4-25-2023 11:30 AM

INVESTIGATING IMPROVEMENTS TO MESH INDEXING

Anurag Bhattacharjee, *Western University*

Supervisor: Mercer, Robert E., *The University of Western Ontario*

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

Abstract

The MEDLINE database currently comprises an extensive collection of over 35 million citations, with more than 1 million records being added each year [28]. The abundance of information available in the database presents a significant challenge in identifying and locating relevant research articles on a given search topic. This has prompted the development of various techniques and approaches aimed at improving the efficiency and effectiveness of information retrieval from the MEDLINE database. A search engine devoted to the research publications on MEDLINE is called PubMed. MeSH, or Medical Subject Headings, is a restricted vocabulary used by PubMed to categorize each article. Human annotators have been used for decades, which is not only time-consuming but also expensive. Due to its enormously complex hierarchically ordered structure, MeSH indexing is a difficult problem in the machine learning domain. We propose a model which addresses all these challenges. We propose an end-to-end model that takes MeSH description into account and combines it with a Knowledge Enhanced Mask attention model to index new research papers. We also calculated the journal correlation of each MeSH term in the MeSH hierarchy.

Keywords: text classification, MeSH term indexing, graph neural network, multi label classifier, LSTM, biLSTM, BERT embedding, deep learning, convolutional neural network, recurrent neural network, attention model.

Summary for Lay Audience

The growth of research in the medical field has resulted in an overwhelming number of research papers, making it challenging to find relevant articles on a particular topic. PubMed, a scientific and biomedical research paper search engine, categorizes each research paper using Medical Subject Headings (MeSH) terms. These MeSH terms serve as tags to categorize research articles. Traditionally, human annotators have manually tagged each article according to its relevant MeSH term, which is a time-consuming and costly process. To address these challenges, we propose an end-to-end model that can automatically analyze a new research paper based on its abstract and title and label it with the relevant MeSH terms.

In this thesis, we aimed to simplify the extraction of MeSH terms by considering their structure and descriptions to determine their semantic meaning. We analyzed research papers in the PubMed database based on their abstracts and titles. By matching the semantic meaning of MeSH words to the text analysis, we automatically assigned the relevant MeSH terms to each paper.

Acknowledgements

I have received immense support and assistance throughout the experiment and writing of this dissertation. My deepest appreciation and sincere gratitude go to my honorable supervisor, Dr. Robert E. Mercer. His expertise in this field and his invaluable feedback have made this project a success.

I could not have taken the journey without the support of my colleague Xindi Wang. I want to thank her for giving me access to her research work and guiding me throughout the experiment.

I would like to thank my parents for their support to pave my way from the beginning of my educational journey and help me reach here. I am also extremely grateful and highly indebted to my wife Atmaja Bhattacharjee for her immense support throughout my whole journey.

Finally, I want to thank my friend Sudipta Singha Roy and my lab mates for their support and valuable suggestions.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The automatic classification of texts has been a long-standing goal of machine learning. Text classification is the task of assigning a predefined category or label to a piece of text. It involves training a model on labeled examples to learn how to recognize the relevant features of each category. Multi-label text classification extends this task by allowing a piece of text to be assigned multiple labels, rather than just one. This thesis investigates some possible improvements to a state-of-the-art multi-labeling deep learning model, KenMeSH [40]. KenMeSH labels biomedical journal papers with multiple Medical Subject Heading (MeSH) terms.

KenMeSH was developed in our lab and achieved state-of-the-art results in the MeSH indexing task. As we have continued to explore ways to improve MeSH indexing, we have experimented with updating different modules of KenMeSH through research and evaluation. These updates have been a gradual progression. We can divide our development process into three distinct phases, which we review in this thesis. In the end, we arrived at a new model architecture for MeSH indexing which we have named BERTKenMeSH.

In this introductory chapter, we will delve into the overall multi-label text classification task, and in particular, the critical research area of MeSH indexing which has attracted significant attention in recent years. We will provide a comprehensive overview of the background of the research question and the significant developments made in this field so far. Our aim is to highlight the importance of this research topic and the need for continued exploration and innovation in this area. We will also discuss our contribution towards improving the KenMeSH model and provide a layout of this dissertation.

1.1 Background

With the widespread availability of the internet and technological advancements, there has been a significant increase in digital data. This has created a new challenge for computer

scientists to effectively retrieve relevant information from the vast volume of data available in various fields. In the past, human annotators were relied upon to classify and index the data manually. However, this process cannot keep up with the constantly growing volume of data. To address this challenge, machine learning, and natural language processing techniques have been developed. Text classification and indexing have emerged as a popular research topic in the field of natural language processing, aimed at improving the efficiency and effectiveness of retrieving data.

Text classification is a powerful process that can automatically analyze text and organize it into predefined categories. It has a wide range of applications, including semantic analysis, topic detection, and language detection. Text classification has revolutionized how we interact with information by enabling curated search results, automated spam detection in emails, and other applications that make our lives easier. This technology is particularly useful because it can process vast amounts of information in real-time, which would be impossible for humans to do alone. Additionally, text classification ensures that documents are consistently categorized, making them easier to retrieve and analyze.

The growth in the number of scientific articles in various fields has been unprecedented in recent years. The availability of these papers in digital libraries has revolutionized the way researchers access and share information. Digital libraries not only provide resource discovery, metadata searching, and reference services but also act as worldwide knowledge networks that enhance the quality of academic study and life. Platforms like PubMed [29], which hosts research articles in the biomedical domain, provide sophisticated searching abilities to support researchers in their work. PubMed is a vital platform for biomedical research, containing a vast number of research articles. As of April 2023, PubMed contains over 35 million citations and abstracts for biomedical literature, with over 1 million new records added annually [28]. With such an enormous amount of data, indexing and categorizing these articles have become increasingly important for efficient and effective information retrieval. The use of MeSH (Medical Subject Headings) indexing is crucial for the accurate categorization of articles and precise search results. MeSH indexing assigns standardized subject terms to articles to facilitate searching and organizing articles based on the concepts they cover. This ensures that relevant articles are retrieved in a search and improves the accuracy and efficiency of literature searches in biomedical research.

The vast volume of free data in digital libraries is essential for study, hence it's crucial to properly index them. The processing of documents with searchable tags improves the organizational productivity and usefulness of digital libraries. Manual indexing is ineffective and expensive to do. The study of automatic document indexing is a burgeoning field of research.

In this thesis, we will deal with biomedical document indexing. To accomplish this task

Figure 1.1: Text Classification Workflow

we will experiment on several deep neural network models and apply those on biomedical documents gathered from PubMed digital library. In Chapter 2 we will discuss some of the machine learning models that we have considered and studied during the process.

1.2 Research Question

In this thesis, we deal with text classification and categorization of biomedical documents. This text classification task is a multi-label classification task. We investigate some possible improvements to a state-of-the-art multi-labeling deep learning model, KenMeSH [40], which labels biomedical journal papers with multiple Medical Subject Heading (MeSH) terms.

