to the weights and updates them using techniques such as stochastic gradient descent.

In mathematical terms, the inputs to the neural network consist of a set of feature-vectors,

$$X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\} \tag{3.6}$$

where $\mathbf{x}_i$ is a feature vector for the $i$th data point.

The input layer takes in each feature in its individual node. Considering a feed-forward network having $L$ dense hidden layers, with layer $L + 1$ being the

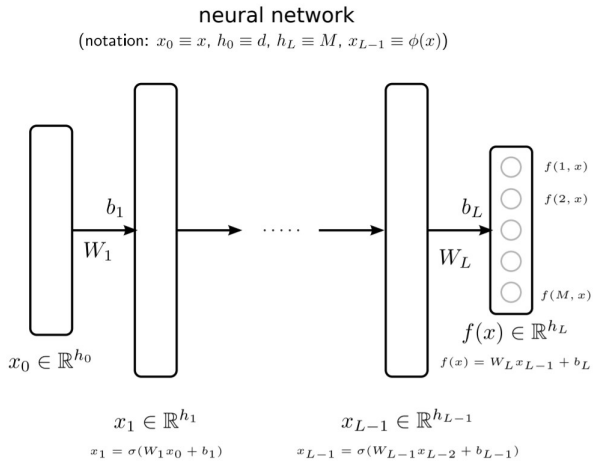



Figure 3.1: Example of a neural network. Here $x_0$ is the first input feature vector. The $L$th layer is the output layer and $f(x)$ is the complete function representing the neural network. [This image has been taken from the teaching resources of Prof. Marc Toussaint's 2019 course "Introduction to Machine Learning".]

output layer, the activation of the first hidden layer will be

$$\mathbf{a}_1 = W_1 \mathbf{x}_1 + \mathbf{b}_1 \tag{3.7}$$

where $W_1$ is the weight matrix for the first layer. $W_1$ contains the weights associated with all the connections between the input layer and the first hidden layer and $\mathbf{b}_1$ is the bias vector associated with the first hidden layer.This activation $\mathbf{a}_1$ is then passed through a non-linear activation function (such as ReLU, sigmoid etc.) to obtain the output of the first hidden layer,

$$\mathbf{z}_1 = \sigma(\mathbf{a}_1) \tag{3.8}$$

The output of the first hidden layer, $\mathbf{z}_1$ is then passed to the second hidden layer and the process continues till the output layer is reached. At the output layer,

$$\begin{aligned} \mathbf{a}_{L+1} &= W_{L+1}(W_L(\dots(W_1\mathbf{x}_1 + \mathbf{b}_1)\dots) + \mathbf{b}_L) + \mathbf{b}_{L+1} \\ \mathbf{z}_{L+1} &= \sigma(\mathbf{a}_{L+1}) \end{aligned} \tag{3.9}$$

Thus the output $\mathbf{z}_{L+1}$ is obtained on one forward pass.

Let the loss be $\ell$. To perform the back propagation pass, the algorithm calculates the gradient of the loss with respect to all the weights and biases,

$$\frac{\partial \ell}{\partial W_l} = \frac{\partial \ell}{\partial \mathbf{z}_{L+1}} \frac{\partial \mathbf{z}_{L+1}}{\partial \mathbf{a}_{L+1}} \frac{\partial \mathbf{a}_{L+1}}{\partial \mathbf{z}_L} \cdots \frac{\partial \mathbf{a}_l}{\partial W_l} \tag{3.10}$$

$$\frac{\partial \ell}{\partial \mathbf{b}_l} = \frac{\partial \ell}{\partial \mathbf{z}_{L+1}} \frac{\partial \mathbf{z}_{L+1}}{\partial \mathbf{a}_{L+1}} \frac{\partial \mathbf{a}_{L+1}}{\partial \mathbf{z}_L} \cdots \frac{\partial \mathbf{a}_l}{\partial \mathbf{b}_l} \tag{3.11}$$

The parameters are then updated using the calculated gradients. For example, using stochastic gradient descent,

$$W_l^* = W_l - \eta \frac{\partial \ell}{\partial W_l} \tag{3.12}$$

where $W_l^*$ is the updated weight matrix and $\eta$ is the *learning rate*, which determines the length of the step taken. The learning rate is an example of a *hyperparameter* which is a non-trainable parameter. Other hyperparameters include the number of layers in the network, the number of epochs for which to run the training and others. The algorithm used for updating the parameters is called the *optimizer*. Examples of optimizers include Stochastic Gradient Descent, Adam, RMSProp, among others.

## 3.4 Convolutional Neural Networks

Convolutional neural networks, first introduced by LeCun [23] are a special type of neural network that can process data having a known grid-type topology [24]. Such data include time-series data, such as sequential text which can be arranged in a grid or images, which can be viewed as a grid of pixels.

Convolution is a special mathematical operation which determines a weighted sum of all values in a particular window. The window is then shifted over the entire input grid and the output consists of the weighted sums obtained from all positions of the window over the input grid. For two functions $f$ and $g$, the convolution

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) g(t - \tau) d\tau \tag{3.13}$$

A convolutional neural network is one which uses this convolution operation in place of the usual matrix multiplication operation in at least one of its layers. The window based function that slides over the grid *input* is called the *kernel* and the output of a convolution layer is called *feature map*. In machine learning applications, the input and the kernel consist of multidimensional tensors. The parameters of the kernel are adapted through training.

As an example [24], the input may include a 2-dimensional image I for which we will need a 2-dimensional kernel K and the output of the convolution operation

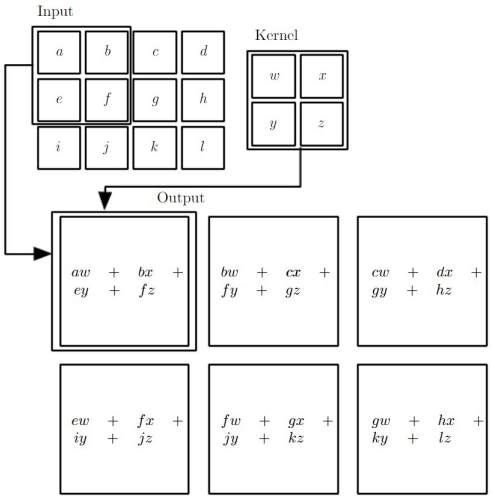



Figure 3.2: An example of a CNN operation [24]. In this example the kernel is applied to windows on the input image. Corresponding positions of the image and kernel are multiplied and the output is the sum of all the products. The kernel moves over the entire input image. In this case since the input image is not padded, the output feature map is smaller in size than the input image.

at position $(i, j)$ will be given by

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{3.14}$$

Convolution is commutative and hence

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, i-n)K(m,n) \tag{3.15}$$

Convolution involves three aspects which can improve a machine learning system [24]:

- **sparse interactions**: In classical neural networks, the interaction between each input and each output is defined by a set of parameters. Hence the number of parameters increases with increase in input-output combinations. On the other hand, convolution with a kernel smaller than the input can reduce the number of parameters by making one output correspond to a number of inputs. As in the case of image processing, this can help detect smaller, meaningful features from a large input image while reducing the

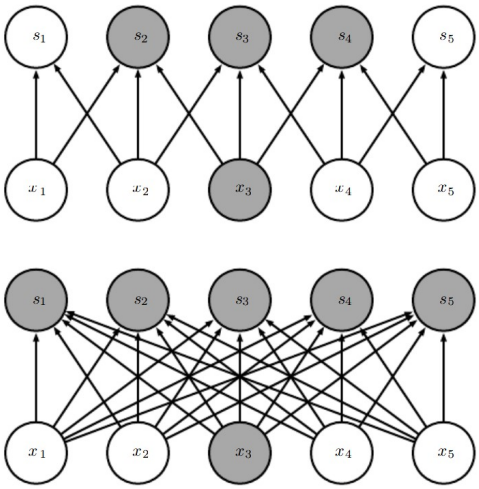



Figure 3.3: An example explaining sparse connectivity from [24]. The diagram on top shows the connections between input and output when a convolutional layer is used. Every input unit affects 3 output units when the kernel is of size 3. However, as seen in the diagram at the bottom, when matrix multiplication is used connectivity is dense and all outputs are affected by all input units.

number of input-output interactions and hence the memory requirement at the same time. The number of input units covered by one output unit is known as the *receptive field*.

- **parameter sharing**: Parameter sharing entails the use of the same set parameters for multiple input locations. In a convolution cell, the same kernel is used on all input positions and hence shared among multiple inputs, whereas in a traditional neural network, each input has its own fixed parameter that is used once during each forward pass. This reduces the memory requirement for the parameters by a large order.

- **equivariant representations**: *Equivariance* of a function suggests the ability of the function to change its output in the same way as its input. A function $f$ is said to be equivariant to another function $g$ if [24]

$$f(g(x)) = g(f(x)) \tag{3.16}$$

In the case of a convolution $f$, $g$ can be a translation operator that moves the input image. The equivariance of the convolution operator to the translation operator means that performing the convolution after shifting the image is equivalent to shifting the image and then performing the convolution. This is known as *translation equivariance*. In case of time series

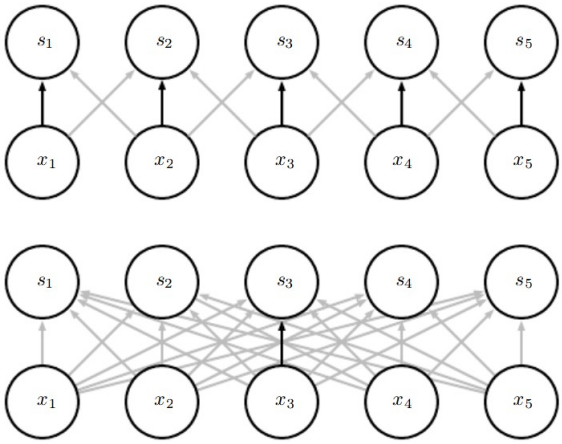



Figure 3.4: An example showing parameter sharing from [24]. The black arrow corresponds to a single parameter that is applied to the element at the centre of the window of size 3 in the top diagram. Due to parameter sharing, the same weight is shared across the center elements of all such windows throughout the input image. But in a fully connected model, each weight is used only once and not shared.

data, the output will correctly reflect the order of appearance of features. Convolution is however not equivariant to rotation or scaling [24].

To make the convolution operation *invariant* to translation and also to down-sample the feature maps, a technique called *pooling* is used. A pooling function replaces the output at a certain position in the feature map with a summary of the nearby outputs [24]. This summary is defined by the type of pooling layer used. For example, the max pooling layer replaces the output at each position with the maximum value from among the neighbouring outputs and a mean pooling layer replaces the output at every position with an average of the neighbouring outputs. The size of the neighbourhood is defined by the pooling window size set as a hyperparameter. Pooling helps to identify the existence of features irrespective of their location. The feature maps are additionally passed through a non-linear activation function in order to add non-linearity to the otherwise linear convolution operation.

Kim [15] proposed a model using a CNN to classify sentences. The model architecture, as shown in Figure 3.5 is a modified version of the model used by Collobert et. al. [14]. If a sentence has $n$ words and each word is encoded with a vector of length $k$, then the sentence can be represented in the form of an $n \times k$

matrix where each row contains the vector for a word. So a sentence can be represented as a concatenation of such vectors [15]:

$$\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \cdots \oplus \mathbf{x}_n \tag{3.17}$$

where $\oplus$ is the concatenation operator and $\mathbf{x}_i$ refers to the vector for the $i$th word. Let $\mathbf{x}_{i:i+j}$ represent concatenation of word vectors $\mathbf{x}_i, \mathbf{x}_{i+1}, \ldots, \mathbf{x}_{i+j}$ and let $\mathbf{w} \in \mathbb{R}^{hk}$ be a convolution kernel, where $h$ is the number of words covered by the kernel window at once. Applying convolution using this kernel, the output feature $c_i$ is a real number defined by

$$c_i = f(\mathbf{w} \cdot \mathbf{x}_{i:i+h-1} + b) \tag{3.18}$$

where $f$ represents the non-linear activation function and $b$ is the bias for the particular kernel. Applying this filter to every window of $h$ words in the text, the feature map $\mathbf{c}$ is generated [15]

$$\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \tag{3.19}$$

where $\mathbf{c} \in \mathbb{R}^{n-h+1}$. This feature map is subjected to a max-over-time pooling operation [14] that keeps the maximum value

$$\hat{c} = \max\{\mathbf{c}\} \tag{3.20}$$

as the feature for a particular kernel. This pooling scheme is also applicable to variable length sentences [15].

The CNN architecture presented by Kim has been used in the preliminary approaches presented in this work.

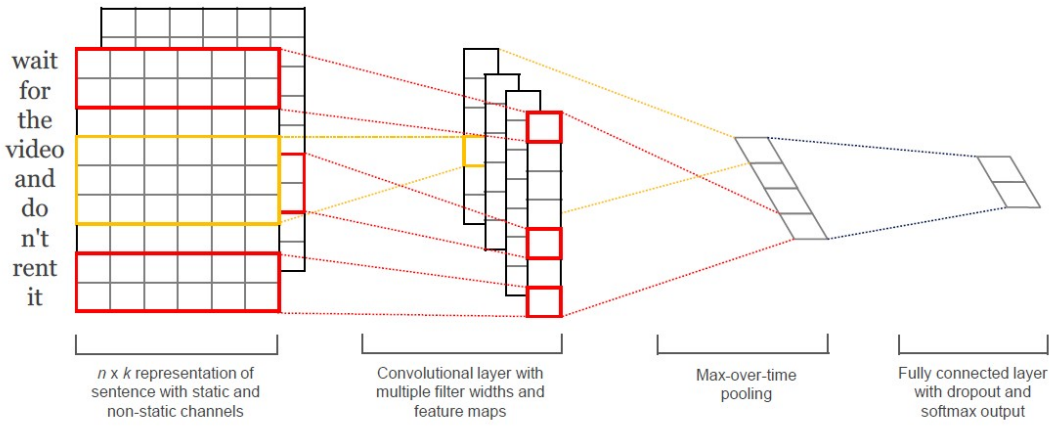



Figure 3.5: Convolutional network for sentence classification [15]

## 3.5 Graph Neural Networks

Graphs are a type of data structure which represents the relationship between objects (*nodes* or *vertices*). The nature and strength of those relations are encoded through *edges* between those nodes. A graph is said to be *undirected* if the edges do not bear any directional information and represent only pairs of nodes, or *directed* if each edge is associated with a direction and points from one node to another node. In mathematical terms, an undirected graph $G$ is defined as

$$G_{undirected} = (V, E) \tag{3.21}$$

where

- $V$ is the set of vertices
- $E \subseteq \{(x, y) \mid (x, y) \in V^2, x \neq y\}$ is the set of edges

A directed graph is defined as a triple

$$G_{directed} = (V, E, \phi) \tag{3.22}$$

where

- $V$ is the set of vertices
- $E$ is the set of edges
- $\phi : E \to \{(x, y) \mid (x, y) \in V^2, x \neq y\}$ is an incidence function that maps each edge to an ordered pair of vertices $(x, y)$

A weighted graph is one where each edge is assigned a weight based on some metric such as cost or importance. Each edge in a graph defines a relation between two vertices called an *adjacency relation*. Two vertices $\{x, y\}$ are said to be adjacent to each other if $(x, y)$ is an edge. This adjacency relation is encoded in a square adjacency matrix A such that

$$\begin{aligned} A_{ij} &= 1, \text{ if } (i, j) \in E \\ A_{ij} &= 0, \text{ otherwise} \end{aligned} \tag{3.23}$$

for an unweighted graph. For a weighted graph,

$$\begin{aligned} A_{ij} &= w_{ij}, \text{ if } (i, j) \in E \\ A_{ij} &= 0, \text{ otherwise} \end{aligned} \tag{3.24}$$

where $w_{ij}$ is the weight assigned to edge $(i, j)$. The degree matrix $D$ of a graph is a diagonal matrix such that

$$D_{ii} = \sum_j A_{ij} \tag{3.25}$$

Graphs are powerful data structures that can be used to represent relationships and interactions between real world objects. Graphs find application in social networks that model interactions among people, in mapping applications and also in chemical analysis of molecular structures.

Images can be considered as fixed size grid graphs where each central pixel in the kernel window is a node connected to neigbhouring pixels (nodes) covered by the window of the kernel. Hence CNNs can be thought of as graph networks operating on fixed grid graphs. Similarly, time sequence text data can be thought of as linear graphs. Thus it is evident that traditional neural network architectures are not well suited for processing arbitrarily shaped graphs, which is where graph neural networks come in. Graph neural networks are a class of neural networks that take graph data as input. For input to these networks, the graphs are described in terms of their *node features* and *adjacency matrices*. A node feature vector is a set of feature values that can uniquely identify a node. The edge information is obtained from the adjacency matrix and the degree matrix.

Typical applications of graph neural networks include node classification and graph classification. In a node classification problem, each node is associated with a ground truth class in the training data, whereas in the graph classification problem, each entire graph is associated with a target class in the training data.

The most commonly used graph neural networks are Graph Convolutional Networks (GCNs), Graph Attention Networks (GATs) and Message Passing Neural Networks (MPNNs). GCNs [4] are simple and powerful and can aggregate node and edge data using a linear transformation. Graph Attention Networks [25] improve on GCNs by learning the aggregation parameter during training. MPNN [26] have a more general update rule that gives more importance to edges.

Kipf and Welling have stated Graph Convolutional networks as the common architecture of all networks that accept arbitrarily shaped graphs [4]. These networks are referred to as convolutional as model parameters are shared across multiple positions on the graph. Such a network thus accepts a graph $G = (V, E)$, described in terms of

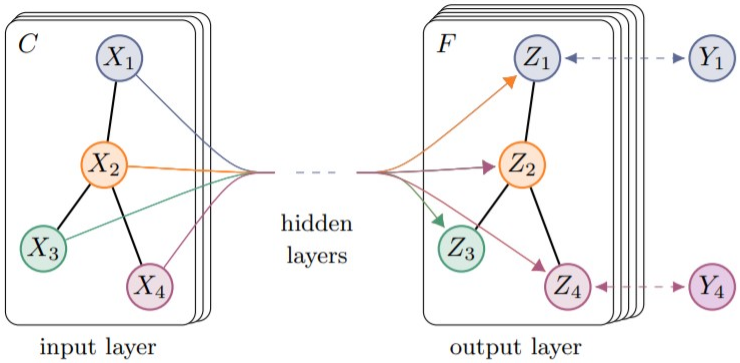



Figure 3.6: A graph convolution network with first order filters [27]

- Node feature vector $\mathbf{x}_i$ for each node i, accumulated in an $N \times D$ matrix $X$ where $N$ is the number of nodes and $D$ is the dimension of each feature vector.
- The graph structure and edge information are encoded in the form of an adjacency matrix $A$ of the shape $N \times N$. Any other equivalent graph representation can be used.

As output, the GCN produces a matrix of shape $N \times F$ where $F$ is the number of output node features. For outputs that consider the entire graph, a pooling operation can be used.

Mathematically, the GCN can be written as a function that takes as input the node feature matrix and the adjacency matrix and generates as output the node output feature matrix.

$$H^{(l+1)} = f(H^{(l)}, A) \tag{3.26}$$

with $H^{(0)} = X$ and $H^{(L)} = Z$, $Z$ being the node output feature matrix and $L$ being the number of GCN layers. The propagation function $f$ as defined in [26] is a simple but powerful one.

$$f(H^{(l)}, A) = \sigma(AH^{(l)}W^{(l)}) \tag{3.27}$$

where $\sigma$ is a non-linear activation function like ReLU. $W^{(l)}$ is the weight matrix for the $l$th layer. Multiplying the node feature vector with the adjacency matrix aggregates all the relationships for each node, weighted by the strengths of the relationships if the graph is weighted. To include the context of the node itself, $A$ is usually added to an identity matrix. Additionally, since the adjacency matrix is not normalized, multiplying with $A$ at every step can cause the weights to

explode. To solve this, $A$ is subjected to symmetric normalization using the degree matrix $D$. Combining all the steps, the forward propagation function of the GCN looks like

$$f(H^{(l)}, A) = \sigma(D^{-\frac{1}{2}}\hat{A}D^{-\frac{1}{2}}H^{(l)}W^{(l)}) \tag{3.28}$$

where $\hat{A} = A + I$ is the adjacency matrix with self loops.

Graph Convolutional Networks have been used in both the approaches presented in this work. The other two types of GNNs are beyond the purview of this research.

## 3.6 Transformers

Sequential text data are a time-series data that need to be understood thoroughly by the machine learning model in order to act upon it or classify it. Traditional ML approaches involving CNNs and recurrent neural networks are not able to handle long range dependencies between words in the input text on their own. To solve this, the concept of attention was developed. Attention mechanisms work with traditional RNNs and help in highlighting important parts of the input text and thus handle dependencies. Transformers build upon the concept of *self-attention*, which involves applying the attention mechanism and thus discovering dependencies within the same sequence in order to create an efficient representation of the sequence.

An attention function is defined as a mapping from a query and a set of key value pairs to an output, all the inputs and outputs being vectors [28]. Mathematically, with input $X$,

- *Query vector*: $q = XW_q$. $W_q$ is the weight matrix for generating the query vector. Query vector represents the current word.
- *Value vector*: $v = XW_v$. $W_v$ is another weight matrix. $v$ can be thought of as representing the information contained in the word.
- *Key vector*: $k = XW_k$, $W_k$ being the corresponding weight matrix. The key vector can be though of as an indexing mechanism for the value vectors.

For a query $q$, the most similar key $k$ is obtained through a dot product of the vectors $q$ and $k$. This is similar to the cosine similarity measure as the most similar key-query combination will have the highest dot product. A softmax operation is applied on this dot product and multiplied with the value vector

$v$. Higher the dot product, higher will be the attention given to that value. The weight matrices $W_q, W_k, W_v$ are trained with the model. Mathematically, self-attention can thus be represented as [28]

$$\text{Attention}(Q, K, V) = \text{softmax}\Big(\frac{QK^\top}{\sqrt{d_k}}\Big)V \tag{3.29}$$

where $Q, K$ and $V$ are the matrices containing the query, key and value vectors, and $d_k$ is a scaling factor equal to the dimension of the key vector. This is also known as *scaled dot-product attention*.

An important variant of the self-attention used in the transformer architecture is the concept of *multihead* attention. Multihead attention involves projecting the keys, values and queries to multiple learned linear projections and then applying the attention functions on each of these projections [28]. Multihead attention enables sourcing information from multiple positions of the input in multiple representations. Mathematically [28],

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)W^O \tag{3.30}$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$. The linear projections are determined by the parameter matrices $W_i^Q, W_i^K, W_i^V$ and the output parameter matrix $W^O$. The transformer architecture consists of a combination of an encoder and decoder, each being made up of stacks of self attention layers and fully connected layers [28]. The *encoder* is made of a stack of 6 identical layers, each of which consists of two sublayers: one with the multihead attention module and the other with the position-wise fully connected layers. Each sublayer has a residual connection around it, making the output of each sublayer [28]

$$\text{LayerNorm}(x + \text{Sublayer}(x)) \tag{3.31}$$

where $\text{Sublayer}(x)$ is the output of the sublayer itself. The position-wise fully connected layer is a feed-forward network applied to each position in the input separately and identically [28]. This can be defined as a combination of two linear operations,

$$\text{FFN}(x) = max(0, xW_1 + b_1)W_2 + b_2 \tag{3.32}$$

The parameters change from layer to layer. The *decoder* has a similar construction but adds a third sublayer in each of its modules - a multihead attention over the output of the encoder. The self-attention sublayers in the decoder are masked appropriately and the output embeddings are offset by one position to ensure that the predictions for a particular position can only depend on the known outputs for positions preceding it, not the subsequent positions.

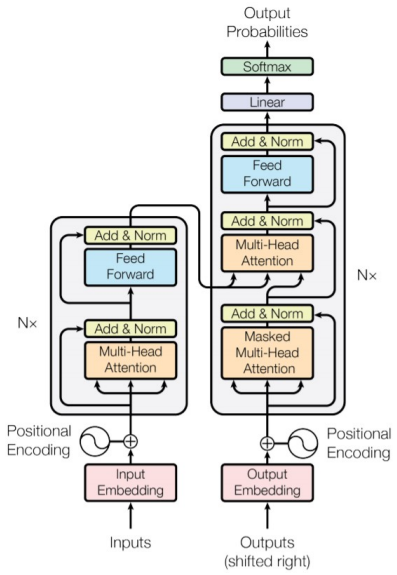



Figure 3.7: The architecture of a transformer with the encoder and decoder stacks [28].

Lastly, the absence of recurrence or convolution makes the model ignorant of positional information in the input. To make use of sequential data, positional information is added in the form of *positional encodings*, which have the same dimension as the input embeddings. The encodings used by the authors are of the form [28],

$$\begin{aligned} \mathrm{PE}_{(pos,2i)} &= \sin(pos/1000^{2i/d_{model}}) \\ \mathrm{PE}_{(pos,2i+1)} &= \cos(pos/1000^{2i/d_{model}}) \end{aligned} \tag{3.33}$$

where $pos$ is the position and $i$ is the dimension. $d_{model}$ is the dimension of the input embeddings.

## 3.7 BERT

BERT, short for Bidirectional Encoder Representations from Transformers is a language model introduced by Devlin et. al. [29] in 2019. It is designed to learn rich bidirectional representations from unlabelled text by taking into account both left and right context in all layers [29].

The BERT framework has two steps:

- *Pre-training*, when the model is trained over unlabelled data,

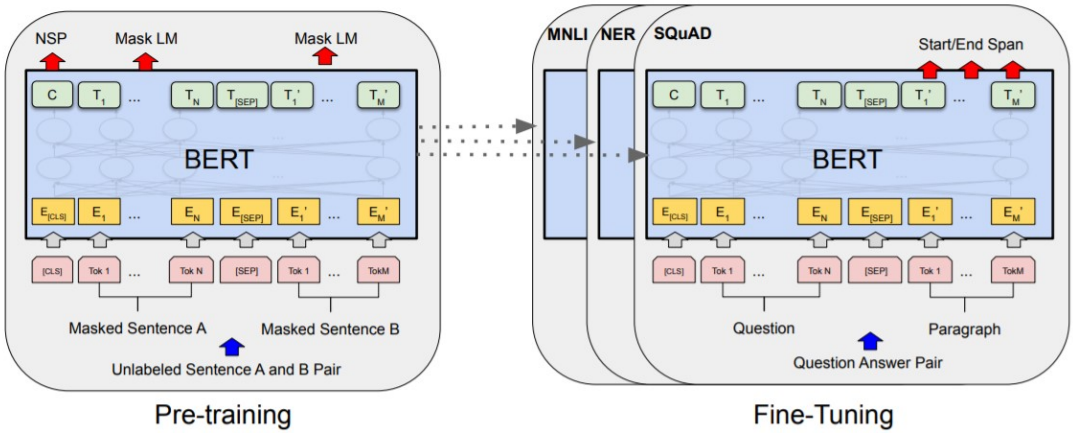



Figure 3.8: The outline of the pre-training and fine-tuning procedures of BERT. Only the output layers need to be changed based on the end goal of the downstream task. The rest of the architecture is preserved. The [CLS] token is used to mark the beginning of an input example and the [SEP] token is used to separate question and answer inputs when the downstream task requires pairs of inputs [29]

- *Fine-tuning*, when the model is initialized with the pre-trained parameters and then fine-tuned using labelled data for the particular task at hand.

BERT has a unified architecture across different tasks, which minimal changes to the model needed when the downstream task changes. The model architecture is a bidirectional transformer encoder based on the model described in the previous section. BERT can handle, as input, both a single sentence and a pair of sentences to accommodate for question-answer tasks. Every sentence is delimited by a special start token [CLS] and the final hidden state corresponding to this token is used as the aggregate sequence representation [29]. BERT is pre-trained in two steps:

- *Masked Language Model*: Some portions of the input tokens are masked at random and the model is made to predict those masked tokens.
- *Next Sentence Prediction*: This step makes sure the model understands sentence relationships which are essential for question-answering and natural language understanding tasks.

There have been multiple modifications to and improvements over the original architecture and training methods used by BERT. Three such models have been used during the experiments in this work: RoBERTa, ELECTRA and SBERT.

### 3.7.1 Roberta

RoBERTa was introduced in 2019 by Liu et. al. [29]. They achieved better results than the original BERT implementation by making some changes in the pre-training process:

- For the Masked Language Model part of the pre-training, the original BERT implementation generated the masks during data preprocessing. Though the data was replicated to obtain multiple combinations of masked tokens, they were static in nature. Liu et. al. replaced this approach with *dynamic masking* where the masking pattern was generated every time a sequence was fed to the model [29].