1.2.1 Text Classification

Text classification is an important research topic in machine learning field. Research is being done to increase the accuracy of text classification. This has tremendously changed the way we can categorise a vast amount of data. Prior to the development in this sector human annotators used to read a document, interpret the meaning and assign labels to the documents. But this was not only time consuming but also unmanageable as the categories might not be similar for all annotators. Text classification classifies texts in certain categories with predefined labels. A typical workflow of a text classifier is show in Figure 1.1.

Some of the common use cases of text classification are:

- **Sentiment analysis** is the process of understanding if a given text is talking positively or negatively about a given subject.

- **Knowledge organization** covers document categorization and indexing in databases, libraries, etc. It arranges content in a way that increases the effectiveness of user queries, like the library taxonomy system, which, for instance, minimises the amount of time users spend navigating libraries.

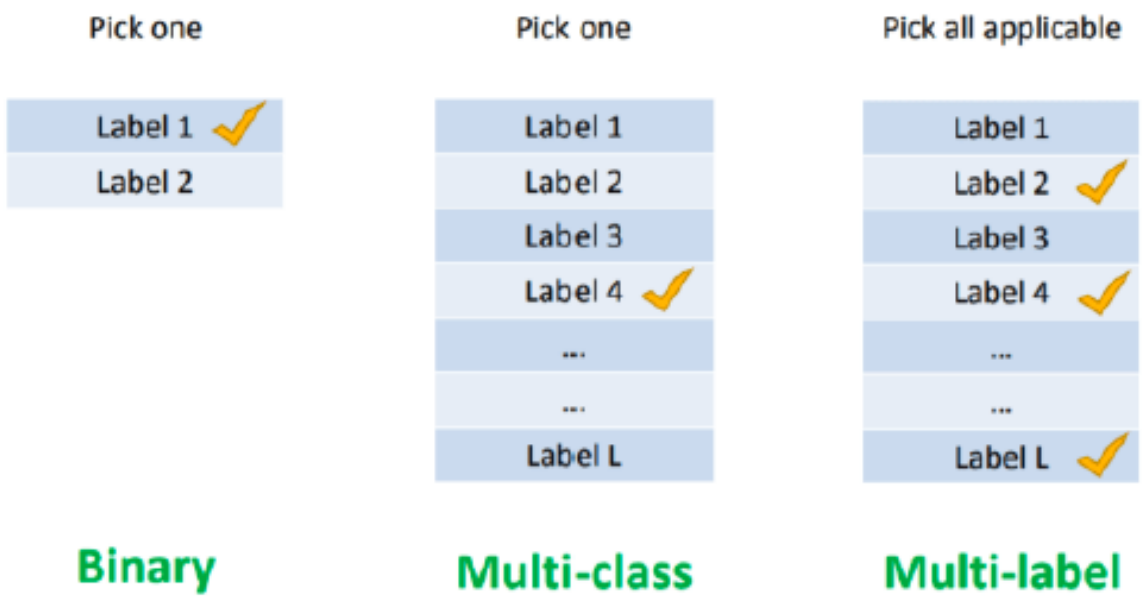


Figure 1.2: Different type of Text Classifications.

- **Topic detection** is a mechanism for grouping subjects and tracking their emergence; it aids users in managing information overload.

- **Information filtering** is the process of eliminating information that is unnecessary or irrelevant from the information flow. The need for more pertinent information is always growing since individuals may now get information from a variety of sources. Artificial information filtering technology is urgently needed to satisfy user demands for text categorization and filtering.

We can categorise the text classification process in three different categories: multi-class classification, multi-label classification, and binary classification. Multi-class classification divides items into at least three classes, while binary classification divides elements into one of two categories. Multi-label classification divides elements into a set with at least two target labels. A visual representation of Binary classification, multi-class classification and multi-label classification are distinguished in Figure 1.2. Multi-label categorization in biomedical papers is the topic of our thesis. Multi-label categorization will be elaborated in the next subsection.

1.2.2 Multi-label Classification

Multi-label classification classifies documents according to a set of target labels. These labels are mutually exclusive. For a training set of n document-label pairs $(x_i, y_i)_{i=1}^{n}$, where $x_i \in \chi \in \mathbb{R}^D$ and $y_i \in \{0, 1\}^L$, D is the number of document features and L is the number of total labels. Each document x_i is associated by relevant label from the label vector y_i. The goal of multi-label classification is to find the most relevant subset of labels from the space of categories for each document, which is $X \rightarrow \{0, 1\}^L$.

1.2.3 Medical Subject Headings Indexing

PubMed and MEDLINE are two most prominent resources of biomedical articles. They are maintained by the United States National Library of Medicine (NLM). Bibliographic data for papers in a variety of life sciences and biomedical areas, including medicine, health care, biology, biochemistry, and molecular evolution, are available in the MEDLINE database. The database includes more than35 million entries from more than 5,200 publications throughout the world.[1] In 2020 (the last available full year data), MEDLINE received just under one million new citations, or slightly less than 3,000 changes every day.[2] PubMed is a web server that can freely access the MEDLINE database of references and abstracts. Some PubMed records have full text articles available on PubMed Central [40].

Medical Subject Heading (MeSH) terms are included in an NLM's controlled vocabulary thesaurus. They are used to index journal articles in MEDLINE. Biomedical papers in NLM databases are categorised using the hierarchically ordered terminology indexing system known as MeSH. MeSH, version 2018, has 28,939 headings [40]. There are 29 check tags, a unique subset of MeSH words that describe research topics, among these MeSH terms. MeSH Indexing for the NLM was created by human annotators at great financial expense. It's crucial to investigate the automated indexing approach to cut costs and boost efficiency. Figure 1.3 shows a snapshot of the MeSH term hierarchy.

1.3 Contributions

Below are the contributions of our thesis:

- We explored various deep-learning neural networks to address the multi-label classification problem.

[1] `https://www.nlm.nih.gov/medline/medline_overview.html`
[2] `https://www.nlm.nih.gov/bsd/medline_cit_counts_yr_pub.html`

Figure 1.3: Tree structure of a branch in MeSH

- We created a corpus of all MeSH terms along with their descriptions.
- We enriched node features to create a more informative graph structure from the MeSH hierarchy. To achieve that we created BERT embedding from the description of each MeSH term and used them as node features.
- We introduced a new model architecture, BERTKenMeSH, by updating the model architecture of KenMeSH. We used PubMedBERT as a feature extractor instead of BioWordVec and Bi-LSTM.
- An important aspect of our project was refactoring the code. We made the code maintainable, more understandable, and better structured. We completely redesigned the codebase using PyTorch Lightning.
- We implemented a utility function to convert a large matrix into a sparse dictionary. We also implemented another utility function that can read the sparse dictionary and convert them into a desired length matrix in the data loading process of the training. It is a significant help in reducing the time to load the data and saved a lot of memory.
- We researched and experimented on various neural nets and different combinations of dilated CNN to find the optimal settings that complement the use of BERT word embeddings for the multi-label classification task.

- We provide an architecture to apply label-wise attention to the BERT output.

1.4 Structure of this document

The purpose of this thesis is to improve automatic MeSH indexing. We have stated our problem and the background behind choosing this topic. In Chapter 2 we will elaborately discuss all the machine learning models, deep learning models, text classification, text representations that were studied to implement the proposed MeSH indexing framework. We will also discuss some background of MeSH indexing and our current attempt to improve them. In Chapter 3, we will discuss the dataset creation and setup of our project. We will also provide information on the challenges faced during the implementation of the framework. Chapter 4 will discuss some evaluation techniques that were used to measure the accuracy of our proposed model and show the results and comparison with some other models. Finally, in Chapter 5, we will conclude our thesis with the milestones we achieved and the limitations that are left behind. We will then propose some of our future plans with this project.

Chapter 2

Related Work

The goal of this research is the automatic labeling of biomedical research papers. These labels are collected from PubMed Central which is a collection of all biomedical research papers and their related Medical Subject Heading (MeSH) Terms. MeSH terms are arranged in a hierarchical tree structure. MeSH terms and their structure is an important part of this research.

In this chapter, we will discuss all the topics that have influenced our research. These topics include text classification, label indexing, and several deep-learning models.

Automatic label indexing has always been a point of interest in research in the field of natural language processing. We will discuss briefly some related research on label indexing. We will also discuss traditional machine learning approaches in text classification and a few deep learning approaches that have been commonly used in classifying texts.

We will then move on to the graph convolution network and its usage in structuring the MeSH terms from the hierarchical tree structure.

2.1 Traditional Machine Learning Approaches in Text Classification

A traditional approach to classifying text is supervised learning. Supervised learning is also called statistical learning because of the use of labeled data sets to train the model. The model is actuated by adjusting the weights through cross-validation. Supervised learning can be divided into two parts, classification, and regression. We will focus on the classification of data.

Text classification can be divided into four phases. They are text processing, feature extraction, training the classifier, and classifying texts using the classifier model.

Text processing is the step to prepare the data for the machine learning process. It involves tokenizing sentences into small pieces of meaningful information. This process is a combi-

nation of chopping a sentence into words and normalizing the data to generate a meaningful uniform sequence. Normalization includes converting all words into lowercase and converting special symbols and numbers to text.

The next step is feature extraction. Removing stop words, and finding out words that might bear significant data related to the document are some of the steps to complete feature extraction from a sequence of texts. Feature extraction helps us extract a subset of terms that describe the data before sending it to classifiers. Feature extraction is a very important step and it may vary according to the design of the responsible engineer and the quality of the data.

The next step is training the classification model. The choice of the machine learning algorithm and preparing the model can contribute immensely to classifying the data. Every model offers some advantages and some drawbacks. Some of the notable classification models are linear classifiers, support vector machines (SVM), naïve Bayes , decision trees, k-nearest neighbors, and random forests [39].

At this point, the model is ready for classifying data based on its training. The model can be evaluated with test data to determine the accuracy of the model.

2.2 Deep Learning Approaches in Text Classification

A deep learning model is an artificial neural network that can learn and extract complex patterns from data using multiple layers. The main inspiration behind deep learning is the human brain. Deep learning has opened a gateway toward the future of computing. Driverless cars can distinguish between a pedestrian and a street light immediately. Voice commands can now control most of the appliances on the go. These two are examples of the deep learning model's application in vision and audio. Deep learning has also enabled various prospects in text processing that was never possible before. Deep learning models need a huge amount of data to train and this has only become possible because of the advancement of computing. Deep learning models can now achieve state-of-the-art accuracy, sometimes exceeding human-level performance, on many tasks.

2.2.1 Artificial Neural Networks in Text Classification

ANN is a biologically inspired computational model which has exceeded the performance of previously available traditional machine learning models[31]. ANN was designed by keeping the structure of human brain cells in mind. A human brain cell is called a neuron.

An artificial neural network usually consists of an input layer, an output layer, and single to multiple hidden layers. A multi-dimensional vector is loaded as an input to the input layer

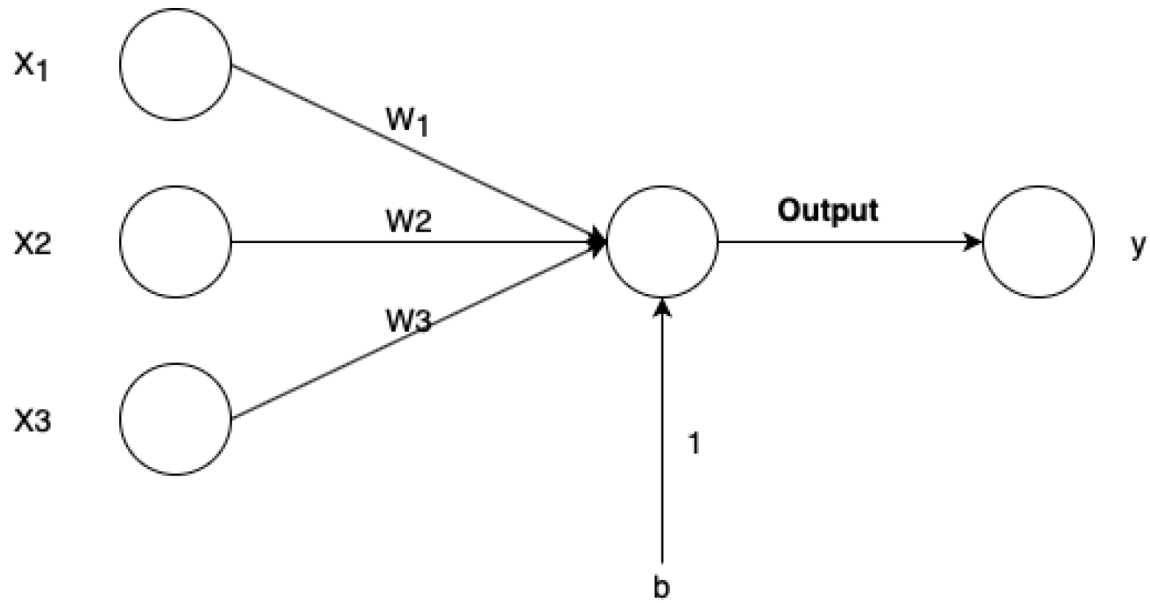


Figure 2.1: A perceptron. (Source: [40]).

and later distributed to hidden layers. Hidden layers learn from their previous hidden layers and adjust the weight based on the errors in the output to improve the accuracy of the final output. This process is called backpropagation, and it allows the neural network to learn from the training data and optimize its output.

2.2.2 Multi-layer Perceptrons

A perceptron can be considered the core base of neural networks which can be compared to a neuron. The perceptron is a machine learning algorithm, which takes input features $x1, x2, ..., xn$ and computes a weighted sum of these inputs. It then applies a threshold function to determine the output y, which is a binary value. Figure 2.1 shows the architecture of a basic perceptron.