- The next sentence prediction loss was removed and instead the model was trained with blocks of sentences across multiple documents.

- The perplexity of the model was increased by training with larger batch sizes.

- Finally, whereas the original BERT implementation used a character level Byte Pair Encoding technique [30] with a vocabulary of size 30000, Liu et. al. used a BPE encoding that used bytes in place of unicode characters as units, with a vocabulary size of 50000, following the approach by Radford et. al [31].

These modifications help RoBERTa achieve state-of-the-art results and outperform the original BERT on evaluation tasks.

### 3.7.2 ELECTRA

The basic concept behind ELECTRA [32] aims at solving one of the shortcomings of BERT with regard to the masking of tokens in the MLM stage of the pre-training. This masking is done in the preprocessing stage and thus requires a large number of training samples to be effective and to achieve multiple combinations of these masked tokens. The authors of [32] have proposed another method to achieve this, without the need to have large number of samples. In the ELECTRA pre-training, instead of masking the tokens, they are replaced with tokens generated by a small generator network [32] and the model is trained to identify those replaced tokens. This is similar to a generative adversarial training, where a generator generates the tokens and a discriminator decides for each token whether it is an original one or a replaced one. According to the authors [32], since this task is defined over all tokens and not only the masked ones, the process requires much less input samples.

The generator $G$ and disriminator $D$ networks in ELECTRA are made of a Transformer encoder to map a sequence of input tokens $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ to vector representations $h(\mathbf{x}) = [h_1, h_2, \ldots, h_n]$. For a given position in the input, $t$, the generator determines the probability of using a token $x_t$ [32]

$$p_G(x_t|\mathbf{x}) = \exp(e(x_t)^\top h_G(\mathbf{x})_t) / \sum_{x'} \exp(e(x')^\top h_G(\mathbf{x})_t) \tag{3.34}$$

where $e$ denotes token embeddings. In this case $t$ only represents positions that are to be masked. On the other hand, for every position $t$, the discriminator $D$ determines whether the token is real or replaced by the generator:

$$D(\mathbf{x}, t) = \text{sigmoid}(w^\top h_D(\mathbf{x})_t) \tag{3.35}$$

where $w$ represents the discriminator weights.

The generator performs the masked language modelling task similar to the BERT pre-training. For this, a random set of tokens are replaced by the [MASK] token and the generator is trained to identify those masked tokens,

$$\mathbf{x}^{masked} = \text{REPLACE}(\mathbf{x}, \mathbf{m}, [\text{MASK}]) \tag{3.36}$$

where $\mathbf{m} = [m_1, \ldots, m_k]$ are positions chosen randomly to mask. The discriminator is trained to identify whether a token has been replaced by a non-original toke by the generator. The input to the discriminator is a corrupted sample $\mathbf{x}^{corrupt}$. The inputs are constructed by the following distributions [32]

$$\begin{aligned} &m_i \sim \text{unif}\{1, n\} \text{ for } i = 1 \text{ to } k \\ &\mathbf{x}^{masked} = \text{REPLACE}(\mathbf{x}, \mathbf{m}, [\text{MASK}]) \\ &\hat{x}_i \sim p_G(x_i|\mathbf{x}^{masked}) \text{ for } i \in \mathbf{m} \\ &\mathbf{x}^{corrupt} = \text{REPLACE}(\mathbf{x}, \mathbf{m}, \hat{\mathbf{x}}) \end{aligned} \tag{3.37}$$

The loss functions are as follows [32]:

$$\begin{aligned} \mathcal{L}_{MLM} &= \mathbb{E}\Big( \sum_{i \in \mathbf{m}} - \log p_G\big(x_i|\mathbf{x}^{masked}\big) \Big) \\ \mathcal{L}_{Disc} &= \mathbb{E}\Big( \sum_{t=1}^{n} -\mathbf{1}(x_t^{corrupt} = x_t) \log D(\mathbf{x}^{corrupt}, t) - \\ &\qquad \mathbf{1}(x_t^{corrupt} \neq x_t) \log(1 - D(\mathbf{x}^{corrupt}, t)) \Big) \end{aligned} \tag{3.38}$$

The combined loss function is

$$\min_{\theta_G, \theta_D} \sum_{\mathbf{x} \in \mathcal{X}} \mathcal{L}_{MLM}(\mathbf{x}, \theta_G) + \lambda \mathcal{L}_{Disc}(\mathbf{x}, \theta_D) \tag{3.39}$$

where $\mathcal{X}$ is a large corpus of raw text, $\theta_G$ and $\theta_D$ are the trainable parameters for the generator and discriminator respectively. The principal difference of this method with a generative adversarial training procedure is that tokens correctly identified by the generator are not considered fake and the generator is trained using maximum likelihood rather than being trained adversarially.

### 3.7.3 SBERT

SBERT develops on BERT and RoBERTa's capability to encode sentence pairs with an aim to determine similarity between a pair of sentences. The original BERT implementation takes both sentences as input which can lead to a huge computational overhead if there is a need to find similar pairs of sentences from a collection [33]. SBERT was presented as a modification of the original BERT using siamese network architecture to derive sentence embeddings to be used with the cosine similarity measure.

The SBERT architecture adds a pooling layer to the output of the BERT layer in order to calculate a fixed length vector for an entire sentence. The pooling layer is either a mean pooling layer or a max pooling layer. Three types of objective functions were proposed based on the end task:

- *Classification Objective Function*: The sentence representations $u$ and $v$ are concatenated with the elementwise difference $|u - v|$ and multiplied with a weight matrix $W_t$ before a softmax layer is applied.

$$o = \text{softmax}(W_t(u, v, |u - v|)) \tag{3.40}$$

- *Regression Objective Function*: This function outputs the cosine similarity between the sentence vectors $u$ and $v$.

- *Triplet Objective Function*: This objective function requires three additional inputs: an anchor sentence $a$, a positive sentence $p$ and a negative sentence $n$. The triplet loss optimizes the network to ensure that the distance between $a$ and $p$ is smaller than that between $a$ and $n$. The following loss function is minimized:

$$\max(\parallel s_a - s_p \parallel - \parallel s_a - s_n \parallel + \epsilon, 0) \tag{3.41}$$

  where $s_x$ represents the sentence embedding for the anchor, positive or negative sentence, $\parallel \cdot \parallel$ is the distance metric used and $\epsilon$ is a margin.

SBERT is used in this work to obtain representations of the initial query input by the user and of document keywords to derive concept representations.

## 3.8 Text Embedding Techniques

Natural Language Understanding involves understanding the text that a user speaks or writes. For humans, understanding that input comes naturally from years of training. But computer algorithms cannot understand natural language in its native form. Moreover since machine learning algorithms are statistical procedures, the input must be represented as vectors that can capture the semantic relationship between words and sentences. Such representations are called *embeddings*. Embeddings are vector representations of words that not only convert text to real-values but also ensure that semantically similar words have similar representations.

One way to vectorize words is to use one-hot encodings that identify the position of a word in the vocabulary. However, this produces sparse vectors that do not work well with neural networks [34]. Dense representations are able to generalize better and are able to capture similarities better.

Two such word embedding techniques have been used in this work: Word2vec and TF-IDF vectors.

### 3.8.1 Word2vec

The word2vec model was introduced in 2013 by Mikolov et. al. [35]. The paper presented two models that can learn rich word representations from large datasets, with the vectors having multiple degrees of similarity [35]. Word2vec can utilize either of two proposed architectures to produce word representations. In the first model, *Continuous Bag of Words*, the model is made to predict the current word from a window of surrounding words. The bag-of-words assumption applies, which states that the order of the words does not influence the prediction. The second model, *Continuous Skip-gram*, predicts the surrounding window of context words using the current word. This model attaches more weight to nearby words than distantly located words. CBOW is said to be faster whereas skip-gram produces better representations for infrequent words [35].

An improvement over word2vec is **doc2vec**, which uses the underlying word2vec models to predict a single fixed-size vector for an entire document consisting of multiple words [36]. The doc2vec model extends the word2vec CBOW architecture by adding a document id vector to the CBOW inputs. Thus the CBOW model not only uses a set of context words to predict a particular word, but also the document id of the document which contains this set of words. The

document id is random at first and is trained with the rest of the word representations. While the word vectors represent word concepts, the document vector is intended to vectorize an entire document. Another approach proposed by the authors was a Distributed Bag of Words model where the input is the document vector and the model is trained to predict a randomly sampled window of words from the document.

### 3.8.2 TF-IDF

TF-IDF, in information theory refers to a statistical measure of the importance of a word in a document in a corpus. The TF-IDF value increases with the number of times a word appears in a document and decreases with the number of documents in the corpus that have that word. Thus it gives more importance to words that appear frequently in a document but can identify that document since it does not appear frequently in other documents in the collection.

TF or term frequency measures the frequency of a word in a document. This measures the generality of a word in a document and is normalized to offset the effect of longer documents. Mathematically, the TF value is calculated as

$$\mathrm{tf}(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \tag{3.42}$$

where $t$ is a term, $d$ is a document in the corpus and $f_{t,d}$ is the raw count of the number of times $t$ appears in $d$. While vectorizing a set of documents, the vocabulary or *vocab* consists of all the words in the corpus. The TF value for each word in the vocab will lie between 0 and 1, both inclusive. The TF will be 0 if the word does not appear in the document under consideration.

IDF or inverse document frequency measures the information content of a term $t$. The IDF value is very low for commonly occurring words like stop words. It is calculated as

$$\mathrm{idf} = \log \frac{N}{1 + |\{d \in D : t \in d\}|} \tag{3.43}$$

where $N$ is the total number of documents in the corpus and $|\{d \in D : t \in d\}|$ is the number of documents the term appears in. The denominator is also known as the *Document Frequency*(DF). The denominator has 1 added to it to account for terms that do not appear in the corpus.

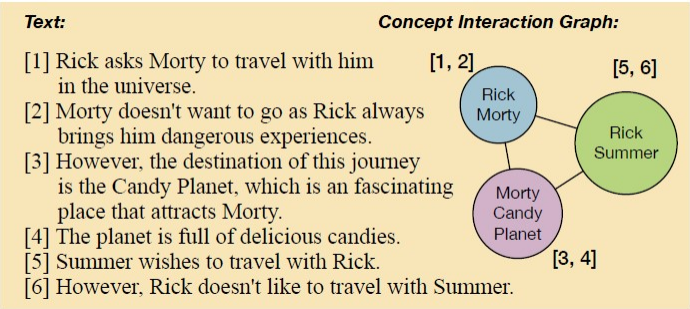



Figure 3.9: A piece of text with its sentences and the corresponding concept interaction graph. Each node is formed by the concepts covered in the sentences and the numbers associated with each vertex represents the sentence numbers that contain that concept [6].

The TF-IDF value is calculated by

$$\text{tf-idf}(t, d) = \text{tf}(t, d) \cdot \text{idf}(t) \tag{3.44}$$

A high TF-IDF value is obtained by words that occur frequently in one particular document and infrequently throughout the corpus. The TF-IDF vectorizer represents a document as a vector containing the TF-IDF values of the words in the vocabulary, specific to that document. Thus the length of the vector is fixed to the number of words in the vocabulary.

## 3.9 Document similarity detection using TF-IDF and GCN

Bang et. al. developed a method using TF-IDF vectors and graph convolutional networks to determine similarity between pairs of news articles [6]. The algorithm develops a Concept Interaction Graph based on the concepts covered in the articles. The concepts are derived from the keywords found in the articles.

The Concept Interaction Graph is defined as an undirected weighted graph which describes a particular document as a graph of a subset of its sentences. Mathematically, a document $D$ generates a graph $G_D$ where each vertex of $G_D$ represents a concept covered in the document. Each concept has a set of sentences associated with it. This concept is found in every sentence in this set [6]. Figure 3.9 shows a generic example of this graph.

The process to construct this concept interaction graph involves the following steps [6]:

- *KeyGraph*: As a first step, the KeyGraph is constructed using the named entities and keywords extracted from the document $D$ using the TextRank algorithm [37]. In the KeyGraph, each vertex has a keyword and each edge between two vertices represents the fact that the keywords in those vertices occur in the same sentence.

- *Concept Detection*: Since documents may have a large number of keywords, concepts can be derived from the KeyGraph using highly connected subgraphs of the same [6]. A highly connected subgraph will imply that those keywords always occur together and are hence correlated. This step divides a KeyGraph into a set of communities

$$C = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_{|C|}\} \tag{3.45}$$

  where each community $\mathcal{C}_i$ contains the keywords clustered within that concept. The authors have used betweenness centrality score algorithm [38] to derive the communities.

- *Sentence Attachment*: This step associates sentences with the concepts derived in the previous step. The sentences are represented by the TF-IDF vectors calculated from their words and the concepts are represented by the TF-IDF vectors of the keywords in the concept, concatenated to form a single piece of text. Each sentence is assigned to the concept which it shares most similarity with. The similarity is calculated in the form of the cosine similarity score between the TF-IDF vectors. A dummy vertex is created to accommodate sentences that cannot be associated with any concept.

- *Edge Construction*: After sentences are associated with the concepts, each concept is represented by a TF-IDF vector of the concatenation of the sentences associated with it. The edge weight between any two concept vertices is determined by the similarity between the TF-IDF vectors of those vertices.

For two documents $D_A$ and $D_B$, the steps mentioned above generate two concept interaction graphs $G_A$ and $G_B$. For comparing these two documents, the graphs are merged. For each common vertex pair between $G_A$ and $G_B$, the sentence sets of the vertices in that pair are merged and the merged graph $G_{AB}$ is obtained [6].