To describe the perceptron in the figure, it has three inputs: $x1, x2, and x3$, each input x_i has a weight assigned according to the importance of the corresponding input. The output $h_{w,b}$ is defined as:

$$h_{w,b}(x) = f\left(W^T x\right) = f\left(\sum_{i=1}^{m} w_i x_i + b\right)$$

In the equation, $x_i \in R^d, W = \{w_1, w_2, ..., w_m\}$, where $w_1 \in \mathbb{R}^D$, b is the bias and f is an activation function. Non-linear functions, such as the sigmoid function, hyperbolic tangent function (tanh), and Rectified Linear Units (ReLU), are commonly used as activation functions in neural networks. Their curves are shown in Figure2.2.

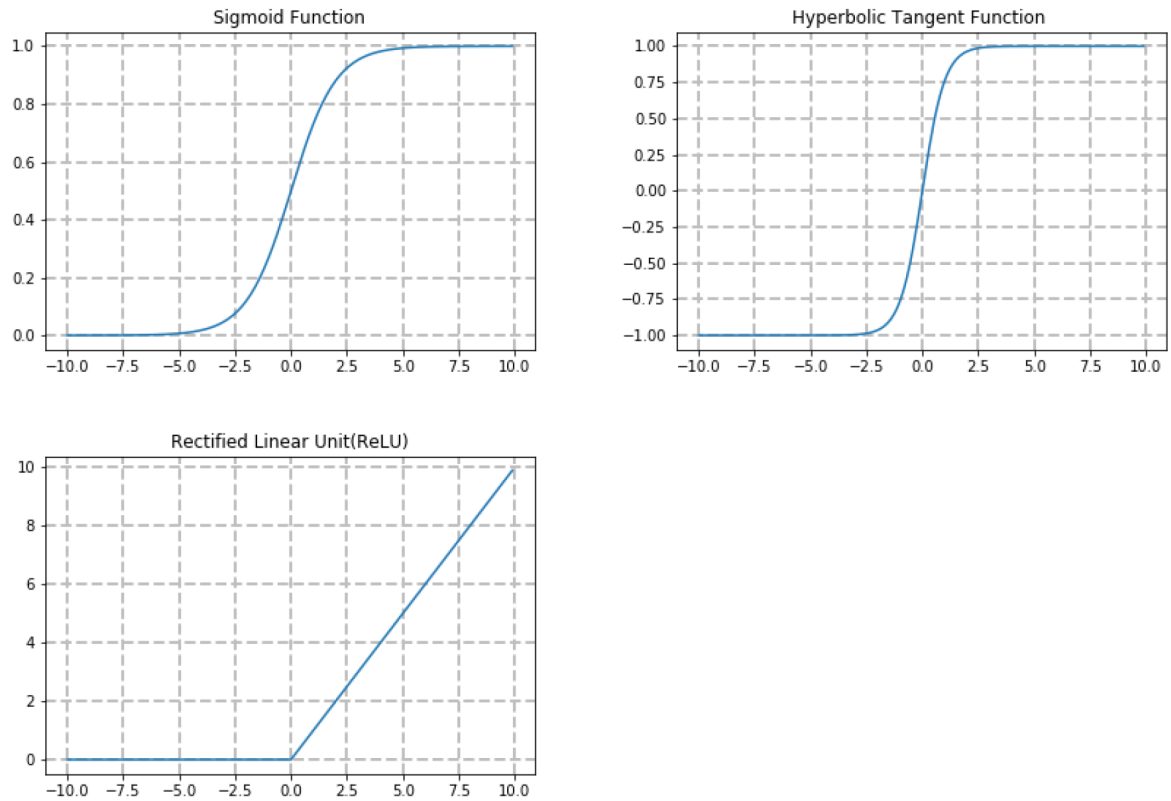


Figure 2.2: Activation Functions. (Source: [40]).

A neural network consists of multiple layers of perceptrons and they are connected in a feed-forward way. That is why this kind of basic neural network is also called Multilayer Perceptrons (MLPs). And, because they are arranged in a feed-forward fashion they are also called Feed-forward neural networks. The aim of such networks is to learn a function that maps $y = f(x, \theta)$ through learning rate θ. A three-layered feed-forward network is illustrated in Figure 2.3.

2.2.3 Convolutional Neural Network

The Convolutional Neural Network (CNN) is an impressive evolution of the Artificial Neural Network (ANN). A CNN consists of convolutional layers which makes it different from other multi-level perceptrons (MLPs). This special quality helps CNN to detect patterns from input data. This is why CNN is widely used for analyzing images and in the field of image processing.

An important aspect of convolutional layers is the filter. A filter is capable of detecting patterns from inputs and gives CNN the special quality to distinguish itself from other MLPs.

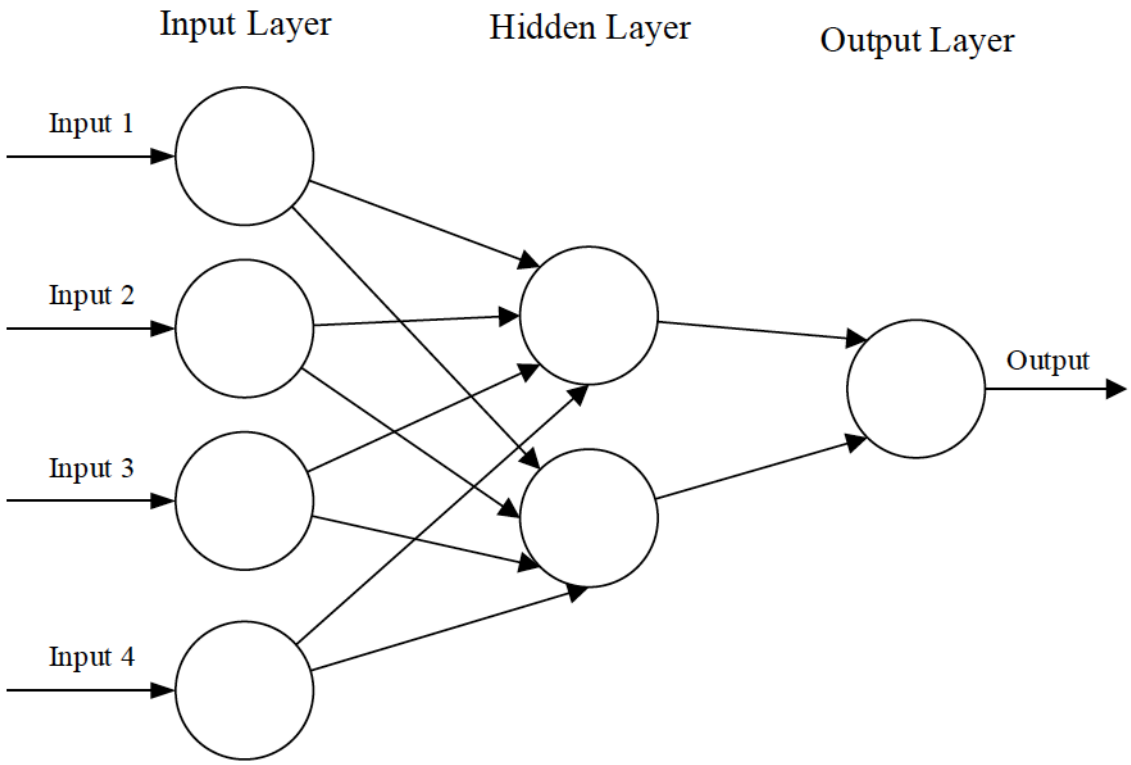


Figure 2.3: A simple three-layered feed-forward neural network (FNN), comprised of an input layer, a hidden layer, and an output layer. This structure is the basis of several common ANN architectures, including but not limited to feed-forward Neural Networks (FNN), Restricted Boltzmann Machines (RBMs), and Recurrent Neural Networks (RNNs). (Source: [31])

A filter is essentially a matrix. An engineer can determine the number of rows and columns a filter will have according to the need of a particular design. This matrix is filled up with random numbers at first. These filters then convolve along every pixel block of an input image, multiplies the values of the pixel block with the filter matrix values, and stores them to generate an output image for the next layer. The deeper the convolutional layers go, the more sophisticated these filters become. These filters might start with detecting patterns like edges, squares, rounds, etc., and can move along to detect objects. Figure 2.4 shows a simple convolutional operation on an Excel sheet. The input is a matrix representation of a handwritten digit 7. The 3×3 filter is applied on the input and the matrix multiplication for each pixel group is stored in the conv1 matrix to generate the output of the first convolution.

Some other important concepts in convolutional networks are stride and padding. Stride is the number of blocks the filter will move during convolution. But, when a filter is applied to an image matrix, the image shrinks and there is a possibility of losing some data. Padding is a way to prevent losing data by adding some extra dimension to the input matrix.

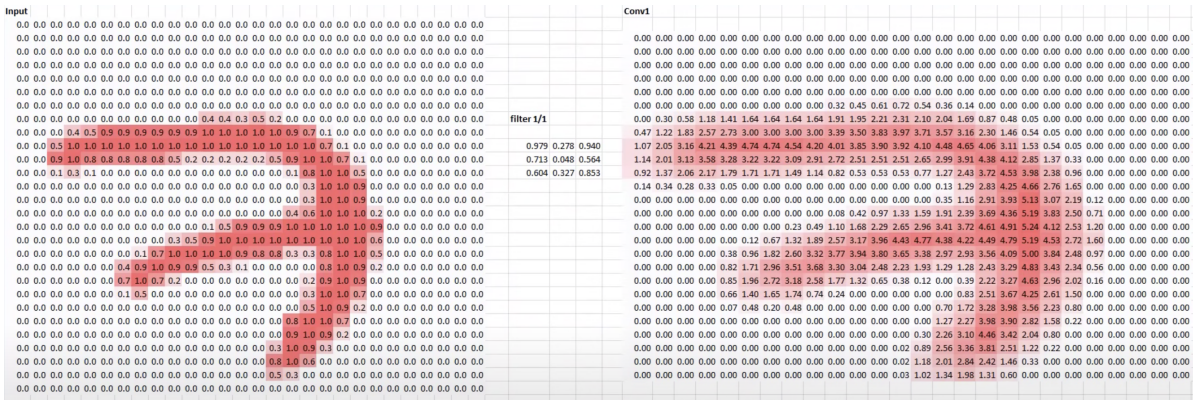


Figure 2.4: A simple convolutional operation. (Source: [13])

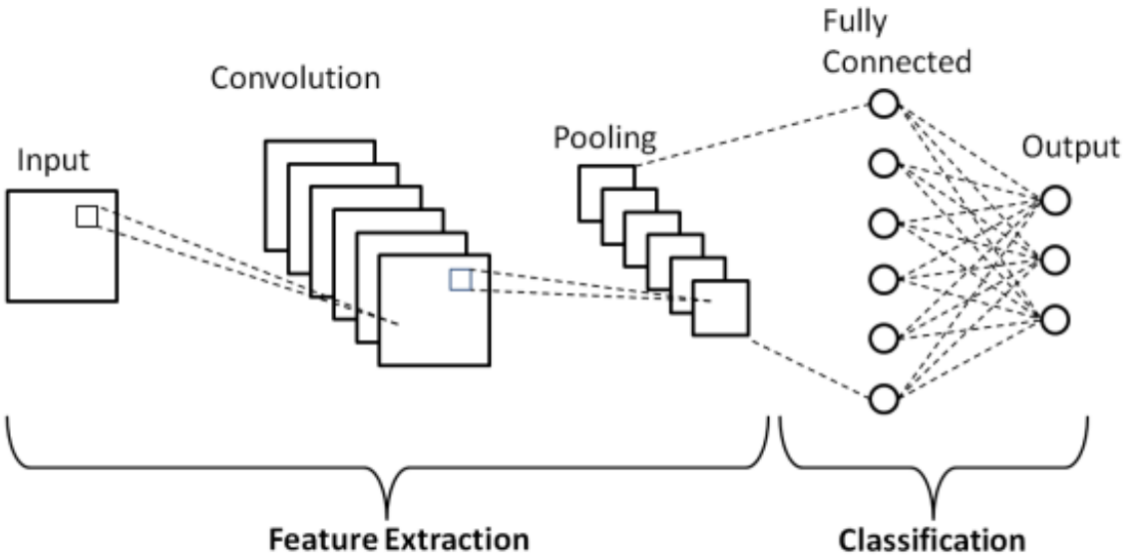


Figure 2.5: A typical CNN architecture. (Source: [14])

Other than the convolutional layer, a CNN architecture has two other important layers. They are the pooling layer and a fully-connected layer. A pooling layer is applied to the output of the convolutional layer. The output of a convolutional layer is a feature map. Pooling is a way to pick high-intensity pixels from the output to detect an object. In other words, the pooling layer reduces the dimensionality of a feature map by taking the maximum, average, or sum. A fully-connected layer computes a class score from activation functions. This is an important step to prepare the output for classification. A typical convolutional neural network with its five layers is shown in Figure 2.5.

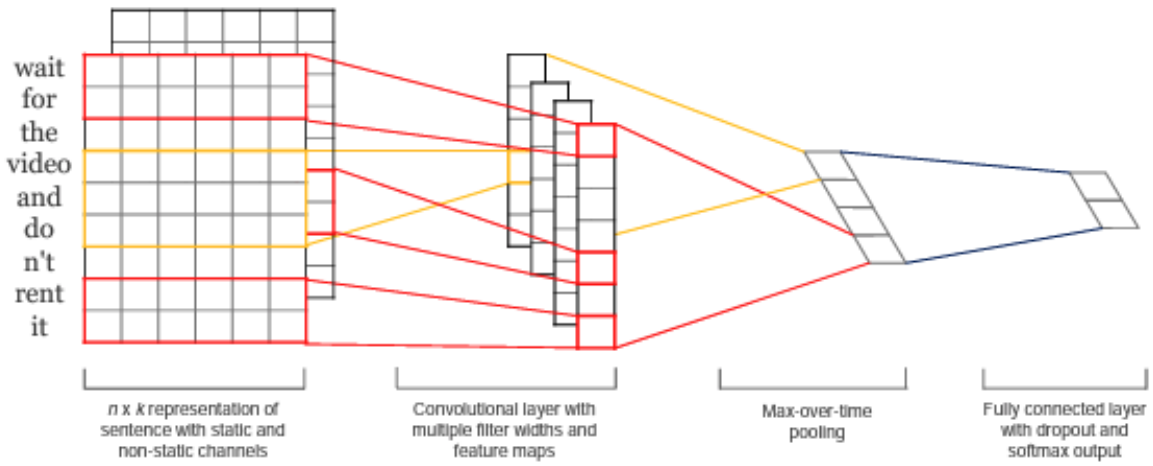


Figure 2.6: Model Architecture of TextCNN. (Source: [18])