The next step involves the use of Graph Convolutional Layers [4] to match the two articles in question. From the merged graph $G_{AB}$, the model learns a matching vector for each vertex $v \in G_{AB}$ in the graph such that the learned vector can represent the semantic relationship between $S_A(v)$ and $S_B(v)$ which are the sets of sentences associated with vertex $v$ from documents $D_A$ and $D_B$ respectively [6]. The match vector $\mathbf{m}_{AB}(v)$ is generated by a siamese encoder operating on the word embeddings of the sentences associated with each vertex $v$. This siamese encoder takes the word embeddings of the sentence sets associated with the vertex and converts them into a context vector through RNN, LSTM or convolutional layers [6]. For $S_A(v)$ the context vector $\mathbf{c}_A(v)$ is obtained and for $S_B(v)$, the context vector is $\mathbf{c}_B(v)$. Taking into account the context of the encoded sentences from the two documents, an aggregation layer concatenates the elementwise abolute difference and the elementwise product of the two context vectors [6] to generate the matching vector.

$$\mathbf{m}_{AB}(v) = (|\mathbf{c}_A(v) - \mathbf{c}_B(v)|, \mathbf{c}_A(v) \circ \mathbf{c}_B(v)) \tag{3.46}$$

where $\circ$ represents the elementwise Hadamard product of two vectors. Apart from this matching vector, term based similarities between the document sets are calculated based on TF-IDF vector similarities, cosine similarities of the TF values, cosine similarity from BM25 scores, Jaccard similarity of 1-grams and values of the Oichai similarity measure [6]. Concatenating these measures, another matching vector $\mathbf{m}'_{AB}(v)$ is obtained.

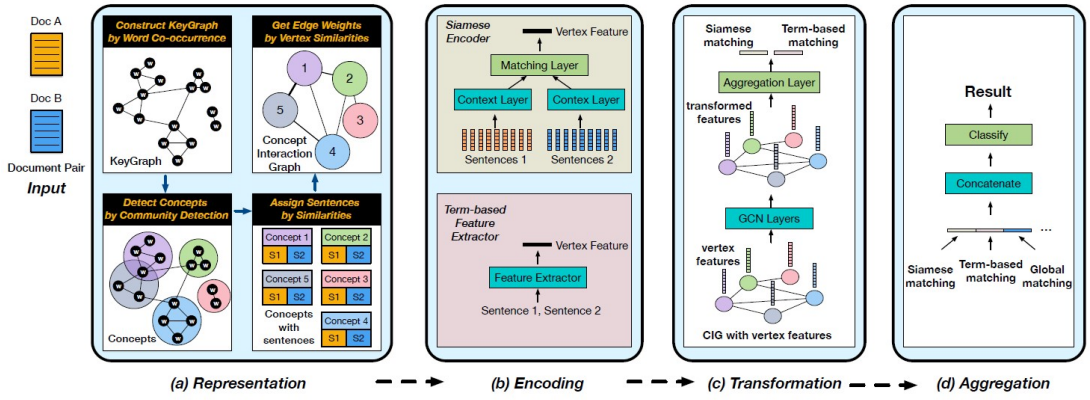



Figure 3.10: Overview of the approach used by [6]. In the Representation phase, the concept interaction graph is generated. The Encoding phase calculates the matching vectors which are then passed through the GCN in the Transformation phase. The Aggregation phase generates the final matching score from the output of the GCN.

After the context and matching vectors are generated, GCN layers [4] are used to aggregate all the information into a similarity score. The merged concept graph $G_{AB}$ is used as input to the GCN, but in this case, each concept is represented by a concatenation of the two matching vectors $\mathbf{m}_{AB}(v)$ and $\mathbf{m}'_{AB}(v)$. The output of the last hidden GCN layer is passed through a mean pooling layer that takes the mean of the last layer hidden vectors of all vertices. The output of this pooling layer is then passed through dense layers to obtain the final matching score [6]. The entire procedure is described in Figure 3.10.

The second approach presented in this work is inspired by this model and uses the idea of the Concept Interaction Graph.

# Chapter 4

# Resources

This chapter describes the datasets used in this work and the various libraries and frameworks used for specific purposes to complete the experiments.

## 4.1 Data

The work in this thesis is inspired by the ClariQ challenge organized by the University of Amsterdam to develop algorithms for generating clarifying questions in response to ambiguous search queries [39]. The first part of the challenge involves generating a system to determine the level of ambiguity in the initial search query. The level of ambiguity is quantized into 4 classes: 1 referring to a clear query and 4 referring to a query that is completely ambiguous. The complete clarifying question generating system must ask a question when the initial query is deemed ambiguous. Among the datasets provided by the challenge organizers, the training dataset, development dataset, along with a list of documents derived for each query from the ClueWeb datasets were used for this work.

### 4.1.1 Training Dataset

The main training dataset is a TSV file containing the following columns:

- *topic_id*: Each unique query is identified by a topic_id. The training dataset contains 187 such unique topic_ids.
- *initial_request*: The initial query of the user that starts the conversation and the query whose ambiguity is to be determined in this work.
- *topic_desc*: A complete description of the topic from the TREC Web Track data.

- *clarification_need*: The measure of query ambiguity on a scale of 1 to 4. The contents of this column serve as the ground truth labels during training the models proposed in this work.
- *facet_id*: An ambiguous query may have multiple end goals. Each facet describes one such end goal of the user. A facet is identified by a unique facet_id. An initial query may have multiple facets associated with it, i.e. each topic_id can correspond to multiple facet_ids.
- *facet_desc*: A complete description of the facet from the TREC Web Track data.
- *question_id*: The ID of the proposed clarifying question as it appears in a question bank provided by the organizers.
- *question*: The question corresponding to the question_id, determined by the initial query and the facet.
- *answer*: The user's answer to the posed clarifying question.

This dataset is based on the Qulac dataset [40] which in turn draws the initial queries and facets from the TREC 09-12 Web Track data. TREC is a conference for information retrieval methods and the Web Track concerns the retrieval of information of data from the web based on user queries. The Qulac dataset collected the initial queries and facets from the TREC Web Track dataset and then crowdsourced the clarifying questions, following refinement of those questions. Finally the authors collected user responses to each intial query-facet-clarifying question triplet.

Examples of the structure of the training dataset are shown in Table 4.2. The table shows an initial query "I'm interested in dinosaurs". Though the end goal of the user is to visit the Discovery Channel's website for dinosaurs, there are more than one directions the conversational system may take. This example shows two such counter-questions that may be generated by the system to clarify the user's intent. Additionally, as is evident, the statement "I'm interested in dinosaurs" does not say anything about what the user wants to search. Hence this query has the maximum ambiguity class of 4.

Table 4.1 shows the distribution of datapoints of various ambiguity level in the training data. The training dataset is itself small, and as is evident from the distribution of data points in the training dataset, queries with the extreme ambiguity levels 1 and 4 are much less in number than those with ambiguity levels 2 and 3.

| **Level of ambiguity** | **Number of queries** |
|---|---|
| 1 | 25 |
| 2 | 74 |
| 3 | 62 |
| 4 | 26 |

Table 4.1: Distrbution of ambiguity classes in the training data.

| topic_id | initial_request | topic_desc | clarification_need | facet_id | facet_desc | question_id | question | answer |
|---|---|---|---|---|---|---|---|---|
| 14 | I'm interested in dinosaurs | I want to find information about and pictures of dinosaurs | 4 | F0159 | Go to the Discovery Channel's dinosaur site, which has pictures of dinosaurs and games. | Q00173 | are you interested in coloring books | no i just want to find the discovery channels website |
| 14 | I'm interested in dinosaurs | I want to find information about and pictures of dinosaurs | 4 | F0159 | Go to the Discovery Channel's dinosaur site, which has pictures of dinosaurs and games. | Q03021 | which dinosaurs are you interested in | im not asking for that i just want to go to the discovery channel dinosaur page |

Table 4.2: Examples of training data.

### 4.1.2 Development Dataset

The development dataset is usually used for tuning hyperparameters and evaluating the performance of the model during training. This dataset is used to calculate the performance metrics of the model after every training epoch. The structure of the dev dataset is exactly similar to the training dataset as explained in the previous section. The distribution of ambiguity classes is shown in Table 4.3. As in the training dataset, the number of samples of classes 1 and 4 is less compared to that of classes 2 and 3.

| Level of ambiguity | Number of queries |
|---|---|
| 1 | 4 |
| 2 | 21 |
| 3 | 16 |
| 4 | 9 |

Table 4.3: Distrbution of ambiguity classes in the development data.

### 4.1.3 Top 10k Documents per Query

The third dataset provided by the organizers contains the top 10000 ranked search results for each query obtained from the ClueWeb09 and ClueWeb12 datasets. These datasets are part of the Lemur Project started by the University of Masachussets, Amherst and the Carnegie Mellon University, with an aim to develop search engines and text retrieval systems aimed at researchers. The ClueWeb09 dataset contains about 1 billion web pages in 10 languages collected between January and February 2009 [41]. The ClueWeb12 dataset was released as a companion to the ClueWeb09 dataset and contains about 733 million webpages collected between February and May 2012 [42]. These datasets are used by researchers for trials involving information retrieval systems and human language technologies. The dataset provided by the organizers of ClariQ contains a dictionary with the top ranked document IDs for each query.

## 4.2 Tools

### 4.2.1 Pytorch

Pytorch is an open-source library for machine learning tasks based on the Torch project, which aimed at developing libraries for scientific computation. Pytorch has both Python and C++ interfaces, but the Python interface has been used in this work. Python provides GPU accelerated tensor processing combined with a type-based automatic differentiation system for neural networks.

The unit of operation in Pytorch is a pytorch tensor, defined as an object of the class torch.Tensor. The Tensor class can handle homogeneous multidimensional arrays of numbers. The models in this work are coded and trained in Pytorch.

### 4.2.2 TextRank

TextRank is a keyword extraction and document summarization algorithm based on PageRank.

PageRank [43], developed by Sergey Brin and Larry Page in 1998, is an algorithm used to rank web pages in search results. It calculates a weight for each web page in the list of results by arranging all pages in a directed graph structure. The nodes have the web pages and one node has a link to another node if the first page has a link to the second page. The algorithm calculates a probability distribution that represents the likeliness of a person randomly clicking on links to arrive at a particular page. The algorithm can be applied to collections of documents on any size. According to the authors' description [43], pages linking to a certain page $A$ can be referred to as citations and represented as $T_1, T_2, \ldots, T_n$. A parameter $d$ known as the *damping factor* is set to a value between 0 and 1 and the measure $C(A)$ is defined as the number of links going out of $A$. Then the PageRank of $A$ will be calculated as

$$PR(A) = (1-d) + d\left(\frac{PR(T_1)}{C(T_1)} + \frac{PR(T_2)}{C(T_2)} + \cdots + \frac{PR(T_n)}{C(T_n)}\right) \tag{4.1}$$

$PR(A)$ corresponds to the principal eigenvector of the normalized link matrix of the web [43]. Thus, intuitively, the $PR(A)$ stands for the probability of a random surfer landing on page A and the damping factor $d$ represents the probability that they will get bored at some point and start surfing afresh, randomly. A page will have a higher rank if many pages link to it, as that might indicate that it is a high quality page.

The TextRank algorithm [37] uses this concept and applies it to portions of text from a document. The graph used by TextRank contains nodes that represent concepts or keywords derived from the document and the edges are usually weighted. So for two vertices $V_i$ and $V_j$ , a weight $w_{ij}$ is assigned to an edge between them. The authors thus modify the PageRank equation to factor in this weight [37]:

$$WS(V_i) = (1-d) + d \sum_{V_j \in In(V_i)} \frac{w_{ji}}{\sum_{V_k \in Out(V_j)} w_{jk}} WS(V_j) \tag{4.2}$$

where $WS(V_i)$ is the weighted score of vertex $V_i$, $In(V_i)$ is the set of edges pointing to $V_i$, that is the set of incoming links to $V_i$ and $Out(V_i)$ is the set of edges going out from $V_i$. The vertices may contain text units of any size or characteristics such as words, collocations, sentences, among others [37]. The score $WS$ is used to rank a vertex in the results.

One application of TextRank utilized in this work is for keyword extraction from documents. The vertices in this case consist of lexical units from the text,

such as words and the edges are made up of any connection that can be derived between those units. As an example, a co-occurrence relation can be determined based on the distance betwen those units as they occur in the text. Two units can be connected if they occur within a window of $N$ words in the text. The graph vertices can be filtered based on application preferences, such as, to only nouns and verbs. The algorithm works on this graph first by tokenizing the text and annotating them with parts of speech tagger. In the approach described by the authors, only single words are taken as the lexical units instead of adding all possible combinations of n-grams. The graph is constructed using the co-occurrence approach. When the final score for each vertex is obtained, the top $T$ vectices are sorted in descending order of their score. This sequence is then post-processed and all the words in the listing that occur adjacently in the text are collapsed into multi-word keyphrases. This algorithm, like PageRank, is completely unsupervised and was able to achieve state-of-the-art performance in precision and F-score across all evaluation systems.

TextRank can also be used for document summarization where the algorithm selects a subset of sentences from an input text. In this case, the vertices consist of complete sentences from the document. The edges are constructed and weighted based on the amount of content overlap, measured either by the number of overlapping tokens between the sentences or by the number of common words of certain parts of speech. The idea behind this approach draws from the concept behind PageRank: a sentence addressing a certain concept using certain tokens refers the user to another sentence addressing the same concept using the same tokens [36]. Given two sentences $S_i$ and $S_j$, and each $i$th sentence having $N_i$ words $w_1^i, w_2^i, \ldots, w_{N_i}^i$, the similarity function can be defined as

$$\text{Similarity}(S_i, S_j) = \frac{|\{w_k | w_k \in S_i \;\&\; w_k \in S_j\}|}{\log(|S_i|) + \log(|S_j|)} \tag{4.3}$$

Thus the text is represented as a weighted graph between sentences. The sentence scores are calculated using the weighted TextRank equation. The top ranked sentences are finally selected for inclusion in the summary.

### 4.2.3 RAKE

Another algorithm used in this work to detect keywords is RAKE or the Rapid Automatic Keyword Extraction [44]. The RAKE algorithm is an unsupervised and domain and language independent algorithm for extracting keywords which achieves scores comparable to TextRank but is computationally more efficient [44].

The principal concept behind this algorithm lay in the fact that stopwords are often removed from extracted natural language text with the assumption that they do not contribute to the information content of the text. RAKE takes in a list of stop words, a set of phrase delimiters and a set of word delimiters [44]. These delimiters are then used to partition the document into candidate keywords which consist only of information-containing words and not the stop words [44]. Co-occurrences of words in the candidates are used to measure word correlations which are in turn used to score the keyphrases. A word co-occurrence graph stores this measure and contains the number of times each pair of words occurs in the same candidate. A candidate score is generated for each keyword based on multiple metrics such as word frequency, word degree and ratio of degree to frequency [44]. Lastly, keywords that occur multiple times in the same order are accumulated with the removed stop words added in between. The candidate score derived from the word co-occurrence graph is also used to rank the keywords in the output.