2.2.4 Convolutional Neural Network in Text Classification

CNN is used a lot in image processing. But, CNN has also been successful in detecting patterns in texts. One of the most popular models used in text classification is TextCNN [10]. This proposed model has only one convolutional layer. Max pooling was used as a pooling layer. And the final layer is a fully-connected layer with dropout and softmax outputs. TextCNN used pre-trained embeddings on 100 billion words of Google News as an input [26]. The input data was essentially an embedded word-to-word vector. Multiple filters of various sizes were used to generate the feature maps from the input. Max-pooling was used on the feature maps to detect the most important features. The pooling process extracted one feature per filter. Those features were then passed on to the fully-connected layer, where softmax was used to output the probability distribution classes. Figure 2.6 shows the architecture of the above discussed TextCNN model. This model was a success in multiclass classification.

XML-CNN [21] was the first attempt to apply CNN on extremely large multi-label text classification. The XML-CNN architecture has some similarities with TextCNN except for the differences in two important layers of the model: the convolutional layer and the pooling layer.

The structure of XML-CNN is illustrated in Figure 2.7. XML-CNN also employs dense word embedding as input. The input document embedding matrix has a dimension of $n \times k$, where n is the length of the padded document and k is word embedding dimensionality. To extract the feature maps, multiple filters of variable sizes are applied during the convolutional layers. The major difference that stands XML-CNN apart from TextCNN is these holistic filters are applied to every position of a word to capture global features of the document. Max-pooling is used as a pooling layer and these pooling layers divide the feature maps into k

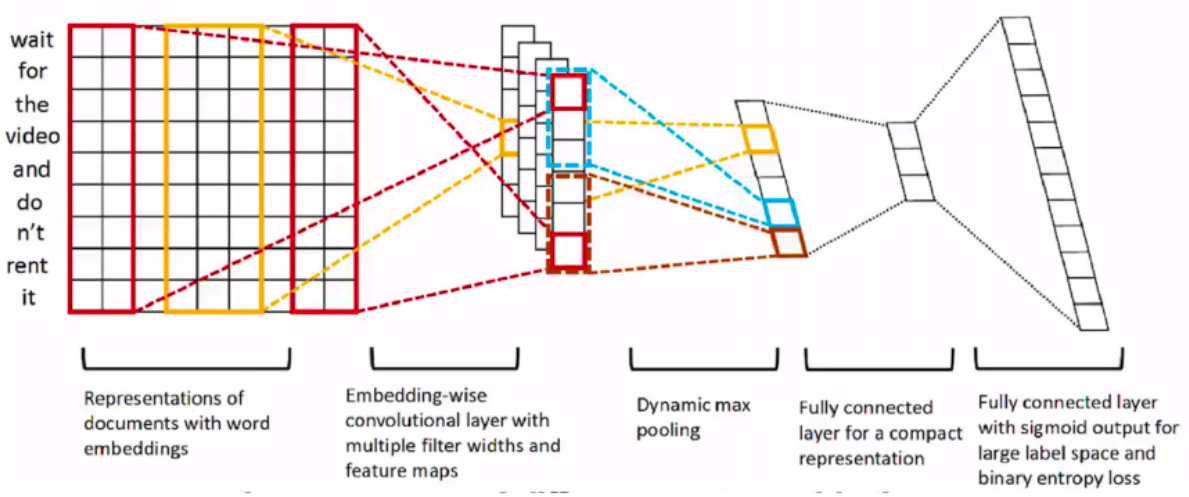


Figure 2.7: Model Architecture of XML-CNN. (Source: [21])

chunks, and collects the maximum value inside each chunk to construct the output. An extra layer is employed before sending this output to the fully connected layer. This layer is a hidden bottle neck layer and this layer helps to reduce the number of parameters from the output of the pooling layer. Finally, the fully connected layer is applied along with dropout and binary cross-entropy loss over sigmoid to output the classified result. This model is simple yet very successful in classifying multi-label text data.

2.2.5 Dilated CNN

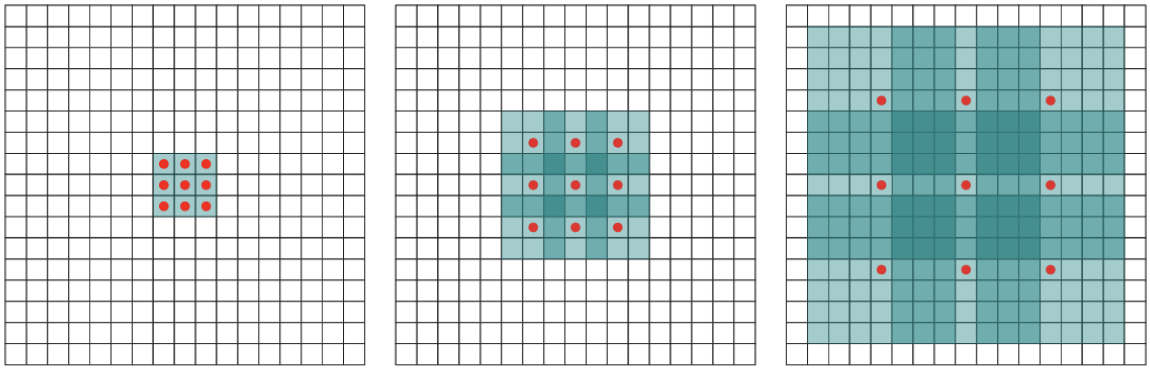
A particular kind of convolution called dilated convolutions enlarges the kernel by creating gaps between the subsequent kernel pieces. The amount by which the kernel is broadened is indicated by a second parameter called the dilation rate. In other words, the kernel skips (l-1) pixels depending on the value of this option. Typically, gaps are added between kernel components.

For $F : \mathbb{Z}^2 \to \mathbb{R}$ be a discrete function, let $\omega_r = [-r, r]^2 \cap \mathbb{Z}^2$ and let $k : \omega_r \to \mathbb{R}$ be a discrete filter of size $(2r + 1)^2$. The discrete convolution operator $*$ can be defined as

$$(F * k)(p) = \sum_{s+t=p} F(s)k(t)$$

We can now generalise this operator. For l to be a dilation factor and let $*_l$ be defined as

$$(F *_l k)(p) = \sum_{s+lt=p} F(s)k(t)$$



(a) (b) (c)

Figure 2.8: Systematic dilation supports exponential expansion of the receptive field without loss of resolution or coverage. (a) F_1 is produced from F_0 by a 1-dilated convolution; each element in F_1 has a receptive field of 3×3. (b) F_2 is produced from F_1 by a 2-dilated convolution; each element in F_2 has a receptive field of 7×7. (c) F_3 is produced from F_2 by a 4-dilated convolution; each element in F_3 has a receptive field of 15×15. The number of parameters associated with each layer is identical. The receptive field grows exponentially while the number of parameters grows linearly. (Source: [46])

Here, $*_l$ is referred to as a dilated convolution or an l-dilated convolution. The familiar discrete convolution $*$ is simply the 1-dilated convolution.