### 4.2.4 spaCy

spaCy is an open source python library for advanced natural language processing developed by Matthew Honnibal and Ines Montani. spaCy is similar to the NLP package NLTK but differs in its target users in that its APIs are mainly geared towards production environments and not only research and academics. To achieve this, spaCy supports deep learning architecture that connect models trained in other frameworks such as Python, Tensorflow etc. spaCy also has built in pipelines for popular and frquently used NLP tasks such as named entity recognition, POS tagging, text classification among others. It also supports popular keyword extraction and summarization algorithms and allows the developer to create pipelines using algorithmic implementations of their choice. Finally, being geared towards production systems, it achieves state-of-the-art accuracy and speed and allows multiple extensions that enable the developer to use external backends and visualizing libraries [45].

In this work, spaCy is used for extracting keywords from text using the TextRank algorithm which is already implemented and supported by spaCy for addition to a text processing pipeline. This pipeline can be further extended for named entity recognition in future versions of this work.

### 4.2.5 ChatNoir

ChatNoir or Elastic ChatNoir is a search engine allowing search over the ClueWeb09 and ClueWeb12 and the Common Crawl Corpora mainly aimed at

free access to researchers. The search engine offers subsecond response times comparable to commercial search engines [46]. The search engine is built upon the open-source Elasticsearch search backend.

To create the data store, the authors have parsed the plain WARC (Web Archive) files for each corpus and determined the content type and encoding each entry uses. Each file is assigned a unique UUID based on the file name. The files are stored in HDFS, with the map files directing the UUIDs to JSON files containing the headers and content, and the URLs to UUIDs [46]. During the mapping and indexing phase, all documents from which no meaningful content can be extracted, are discarded and the Elasticsearch index is prepared based on a multifield JSON document that contains all information from each web page. The web frontend uses plain text-based search at first to retrieve documents and then reranks them using a more complex query. The BM25 retrieval model is used for obtaining the documents. The system provides a powerful API to researchers who can use the data without gaining access to each corpus separately.

In this work the ChatNoir API has been used to retrieve full documents based on UUIDs derived from the document IDs provided in the dataset.

# Chapter 5

# Experiments and Results

This chapter describes the approaches that have been explored in order to achieve ambiguity detection and explains and analyses the results achieved.

## 5.1 Preliminary Approaches

The data available for the task at hand are only the initial queries input by the user. This makes this task similar to a sentence classification task. For any model that takes text as input, the text needs to be cleaned and processed in order to make it suitable for input to the model.

The text preparation process comprised of removing the stop words from the queries and tokenizing the queries. Stop words are commonly used words that do not usually hold any meaning or affect the results of a search, such as "a", "the" and others. Removal of such stop words reduces the size of input to be processed. The algorithms cannot understand natural language at the input layer and hence the text needs to be represented numerically as vectors of real numbers, also known as embedding, as explained before. Embedding converts each unit of text into a vector representation. For the embedding process, the text needs to be broken down into units, known as tokens. In this work, a token refers to a word. As an example, the input query

```
Tell me about folk remedies for a sore throat.
```

Tokenization will convert this sentence into a list of tokens or words, in this case:

```
['tell', 'me', 'about', 'folk', 'remedies', 'for', 'a', 'sore',
'throat']
```

After removal of stop words from among the tokens, the text looks like

```
['tell', 'folk', 'remedies', 'sore', 'throat']
```

In natural languages, many words have separate forms depending on the context. However from a search query point of view, they may convey the same meaning. For this, a lemmatizer is applied on the tokens. A lemmatizer converts the inflected forms of a word to its base form or dictionary form, also known as *lemma*. For example, the lemma of "walking" will be "walk" and that of "better" will be "good". In the example query, the word "remedies" will be converted to its lemma "remedy". The modified text will be

```
['tell', 'folk', 'remedy', 'sore', 'throat']
```

Finally, this form can be encoded with a vectorizer. Stop word removal and lemmatization were done only for the approaches that did not involve a transformer based vectorizer. Transformer language representation models such as BERT benefit from the correct sequence of stop words and content words and can capture semantic representations better. Hence tokenization for transformer based models were done by their specific pre-trained tokenizers that added special tokens required for the transformer model to work.

In the first initial approach, the initial queries were embedded using word2-vec [35], pre-trained on Google News text, with vector dimension of 300. Each token in the list is replaced by a corresponding vector. If any word does not exist in the list of embeddings, it is assigned a randomly generated vector. After embedding, the query will look like

```
[array([[-0.01879883, -0.11816406, -0.14355469, ..., -0.05566406,
0.12255859, -0.10253906], [-0.12353516, 0.07226562, 0.171875
, ..., -0.02246094, 0.06689453, 0.02685547], [-0.04125977,
-0.20800781, 0.06445312, ..., -0.18457031, 0.17871094,
-0.00747681], [ 0.484375 , 0.12255859, -0.15722656, ...,
-0.08886719, -0.04296875, 0.01916504]], dtype=float32),
array([[-0.01879883, -0.11816406, -0.14355469, ..., -0.05566406,
0.12255859, -0.10253906], [-0.12353516, 0.07226562, 0.171875
, ..., -0.02246094, 0.06689453, 0.02685547], [ 0.484375 ,
0.12255859, -0.15722656, ..., -0.08886719, -0.04296875,
0.01916504], [-0.04125977, -0.20800781, 0.06445312, ...,
-0.18457031, 0.17871094, -0.00747681]], dtype=float32),...
```

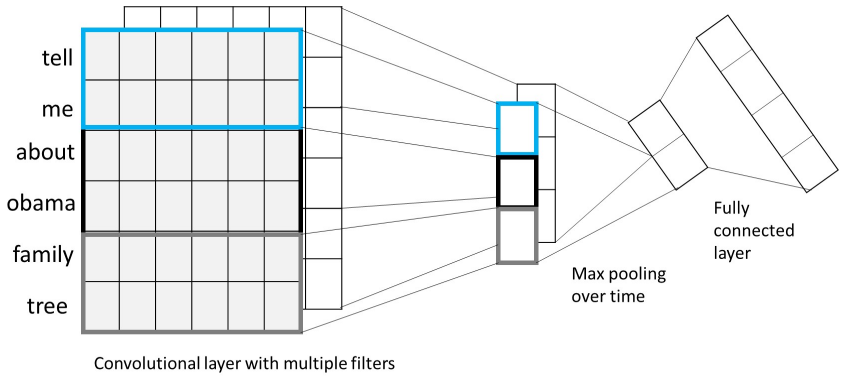



Figure 5.1: The CNN based query classification model. The input is represented as a matrix of word vectors, on which multiple convolution kernels operate. The feature maps are then subjected to max pooling over time which produces a single vector from the feature maps. This vector is then processed by the fully connected layer to output the class probabilities.

where each subarray represents a word turned into a 300-dimension vector that captures its semantic relationship with other words.

The word2vec encoded vectors were passed to a CNN similar to that proposed by [15] and explained in section 3.4 and the output of the convolutional layers were passed on to dense layers for the final classification. To account for queries of varying lengths, the inputs matrices were padded with 0-rows to make the number of rows same as the maximum number of words for a query in the dataset. The convolutional layer can be made of multiple filters having same or different receptive fields. An overview of the model is shown in Figure 5.1.

Since the number of queries in the dataset is small (only 187 unique queries), the number of datapoints was increased using data augmentation for which 100 random permutations of the lemmatized tokens were added to the dataset and were labelled similarly to the original query. These permutations were then encoded using word2vec and the same CNN was used to classify the queries.

Developing on this approach, the second initial approach consisted of embedding the input queries, with only their punctuation removed, using RoBERTa instead of word2vec. BERT based models are known to produce vectors that better represent context in the text and hence they are expected to perform better when encoding queries. In this case, the RoBERTa-base model was used and the words in the queries were converted to vectors of 1024 dimensions.

Another approach involved using the flattened RoBERTa embeddings directly as inputs to the dense layers for classification without using convolutional layers. However, this approach did not elicit any observable improvement in the results.

Finally, combining the approaches, augmented inputs encoded using RoBERTa were flattened and passed to the dense layers, again without any observable gains in performance.

The initial approaches exhibited the need for more information regarding the user's intent. Since the scope of this work involves open-domain conversations, relying on user behaviour from previous similar searches was not possible or feasible since a new query may turn out to be entirely unprecedented. In such a system, when a user provides a search query, the only other source of information available to the system is the set of documents retrieved by the search operation. The primary concept behind using this set of documents originated from [5] where the authors classified each document into pre-defined categories, which may not be feasible in case of a open-domain query. However, the similarity between the documents may provide clues about the range of topics covered by the documents. If the top-ranked documents in the search results diverge from each other, then it may point to an ambiguous query from the user. But if they are mostly similar to one another, then the query may be deemed clear and unambiguous.

To follow this approach, access to the search results for each query, were required. The dataset provides a dictionary with the IDs of the top 10000 documents retrieved for each search query from the ClueWeb09 and ClueWeb12 datasets. Access to the full HTML text for these documents was obtained through the API provided by the ChatNoir search service [46].

The documents retrieved have all HTML tags and formatting information and hence need to be preprocessed. As an example, a full retrieved HTML document is of the form:

```
<!doctype html>\n<meta charset="utf-8">\n<title>A Letter written
on Oct 30, 1937</title>\n<body>\n\n<h1>A Letter written on
Oct 30, 1937</h1> \n<p> </p> \n<blockquote> Department of
Chemistry, Yenching University, Peiping, - October \n30, 1937.
<br>\n\n<p> <b>Extracts from the Adolph Diary:  1937.</b> </p>\n
<p> <em>January:</em> Ice skating on the Yenching lake at its
prime.  EHA, in \nthe 8th grade, North China American School,
makes the hockey team.  HMA, \nfreshman at Mount Holyoke College,
amkes her second all-Holyoke athletic team \n(basketball).  WHA
Jr., sophomore at Yale University (Calhoun College), works in
\nbiology and plays football and basketball on his college team.
<\blockquote><\body>
```

The HTML tags do not contribute to the meaning of the text and hence need to be ignored for successful embedding of the document. The text is processed using the BeautifulSoup package to remove all HTML tags and newline characters. After processing, the text is turned to

```
A Letter written on Oct 30, 1937 A Letter written on Oct 30, 1937
Department of Chemistry, Yenching University, Peiping, - October
30, 1937.  Extracts from the Adolph Diary:  1937.  January:  Ice
skating on the Yenching lake at its prime.  EHA, in the 8th grade,
North China American School, makes the hockey team.  HMA, freshman
at Mount Holyoke College, amkes her second all-Holyoke athletic
team (basketball).  WHA Jr., sophomore at Yale University (Calhoun
College), works in biology and plays football and basketball on
his college team.
```

This cleaned text can now be encoded using a document encoding algorithm such as Doc2vec or SBERT.

Multiple attempts were made using various models before the final models were decided upon. Upon exploring multiple models to process the interrelationship between documents, graph convolutional networks were finally decided upon, since the relationship and similarities between multiple documents can be expressed in the form of a graph and graph convolutional networks have the capability to generate latent representations of the graphs that can then be used to classify them. In order to use graph convolutional networks, the relationship between the documents needed to be expressed in terms of a node feature matrix and an adjacency matrix, as explained in Section 3.5. Since we require the similarity between the topics covered in the documents, cosine similarity was chosen

as an appropriate measure to determine the graph edge weights. To prepare the node feature matrix, the initial attempts included the pre-trained document embedding techniques Doc2vec and SBERT. The output from the graph convolutional network can be passed onto dense layers for classification.

For this, a graph convolutional layer unit was prepared according to [4]. The weights and biases were initialized from the uniform distribution. The initial experiments included simple networks with GCN layers followed by dense layers. The node feature matrix consisted of documents embeddings obtained with SBERT or doc2vec. To prepare the adjacency matrix, cosine similarities were calculated between all pairs of documents retrieved for a particular query. The document graph was an undirected graph with the nodes having the document embeddings and the edges having weights assigned by the similarity between two document nodes.

To improve on the performance of the model and add the context of the initial query to the classification process, the initial query was encoded using SBERT and concatenated to the flattened output of the GCN, with a slight improvement in performance.

Encoding the entire document requires a large amount of memory and due to the limited amount of resources available, each document was replaced by a concatenation of the top 100 keywords obtained by the TextRank algorithm.

## 5.2 Approach 1: Document Similarity Graph

This approach draws from the initial approaches to take into account the documents retrieved in the search results. The SBERT model that is used to encode the documents is added to the complete architecture of the classifier and the weights of the transformer are fine-tuned during the training of the entire model. An overview of the architecture can be seen in Figure 5.2.

The input HTML documents are first processed to remove the HTML tags and other unnecessary characters. The top 100 keywords for the documents are extracted using the TextRank algorithm and concatenated with spaces as delimiter to represent the entire document. The initial query is converted to lowercase and all punctuation characters are removed.

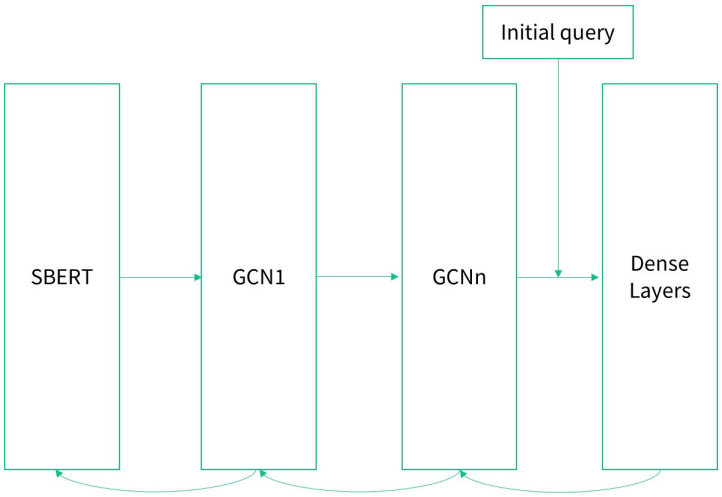



Figure 5.2: Overview of the first approach presented in this work. The documents are input to the first SBERT layer and after processing, are passed onto the GCN layers. The output of the last GCN layer is flattened and concatenated with a static SBERT embedding of the initial query text. Finally the dense layers perform the classification. The arrows at the bottom of the image pointing backwards from the final layer to the SBERT layer represent the backpropagation operation.

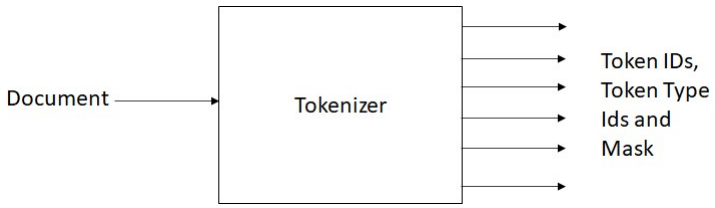



Figure 5.3: The Tokenizer takes the document as input and its output consists of the token IDs, token type IDs and the attention mask.

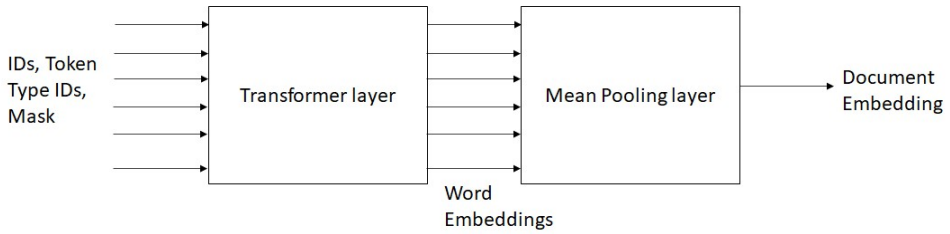



Figure 5.4: The document embedding model. The first layer consists of a transformer that vectorizes the input document, followed by a mean pooling layer to obtain a single vector for the entire document.

The input documents are processed using the pre-trained tokenizer corresponding to the transformer architecture used. The tokenizer takes in the document, tokenizes the text and adds special tokens as required by the language representation model. The tokenizer outputs three lists:

- *IDs*: The IDs help locate the tokens from the vocabulary the tokenizer is trained on. This list contains numerical representations of the tokens in the text, including special tokens as required by the downstream model, from the tokenizer vocabulary.
- *Token Type IDs*: This list is necessary for models which take in a pair of sequences as input, such as question-answer models. The token type IDs identify tokens belonging to each sequence.
- *Attention Mask*: The attention mask notifies the downstream model about the tokens it needs to attend to, such as valid text tokens, and about tokens it needs to ignore, such as padding tokens.

The tokenizer operation is shown in Figure 5.3.

The SBERT layer consists of a transformer layer followed by a mean pooling layer, as proposed in [33]. The transformer model is used to embed the words in the text and the mean pooling layer calculates the mean over all such embedding vectors and converts the document into a vector of fixed size, as shown in Figure 5.4. The language representation model takes the tokenizer outputs and converts them into a vector for each word. For example, the RoBERTa base model generates a 768-length token for each word in the text. The mean-pooling layer then takes the mean for each feature in that vector over all the words and outputs a single vector for the whole document of length 768. The weights of the transformer layer are finetuned as part of the training of the whole classification model. Additionally, the initial query is embedded using a static pre-trained

SBERT model for concatenation to the graph convolutional layer output. The base transformer model for embedding the words in the initial query maybe the same as the base model for embedding the words in the documents, or a completely different model.

For a set of documents

$$\mathcal{D} = \{D_1, D_2, \ldots, D_d\} \tag{5.1}$$

the embedding process outputs a set of vectors

$$\mathcal{E} = \{\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_d\} \tag{5.2}$$

These vectors are then processed into the Document Similarity Graph for input to the graph convolutional layers.

The Document Similarity Graph is a weighted undirected graph with a set of vertices $V$ and a set of edges $\{(v_i, v_j)|v_i, v_j \in V, i \neq j\}$. Each vertex $v \in V$ represents a document and is defined by the embedding vector of the document. Thus the number of vertices in the graph is the same as the number of documents taken into consideration for each query. Thus the node feature vector is a $d \times k$ matrix where $d$ is the number of documents and $k$ is the length of each document vector. The weight assigned to each edge is determined by the cosine similarity between the documents belonging to the end vertices. So for two vertices $v_i$ and $v_j$ having documents $d_i$ and $d_j$, the cosine similarity score is calculated by

$$\text{cos\_sim}(v_i, v_j) = \frac{v_i \cdot v_j}{|v_i||v_j|} \tag{5.3}$$

which is the dot product of the normalized document vectors. Thus the adjacency matrix is a $d \times d$ matrix with similarity scores between all possible document pairs. Self loops are added to the graph because the similarity of a document is always 1. The degree matrix is derived from this adjacency matrix as explained in Section 3.5. The adjacency matrix is normalized and then input into the graph convolutional layers for processing. The output of the last graph convolutional layer is flattened and concatenated with the vector representation of the initial query to add the context of the initial query to the processing, as shown in Figure 5.5. This merged vector is then passed to the dense layers to obtain the classification probabilities for the four ambiguity classes.

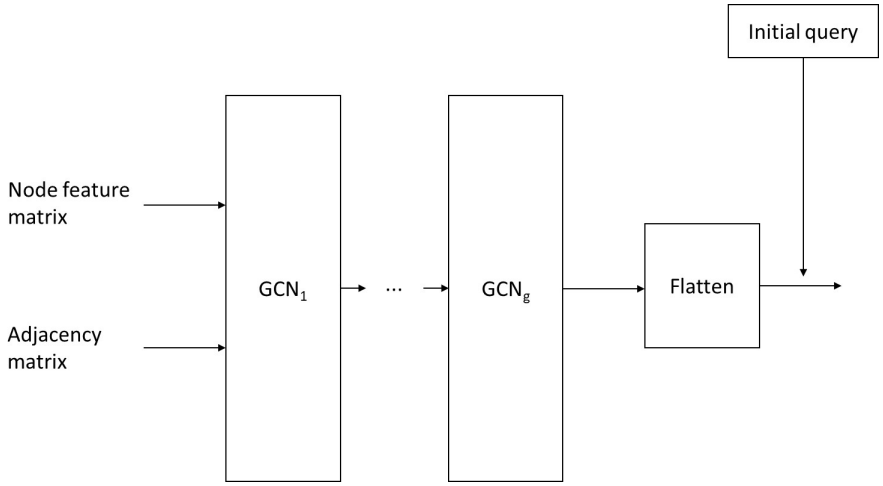



Figure 5.5: The graph convolutional layers and the concatenation of the output of the last layer to the initial query.

## 5.3 Approach 2: Concept Graph

The second approach draws from the work of Bang et. al. [6], explained in Section 3.8. However, since the model proposed in the paper is concerned with only a pair of documents at a time and determines document similarity using similarities between sets of sentences in those documents, this approach was not directly applicable to this work which can take into account a much larger number of documents and hence the sentence based processing can become expensive in terms of spatial and temporal complexity.

The steps for preparing the graph for the second approach is shown in Figure 5.6. In this approach, each document is preprocessed by extracting a set of top keyphrases used in the document. The number of keywords used is a hyper-parameter that can be varied. For each query, an IDF dictionary is generated from all the documents taken into account. The IDF dictionary contains the IDF values for all the words in those documents.

This approach relies on a Concept Graph instead of the Document Similarity Graph as in the previous model. Concepts are defined as clusters of closely related keyphrases. To cluster the keyphrases, they are encoded using a sequence embedding algorithm (such as sentence transformers) that can generate a vector of fixed length for keyphrases of varying lengths and also capture the semantic relationship between different sequences. These sequences are then collected into a pre-defined number of groups to obtain the concepts over all the retrieved

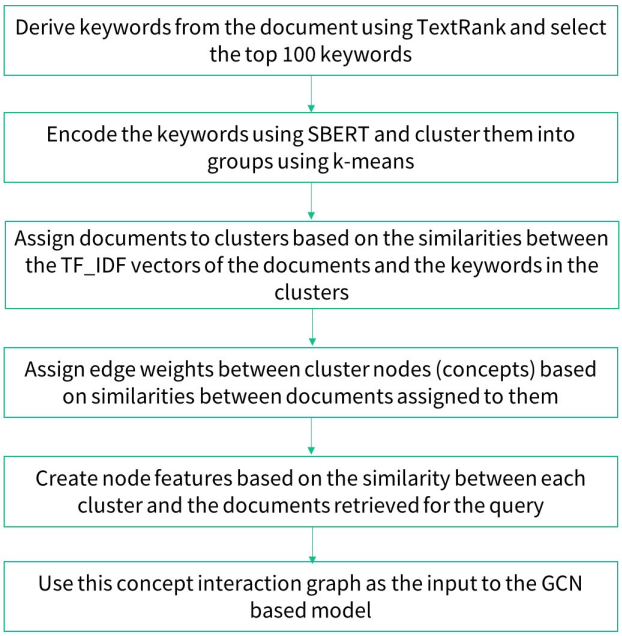



Figure 5.6: The steps for preparing the Concept Graph from documents.

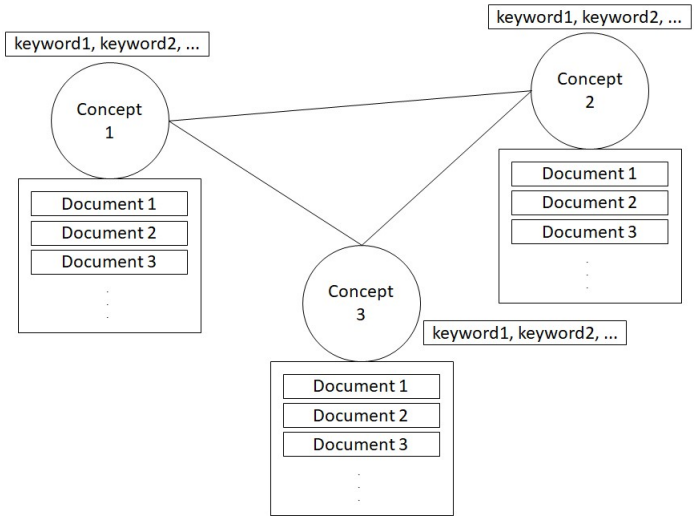



Figure 5.7: The structure of the Concept Graph. Each concept is defined as a cluster of keywords and has a set of documents associated with it. The edges are weighted by the similarity between those document clusters.

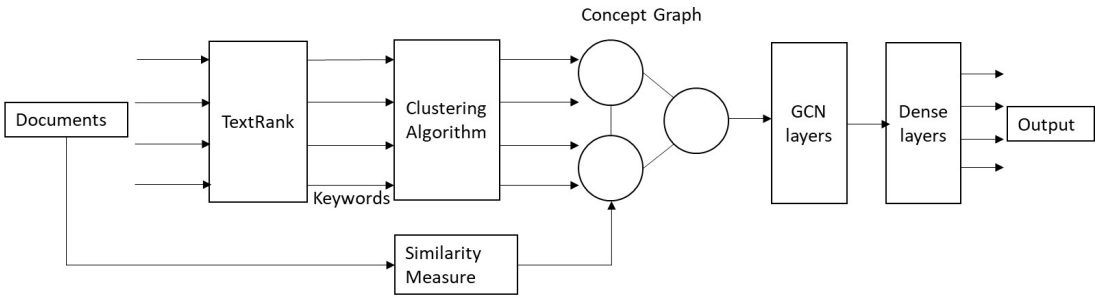



Figure 5.8: The query ambiguity classification model using the Concept Graph.

documents.

$$C = \{c_1, c_2, \ldots, c_m\} \tag{5.4}$$

where $m$ is the pre-defined number of clusters. The clusters or concepts then form the vertices of the concept graph.

Each concept $c_i$ is now defined in terms of the concatenation of keywords belonging to that concept and quantified as the TF-IDF vector of the concatenated sequence.

$$c_i = \text{TF-IDF}([k_1, k_2, \ldots, k_{p_i}]) \tag{5.5}$$

where $p_i$ is the number of keywords in concept $c_i$. Documents are assigned to each concept based on the cosine similarity between the TF-IDF vectors of the concept and the document. Thus each vertex of the concept graph has a set of documents associated with it. The weights of the edges between the concepts are determined by the cosine similarity scores between the document clusters associated with the end vertices.

$$W_{ij} = \cos\_\text{sim}(DS_{c_i}, DS_{c_j}) \tag{5.6}$$

where $W_{ij}$ is the edge weight between concepts $c_i$ and $c_j$. To calculate this cosine similarity measure, each document cluster is represented by a concatenation of the documents in that cluster and the TF-IDF vector for this sequence is generated.

$$DS_{c_i} = \text{TF-IDF}([D_1, D_2, \ldots, D_{d_i}]) \tag{5.7}$$

where $DS_{c_i}$ represents the set of documents associated with concept $c_i$ and $d_i$ is the number of documents assigned in that set. The TF-IDF vectors are then compared using the cosine similarity equation 52. The adjacency matrix of the Concept Graph is thus an $m \times m$ matrix having the similarity measure between all possible pairs of document clusters in the graph. The node feature vector is calculated by the cosine similarity between the TF-IDF vector of each concept

and that of the documents retrieved by the search operation. Hence the node feature matrix is an $m \times d$ matrix where $d$ is the number of documents retrieved for each query. The concept graph structure is shown in Figure 5.7.

This graph is then passed on to the graph convolutional layers for further processing, as in the previous approach. However, contrary to the first approach, the output of the last graph convolutional layer is not concatenated to the initial query representation. It is flattened and passed to the dense layers for final classification. The complete model is shown in Figure 5.8.

## 5.4 Results

### 5.4.1 Preliminary Approaches

| **Convolutional layer(s)** | **Precision** | **Recall** | **F1 Score** |
|---|---|---|---|
| **64 filters** with a receptive field of **2 words** each | 0.3486 | 0.44 | 0.3818 |
| **128 filters** with a receptive field of **2 words** each | 0.3161 | 0.4 | 0.344 |
| **32 filters** with a receptive field of **2 words** each | 0.3909 | 0.46 | 0.3841 |
| **64 filters** with a receptive field of **3 words** each | **0.3594** | **0.44** | **0.3946** |
| **128 filters** with a receptive field of **3 words** each | 0.2940 | 0.38 | 0.3201 |

Table 5.1: Query classification using word2vec and convolutional layers.