2.3 Recurrent Neural Network

The recurrent neural network (RNN) is a class of artificial neural networks that is popular for processing sequential data. When it comes to a meaningful sentence, processing a sentence only as words is not enough. Without maintaining the sequence a sentence loses its inherent semantics. Traditional deep neural networks often work at the word level and the semantics of the sentence is often lost. However, RNNs are designed to take a series of inputs without any predetermined fixed length. But the most unique concept of RNN is that they can process a sequence of data by retaining the previous state in the sequence, which helps it process a sequence by retaining the memory of the previous value. In other words, the RNN operation is recurrent, the output of the current operation will be the input for the next stage. Figure 2.9 shows a sample structure of the RNN architecture.

The figure illustrates how a simple RNN works. For any given time step t and an input x_t, the network generates a hidden state h_t as an output. This output is then forwarded to the next step as input. This is called the feedback loop in a Recurrent Neural Network. A typical step in RNN consists of Weight (W_t) and bias (b_t). These weights and biases are updated at

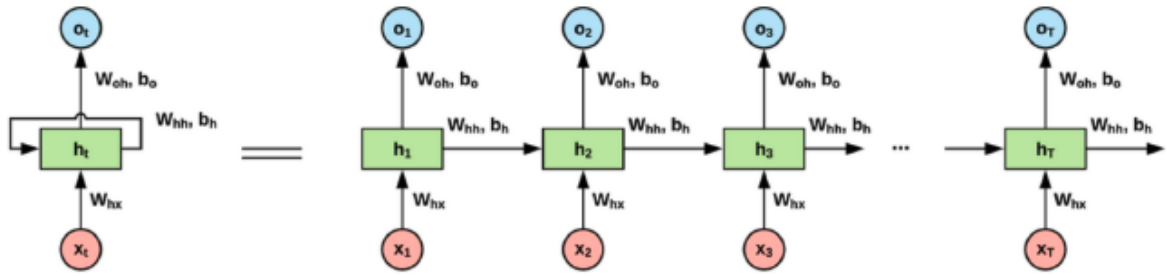


Figure 2.9: Sample RNN structure (Left) and its unfolded representation (Right). Source: [37]

each epoch of the training to achieve the desired accuracy. An activation function is also used to normalize the output at each step. An RNN is constructed as a series of multiple identical neural nets to process the information at different time steps. The working principle of an RNN can be explained by the following equations:

$$h_t = f\left(W_{hh}h_{t-1} + W_{hx}x_t\right)$$

$$o_t = W_{oh}h_t$$

For every time step t and hidden state h_t an output o_t is generated. f denotes the activation function. W_{hh} is the weight matrix and it denotes the weight between hidden states of two-time steps. h_{t-1} is the information or the hidden state from the previous time step carried forward to the current step. W_{hx} represents the weight between the input and the previous hidden states, whereas, W_{oh} is the weight matrix between the hidden state and the output. All these weights and biases are initialized with random values and they are updated to reach the accurate weight and bias for the operation via backpropagation through time (BPTT).

An RNN is perfect for classifying data where sequence carries a lot of semantic information. Language modelling, speech recognition, machine translation, image captioning are some of the many use cases where RNN really shines. In theory, the sequence in an RNN can be infinite which means it can deal with an infinitely large number of sequential data. But in application, when dealing with lengthy sequential data, the basic RNN model faces two problems: the vanishing gradient and the exploding gradient. In the vanishing gradient problem, the gradient shrinks dramatically, making it difficult for the weights to change their values and perhaps putting a halt to neural network training. As an illustration, if we refer to the equation:

$$\begin{cases} x^{(n)} = W^n x^{(0)} & x^{(i)}, W \epsilon \mathbb{R} \\ & i \in [0, n] \end{cases}$$

$$W^n x^{(0)} \rightarrow \begin{cases} \infty; & W > 1 \\ 0; & W < 1 \end{cases}$$

These equations illustrate that, for a very large number of n, if the value of W is greater than 1, it will eventually explode, otherwise if the value of W is less than 1, it will gradually diminish until it vanishes. This value directly affects the gradient value.

Several other models, including Leaky Recurrent Unit, Gated Recurrent Units (GRU), and Long Short-Term Memory (LSTM), have been put forth to address these problems. We can see these models as gradual progressions towards addressing these two problems. We consider only LSTM below since it is has importance later in the thesis.

2.3.1 Long Short-Term Memory

Long Short-Term Memory [12] architecture is a progression of RNN model that has been designed to overcome the shortcomings of the basic recurrent neural network. LSTM maintains a similar chain like architecture like RNN, but instead of a hidden unit, LSTM replaces it with an unit called an LSTM cell and the outcome from each cell state is called a cell state. So, in an LSTM every state stores a cell state along with the hidden state. The fundamental benefit of a cell state is that LSTM cells have the option to read from the previous cell state, write to it, or reset it at each time step, using a gating mechanism. This option is available at every unit of the network. A bare LSTM structure compared to basic RNN is illustrated in Figure 2.10. Here C denotes the cell state for each cell.

An LSTM cell is constructed with three gates. This gates can be considered as binary gates: Input gate, Forget Gate and Output gate. Forget gate decides which information needs to be stored and which information can be forgotten. This sigmoid layer takes h_{t-1} and x_t as input and generates a number between 0 and 1 for each number in the previous cell state (C_{t-1}). This layer generates 0 to tell the cell state to forget the information completely and 1 to preserve it. Eqn. 2.1 demonstrates how the forget gate works. The next step is divided into two parts. The first part is a sigmoid layer named the Input Gate that decides whether the value needs to be updated. The second part is a tanh layer that helps in producing a vector that will be passed to the cell state value $\tilde{C}_t$ (Eqn. 2.2, 2.3). These two parts of this state decide the importance of a new information. Then the cell value C_{t-1} is multiplied with the forget gate value f_t to remove unnecessary information of the past. This information along with the multiplication of input gate value and candidate cell value are summed up to form a new cell value C_t (Eqn. 2.4).