The first preliminary approach involved the CNN based model, following Kim [15]. The models performed to a limited extent given the scarcity of adequate information to decide on the ambiguity of the input query. This model is appropriate for classifying sentences when the classes can be decided only based on the information contained within the sentence. But ambiguity cannot be decided from the words in the input query. The queries were processed, as explained in the previous section, with the removal of stopwords and punctuation and the words were embedded using word2vec, pre-trained on the Google News dataset, into vectors of length 300. Since the training dataset provided by the challenge organizers has only 187 unique queries, data augmentation was performed with 100 permutations of each query, where possible. The query vector matrices were

padded with 0-rows to account for queries with varying lengths. The results from this approach, on the dev set, have been summarized in Table 5.1.

As evident from the excerpt of results in Table 5.1, the performance of the model depends on the number of kernels and the receptive field. On the dev set, the highest F1 score achieved during the experiments was 0.39. Expressing the results as a confusion matrix in Figure 5.9, where the contents of a cell $ij$ depicts the number of times the algorithm predicted a class $j$ for a sample with true class $i$, it is evident that this approach has worked best with queries of ambiguity class 2, as it has successfully classified 13 queries with ambiguity level 2. It has also

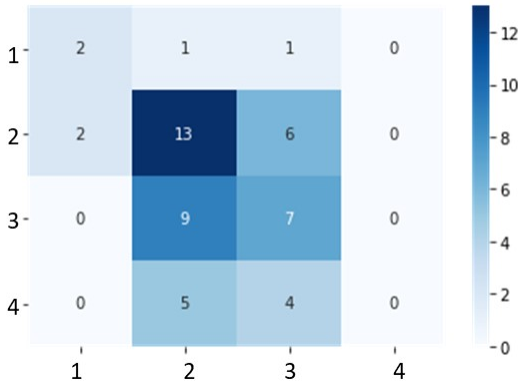



Figure 5.9: Confusion matrix for the approach with CNN and word2vec embeddings. The rows represent the ground truth labels and the columns contain the predicted values.

correctly identified 2 clear queries but has failed to identify completely ambiguous queries and classified a larger number of them as class 2.

Attempts to improve this model involved embedding the queries using RoBERTa which performs better in capturing the semantic relationship between words in the queries. The best results for the experiments using RoBERTa embeddings of length 1024 are summarized in Table 5.2.

| **Model** | **Precision** | **Recall** | **F1 Score** |
|---|---|---|---|
| RoBERTa embeddings + augmentation + CNN | 0.4183 | 0.38 | 0.3717 |
| RoBERTa embeddings + augmentation + dense layers | 0.4861 | 0.4 | 0.3719 |

Table 5.2: Query classification using RoBERTa embeddings.

As is evident from the results, there was no observable improvement in results in the absence of further information about the queries.

After these initial attempts, it was decided to take into account additional information relevant to the ambiguity of the queries, in the form of documents retrieved in the search results. Keeping in mind the memory, computing power and storage space of the available resources, plain-text HTML versions of the top 100 ranked documents were downloaded through the ChatNoir API. The documents were cleaned of unnecessary markup as explained in the previous sections and was processed using a graph convolutional network. Multiple architectures with GCNs were tested with both Doc2Vec and SBERT embeddings of documents with resultant F1 scores between 0.26 and 0.40. Adding the ranking information of the documents in the form of one-hot encoded vectors and also as weights on the node features in the graph (document embeddings) did not provide any observable improvement in the results and in some cases led to degradation of performance.

### 5.4.2 Document Similarity Graph

In this approach, the SBERT weights were trained along with the classification model and hence the initial layers are responsible for a high spatial complexity. To take into account the limitation of GPU memory available for training, each document was represented as a concatenation of the top 100 keywords extracted from it by both the TextRank and Rake algorithms. The TextRank keywords are also used in the second approach (Concept Graph) to create the concept clusters. Hence the starting points of both algorithms have common ground. In this first approach, the SBERT module was made of a smaller Electra model with output embeddings of length 384 to enable more documents to be processed for each query. The base model used for SBERT encoding of the initial query was a MiniLM model pre-trained on the Microsoft Marco datasets. GCN module had 3 layers with 10 GCN cells each with dropout. The intial query dataset was augmented using random permutations of the word tokens, with the assumption that the retrieved documents would remain the same with the same ranks for all permutations of the same initial query.

The model was tested with 10 and 20 documents per query as memory requirements did not allow more than that. The results are summarized in Table 5.3 (TextRank Keywords) and Table 5.4 (Rake keywords).

| Documents per Query | Precision | Recall | F1 Score |
|---|---|---|---|
| 10 | 0.296 | 0.41 | 0.32 |
| 20 | 0.2942 | 0.392 | 0.34 |

Table 5.3: Document Graph Approach using TextRank Keywords

| Documents per Query | Precision | Recall | F1 Score |
|---|---|---|---|
| 10 | 0.3522 | 0.4 | 0.35 |
| 20 | 0.4213 | 0.42 | 0.42 |

Table 5.4: Document Graph Approach using Rake Keywords

From the tables, it is apparent that the metrics improve when Rake keywords are used and also when the number of documents per query are improved. The confusion matrix in Figure 5.10 shows an example of the document graph approach's performance.

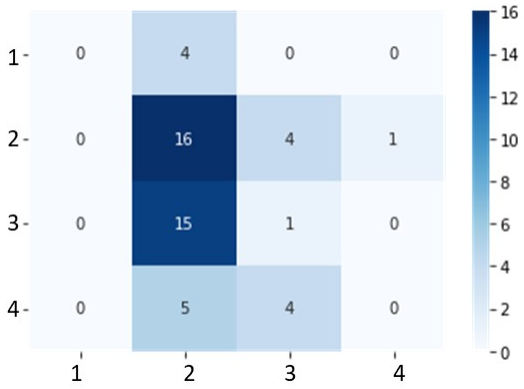



Figure 5.10: Confusion matrix for a sample run of the Document Graph approach with the documents represented as TextRank keywords.

The algorithm has successfully identified the majority of class 2 queries, but has failed to identify clear queries and in this set of predictions shows an inclination towards labelling queries as class 2 ambiguous. However, it has not completely misidentified any ambiguous query of level 4 as unambiguous.

### 5.4.3 Concept Graph

For the concept graph approach, the top 100 keywords of each document was extracted using TextRank and clustered into concepts using K-means clustering. The TF-IDF vectors of the documents and concepts were calculated and the Concept Graph was constructed as explained in Section 5.3. This graph was then input to the graph convolutional model for further processing and classification. There were 2 principal hyperparameters in this model: the number of documents per query and the number of clusters or concepts. The number of concepts directly affects the size of the graph as the concepts form the vertices. Thus the number of clusters effectively affects the memory complexity of the process. The results have been summarized in Table 5.5.

| **Documents per Query** | **Number of Concepts** | **Precision** | **Recall** | **F1 Score** |
|---|---|---|---|---|
| 20 | 10 | 0.2578 | 0.42 | 0.32 |
| 50 | 10 | 0.2469 | 0.34 | 0.29 |
| 50 | 20 | 0.422 | 0.4 | 0.33 |
| 100 | 20 | 0.5453 | 0.46 | 0.44 |

Table 5.5: Concept Graph approach.

Increasing the number of documents as well as the number of concepts improves the results. The confusion matrix for this model is shown in Figure 5.11.

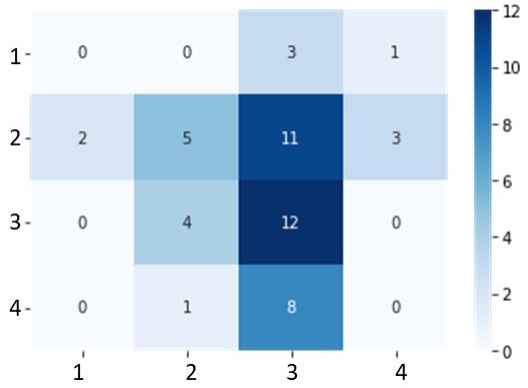
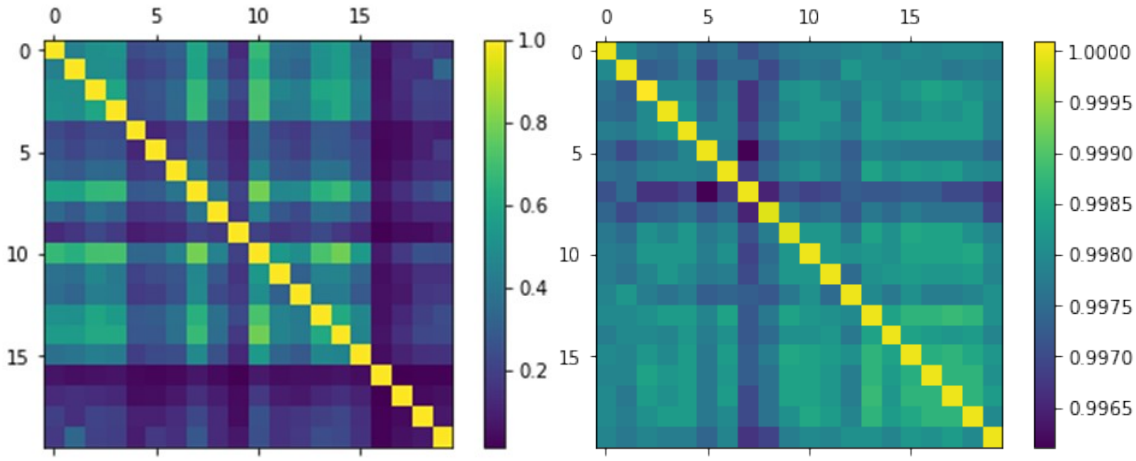



Figure 5.11: Confusion matrix for the Concept Graph approach

This model also did not misidentify any completely ambiguous query of level 4 as unambiguous. But it identified most of the level 2 queries as level 3. A majority of the level 3 queries were correctly identified and most of the level 4 queries were identified as level 3.

## 5.5 Analysis

Analysing the two models further, it is apparent that while the Document Graph approach has a proclivity towards level 2, the Concept Graph approach is more inclined towards level 3. This behaviour may be attributed to the fact that with the constrained training performed, the Document Graph approach has a limited visibility upto 20 of the retrieved documents, which in some cases may not be fully indicative of the expanse of concepts covered by the documents. The Concept Graph approach has a better coverage of 100 documents and hence logically can have better clustering of documents based on concept similarity. An example of this can be seen in topic ID 106, initial query: "I'm looking for universal animal cuts reviews". This query has a ground truth ambiguity level 3 and according to the provided training data, the ambiguity arises from whether the user wants to buy Universal Animal Cuts, want to know about their nutrition values or want to know about their safety for consumption. Thus this topic has sufficient scope of ambiguity but is not completely open-ended. This topic is correctly identified as level 3 by the Concept Graph approach, but is misidentified as level 2 by the Document Graph approach. Analysing the adjacency matrices generated by the two approaches (Figure 5.12):



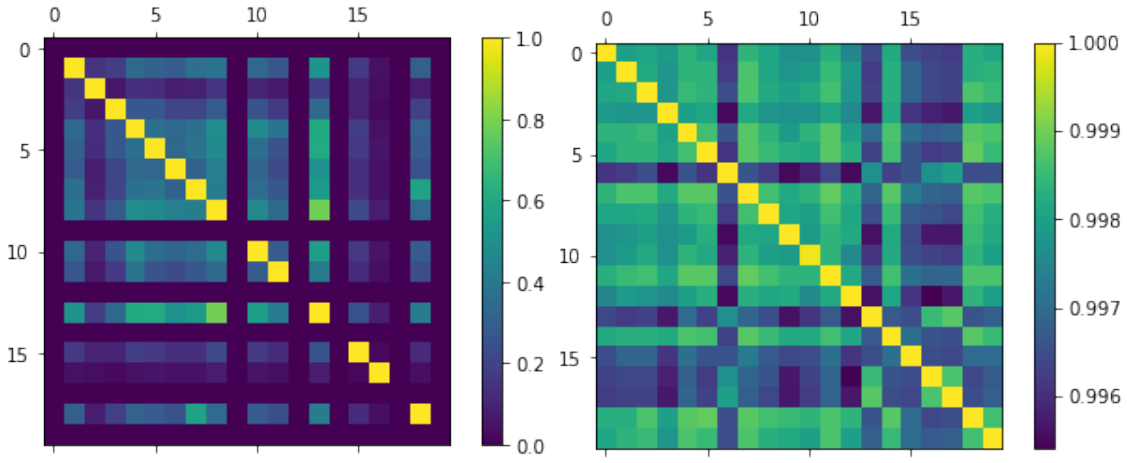

(a) Concept Graph (b) Document Graph

Figure 5.12: Adjacency matrices for topic 106 for the Concept Graph approach (a) and the Document Graph approach (b)

In the diagrams, the darker the colour, the lower is the cosine similarity value. The adjacency matrix on the left represents the similarity between the sets of documents associated with the concepts, whereas the matrix on the right shows the cosine similarity between the document representations. The Concept

Graph approach, in this case, takes into account 100 documents with 20 concepts and hence the adjacency matrix captures the relationship between the concepts better. The adjacency matrix for the Concept Graph approach shows clear disparity between multiple clusters particularly between the lower numbered ones and higher numbered ones. This indicates distinct concepts and disparate sets of documents associated with them, which is an indicator of the level of ambiguity. Additionally, the larger number of documents in consideration makes the Concept Graph approach better poised to detect the range of concepts covered in the documents. On the other hand, the adjacency matrix of the Document Graph approach considers only 20 documents for the query and as evident from the diagram, does not detect much disparity between them. The top 20 documents in this case are mostly similar and contain sites selling Universal Animal Cuts. Hence the matrix has almost uniform values. Evidently, the additional concepts are covered in lower ranked pages, which are accessed by the Concept Graph approach. Thus it can be deduced that having clustered concepts from a larger number of documents helped the Concept Graph approach detect the level of ambiguity correctly whereas a smaller number of less disparate documents embedded using a smaller model led the Document Graph approach to assign a lower level of ambiguity to the query.

One of the shortcomings of the Concept Graph approach can be seen in the adjacency matrix for topic ID 101, initial query: "Find me information about the Ritz Carlton Lake Las Vegas.": This query is ambiguous as the user's intent is not clear. It is a level 2 ambiguous query and can mean that the user wants to know about the location of the Ritz Carlton Lake resort in Las Vegas or they may want information about its capacity or have some interest in its history. This query is correctly classified by the Document Graph approach as a level 2 query but misidentified by the Concept Graph approach as level 3. The adjacency matrices are shown in Figure 5.13




(a) Concept Graph (b) Document Graph

Figure 5.13: Adjacency matrices for topic 101 for the Concept Graph approach (a) and the Document Graph approach (b)

Analysing the adjacency matrix for the concept graph approach, it can be seen that some of the rows and columns have 0 values. This can be attributed to the fixed number of concepts that have to be prepared for each initial query. Some of the concepts may not have any document associated with them and the number of clusters of keywords may be more than required, leading similar keywords to be grouped into separate concepts and similar documents being assigned to different concepts. Also the concepts that do not have any documents assigned to them fail to influence the output. This can lead to incorrect labelling of the ambiguity, as seen in this case. On the other hand, the Document Graph approach, although recording high similarity between documents, can capture the separate concepts covered by the documents since the search results for this query present the entire variety of end goals within the first 20 documents.

Thus, taking into account the restrictions on the number of documents within the scope of the document graph approach, it's tendency to go with the lower level of 2 can be explained. It works better on queries whose search presents a better variety of subjects within the first few results. The concept graph approach works better for queries which require a greater number of documents to fully understand the expanse of its intents, but in the present implementation, suffer from the effects of having a fixed number of concept clusters, which makes it more inclined towards the higher ambiguity level of 3. The number of samples with ambiguity level 1 and 4 are much less in both the training and development datasets, which make it difficult to assess the performance of the models on those

samples. However, it is reassuring to see that the tendency of the models to designate ambiguous queries as unambiguous or unambiguous queries as having an extreme ambiguity level of 4 is very low.

Finally, the achieved results were compared with the top ranked entries in the ClariQ challenge. In the absence of access to the test dataset labels, the comparison only included scores on the dev dataset and the entries have been sorted according to their F1 scores. The results are summarized in Table 5.6. Since these models have not been published yet, it was not possible to perform detailed comparison of the proposed approaches with them.

| **Model** | **Precision** | **Recall** | **F1 Score** |
|---|---|---|---|
| BartBoost | 0.7008 | 0.7 | 0.6976 |
| cneed_add_prior_v2 | 0.62 | 0.6 | 0.5984 |
| Roberta+ Stats | 0.62 | 0.58 | 0.5717 |
| Roberta+++ | 0.6039 | 0.56 | 0.5551 |
| Roberta++ | 0.5807 | 0.54 | 0.5375 |
| cneed_merge | 0.5830 | 0.52 | 0.5192 |
| cneed_dist | 0.5452 | 0.52 | 0.5177 |
| **Concept Graph model** | 0.5453 | 0.46 | 0.44 |
| BERT-based-v2 | 0.5218 | 0.4800 | 0.4253 |
| **Document Graph model** | 0.4213 | 0.42 | 0.42 |
| Triplet | 0.4161 | 0.48 | 0.4178 |
| Roberta+ CatBoost | 0.1402 | 0.28 | 0.1854 |

Table 5.6: Comparison with ClariQ challenge entries.

# Chapter 6

# Conclusion and Future Work

Natural language based systems are gradually replacing traditional modes of interaction with the internet. Searches are gradually going hands free and users expect results that are to the point and require as little effort on their part as possible. In order to achieve that, generating and asking clariyfing questions has been proven to be a way to move forward to narrow down the field of search. The aim of this work was to explore ways to detect when to ask such clarifying questions. Clarifying questions are only required when there is ambiguity in the user query. The level of ambiguity raises the need for a clarifying question.

As seen in the related literature, existing attempts involved approaches to classify user intent based on pre-defined notions of query goals, which may not work in an open-domain setting. This work has explored methods to detect the level of ambiguity in the first user input in such a conversational setting. The initial query determines whether counter questions need to be asked at all and hence this work can be extended to include steps from multi-turn conversations.

The approaches started from simple ones including classifying queries using sentence classification techniques like CNNs without additional information. CNNs were then replaced by BERT based approaches, also in the absence of additional information to identify the intents the query can represent. This is where the documents retrieved by a search engine come into the picture. Documents cover multiple topics that may provide information about the range of user goals the query can represent. The more diverse the documents, the more the probability of the query being ambiguous. The Document Graph approach makes use of the similarities between document vectors to generate a graphical structure that can be processed by a graph convolutional network to extract the relationship and diversity of the document set. This approach requires the document embedding model, namely a BERT model followed by a pooling layer to be

finetuned alongside the GCN. This aims to optimize the document embeddings for the downstream task. The output of the GCN is concatenated with a vector representation of the initial query, which adds additional context. The Concept Graph approach draws from the work by [6] and adapts it for working on multiple documents. Keywords extracted from the documents are clustered to form concepts that are then used to group the retrieved documents. The graph data are formed by the similarities between the sets of documents assigned to individual concepts. A similar GCN as the Document Graph approach is now used to process the concept graph, but the context of the initial query is no longer added to its output.

Analysing the outputs of the various approaches explored, the performance of the CNN based model varies with the number of kernels and the receptive field. In the limited experiments performed, a CNN layer with 64 kernels covering 3 words each, was found to be optimal. This model was successful in classifying most of the class 1 and 2 queries but faltered with classes 3 and 4. The Document Graph approach had a proclivity towards the class 2 possibly due to the reasons explained in the Section 5.5: mainly due to the scope of documents covered and the embedding technique used to represent the documents. The Concept Graph approach similarly had a tendency towards the class 3, due to mandatory fixed number of concepts.

The approaches presented in this work suffered from a lack of training data as well as sufficiently diversified validation or testing data. The distribution of classes in the training and development datasets show a skew towards classes 2 and 3. From the confusion matrices of the explored approaches, it is evident that the models have problems in identifying completely ambiguous (level 4) and completely clear (level 1) queries. Queries of level 2 and 3, even on misclassification, are being mostly restricted to those 2 classes only and hence may not pose much difficulties. The models thus need to be trained to identify the extreme classes clearly. Additionally, due to memory requirements, the documents in the Document Graph approach had to be represented as a concatenation of keywords, which is not an optimal representation for BERT based embedding. In spite of that, only a limited number of documents could be considered for each query. On the other hand, this work also analyses the positive results and it is seen that the models are successful in correctly classifying a majority of queries. In fact, the concept graph approach, with a larger view of the document space offers greater precision at its best performance.

There are multiple ways in which these models can be improved. The Document Graph approach will benefit from improvements to the document embedding technique and sufficient computational and memory power to take more documents into consideration for each query. For further improvement of the Concept Graph approach, similarities between concepts and documents can be explored further through methods inspired by the matching vector of [6], in order to add more context for the graph convolutional networks to process. Additionally, keyword clustering can be made independent and algorithms that do not require the number of clusters to be pre-defined, can be used. The keygraph model proposed in [6] can also be modified for use with multiple documents, such that the number of concepts can be adapted for each set of documents and concepts which do not have any documents associated with them can be eliminated. For both the approaches, processing a larger number of documents for each query, within limits of reasonable time complexity can lead to better results and sufficiently large and diversified training and validation datasets can help train parameters and finetune hyperparameters properly for the models to generalize better. Apart from this, multiturn conversations can be taken into account by adding context vectors that encode key information from the previous steps in the conversation.

This work has explored a number of methods to classify the level of ambiguity in the first query raised by a user when they want to search for something. The experiments and the results demonstrate the fact that generic sentence classification algorithms can only have limited success in this field of application when the classification problem cannot be solved using only the words in the sentence. Retrieved documents can prove to be an important source of information when deciding if a query is ambiguous. However, the documents need to be represented properly and a sufficiently diverse set of documents need to be taken into account to correctly explore all the subjects covered. Such relationship between documents can be represented as graphs and as shown in this work, graph convolutional networks show promise when processing such data. However, the processes can be improved and better document representation techniques can be utilized to model the graphs, including developing a better method for calculating the edge weights, augmenting conventional similarity measures. Finally, for the system to work properly in open-domain settings, the model needs to be trained on a sufficiently diverse dataset. This work provides an important first step towards detecting ambiguity in the first turn of a multiturn conversational encounter and can be integrated into an end to end system for detecting query ambiguity and generating clarifying questions.

# Bibliography


[1] Terrance Goan, S. Henke, Nels E. Benson, and Oren Etzioni. A grammar inference algorithm for the world wide web. 2002.

[2] Johannes Kiesel, Arefeh Bahrami, Benno Stein, Avishek Anand, and Matthias Hagen. Toward voice query clarification. In *SIGIR*, pages 1257–1260, 2018.

[3] Andrei Broder. A taxonomy of web search. *SIGIR Forum*, 36(2):3–10, September 2002.

[4] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

[5] Ruihua Song, Zhenxiao Luo, Ji-Rong Wen, Yong Yu, and Hsiao-Wuen Hon. Identifying ambiguous queries in web search. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, page 1169–1170, New York, NY, USA, 2007. Association for Computing Machinery.

[6] Bang Liu, Di Niu, Haojie Wei, Jinghong Lin, Yancheng He, Kunfeng Lai, and Yu Xu. Matching article pairs with graphical decomposition and convolutions, 2019.

[7] Daniel E. Rose and Danny Levinson. Understanding user goals in web search. In *Proceedings of the 13th International Conference on World Wide Web*, WWW '04, page 13–19, New York, NY, USA, 2004. Association for Computing Machinery.

[8] Uichin Lee, Zhenyu Liu, and Junghoo Cho. Automatic identification of user goals in web search. In *Proceedings of the 14th International Conference on World Wide Web*, WWW '05, page 391–400, New York, NY, USA, 2005. Association for Computing Machinery.

[9] Bernard J. Jansen, Danielle L. Booth, and Amanda Spink. Determining the informational, navigational, and transactional intent of web queries. *Information Processing and Management*, 44(3):1251–1266, May 2008.

[10] Lynda Tamine, Mariam Daoud, Ba-duy Dinh, and Mohand Boughanem. Contextual query classification in web search. In Joachim Baumeister and Martin Atzmüller, editors, *LWA 2008 - Workshop-Woche: Lernen, Wissen & Adaptivität, Würzburg, Deutschland, 6.-8. Oktober 2008, Proceedings*, volume 448 of *Technical Report*, pages 65–68. Department of Computer Science, University of Würzburg, Germany, 2008.

[11] Jan Trienes and Krisztian Balog. Identifying unclear questions in community question answering websites. *Advances in Information Retrieval*, page 276–289, 2019.

[12] Egbert Ammicht, Alexandros Potamianos, and Eric Fosler-Lussier. Ambiguity representation and resolution in spoken dialogue systems. In Paul Dalsgaard, Børge Lindberg, Henrik Benner, and Zheng-Hua Tan, editors, *EUROSPEECH 2001 Scandinavia, 7th European Conference on Speech Communication and Technology, 2nd INTERSPEECH Event, Aalborg, Denmark, September 3-7, 2001*, pages 2217–2220. ISCA, 2001.

[13] H. Hashemi. Query intent detection using convolutional neural networks. 2016.

[14] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537, 2011.

[15] Yoon Kim. Convolutional neural networks for sentence classification, 2014.

[16] John McCarthy. What is artificial intelligence? 2004.

[17] IBM Cloud Education. What is artificial intelligence (ai)? `https://www.ibm.com/cloud/learn/what-is-artificial-intelligence`.

[18] Alan M Turing. Computing machinery and intelligence. In *Parsing the turing test*, pages 23–65. Springer, 2009.

[19] IBM Cloud Education. What is machine learning? `https://www.ibm.com/cloud/learn/machine-learning`.

[20] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition, 2010.

[21] H B Barlow. Unsupervised learning: introduction. In Geoffrey E. Hinton and Terrence Joseph Sejnowski, editors, *Unsupervised Learning: Foundations of Neural Computation*, pages 1–17. Bradford Company Scituate, MA, USA, 1999.

[22] T.K. Moon. The expectation-maximization algorithm. *IEEE Signal Processing Magazine*, 13(6):47–60, 1996.

[23] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio. Object recognition with gradient-based learning. In D. Forsyth, editor, *Feature Grouping*. Springer, 1999.

[24] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[25] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[26] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. *CoRR*, abs/1704.01212, 2017.

[27] Thomas Kipf. Graph convolutional networks — thomas kipf — university of amsterdam. `https://tkipf.github.io/graph-convolutional-networks/`. (Accessed on 07/30/2021).

[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[29] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[30] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *CoRR*, abs/1508.07909, 2015.

[31] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. 2019.

[32] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. ELECTRA: pre-training text encoders as discriminators rather than generators. *CoRR*, abs/2003.10555, 2020.

[33] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *CoRR*, abs/1908.10084, 2019.

[34] Yoav Goldberg and Graeme Hirst. *Neural Network Methods in Natural Language Processing*. Morgan & Claypool Publishers, 2017.

[35] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[36] Quoc V. Le and Tomás Mikolov. Distributed representations of sentences and documents. *CoRR*, abs/1405.4053, 2014.

[37] Rada Mihalcea and Paul Tarau. TextRank: Bringing order into text. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 404–411, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[38] Hassan Sayyadi and Louiqa Raschid. A graph analytical approach for topic detection. *ACM Transactions on Internet Technology (TOIT)*, 13, 12 2013.

[39] Mohammad Aliannejadi, Julia Kiseleva, Aleksandr Chuklin, Jeff Dalton, and Mikhail Burtsev. Convai3: Generating clarifying questions for open-domain dialogue systems (clariq), 2020.

[40] Mohammad Aliannejadi, Hamed Zamani, Fabio Crestani, and W. Bruce Croft. Asking clarifying questions in open-domain information-seeking conversations. *CoRR*, abs/1907.06554, 2019.

[41] Clueweb09. `https://lemurproject.org/clueweb09/`.

[42] Clueweb12. `https://lemurproject.org/clueweb12/`.

[43] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

[44] Stuart Rose, Dave Engel, Nick Cramer, and Wendy Cowley. *Automatic Keyword Extraction from Individual Documents*, pages 1 – 20. 03 2010.

[45] spacy. `https://spacy.io/usage/spacy-101`.

[46] Janek Bevendorff, Benno Stein, Matthias Hagen, and Martin Potthast. Elastic chatnoir: Search engine for the clueweb and the common crawl. In Gabriella Pasi, Benjamin Piwowarski, Leif Azzopardi, and Allan Hanbury, editors, *Advances in Information Retrieval*, pages 820–824, Cham, 2018. Springer International Publishing.

# Declaration

I hereby declare that the work presented in this thesis is entirely my own. I did not use any other sources and references that the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted hard copies.

01.08.2021, Avik Banerjee