$$f_t = \sigma\left(W_f\,[h_{t-1}, x_t] + b_f\right) \tag{2.1}$$

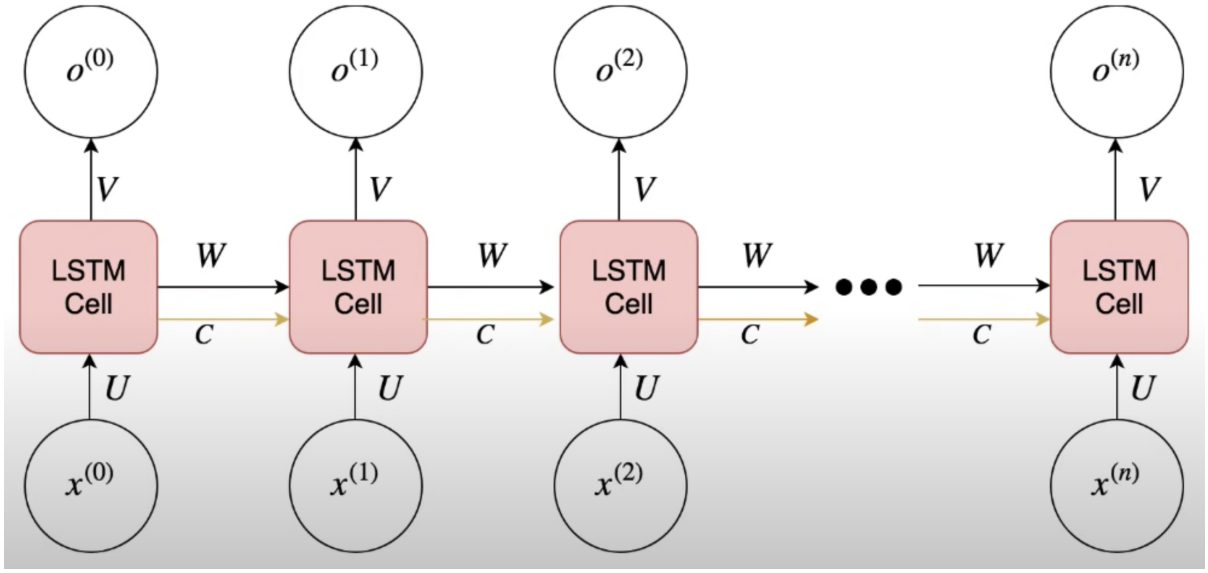


Figure 2.10: Basic Architecture of a LSTM Network. (Source: [7])

$$i_t = \sigma\left(W_f\left[h_{t-1}, x_t\right] + b_i\right) \tag{2.2}$$

$$\tilde{C} = \tanh\left(W_f\left[h_{t-1}, x_t\right] + b_c\right) \tag{2.3}$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{2.4}$$

$$o_t = \sigma\left(W_f\left[h_{t-1}, x_t\right] + b_o\right) \tag{2.5}$$

$$h_t = o_t * \tanh\left(C_t\right) \tag{2.6}$$

The third and final gate of the LSTM cell is the Output gate. For the time step, it generates the subsequent hidden state (h_t). The information from the preceding inputs is propagated by (h_t). To generate this, a sigmoid function is applied on the prior hidden state (h_{t-1}) and the current input (x_t) (Eqn. 2.5). The information that the concealed state would transmit to the following time step is then determined by passing the new cell state value to a tanh function and multiplying it by the sigmoid output. This represent the hidden state (h_t) at the present time step t. Eqns. 2.1–2.6 are taken from Olah [30]. Figure 2.11 shows an internal structure of LSTM.

Gated Recurrent Unit

In 2016, Cho et al. [6] introduced a dramatic update to LSTM. They combined the forget gate and the input gate into an update gate. This update gate in this model can make the decision of which information is to preserve and which one to forget in one single step. They also merged the cell state and hidden state into one state. All of these changes presented a simpler

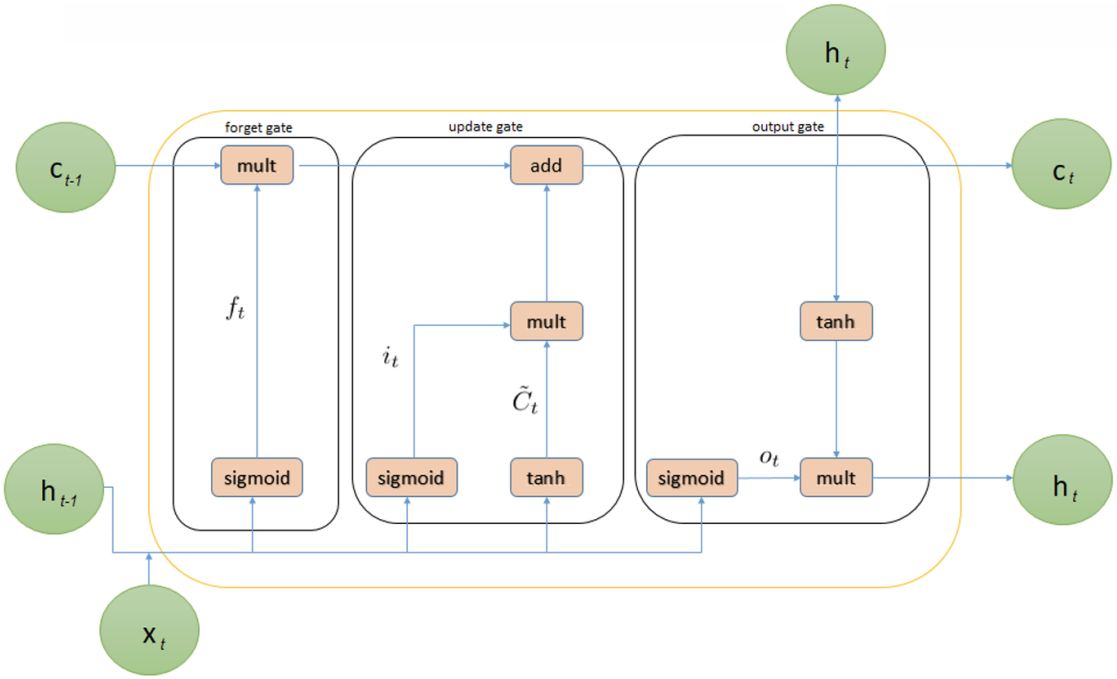


Figure 2.11: The internal structure of Long short-term memory. (Source: [14])

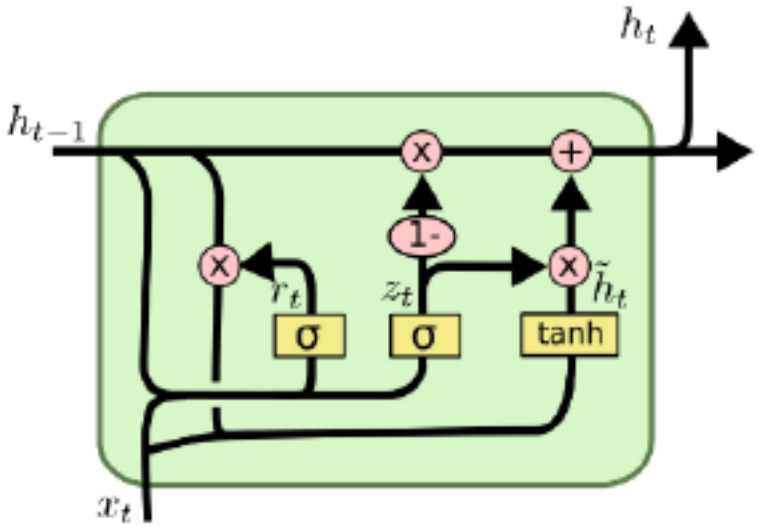


Figure 2.12: Gated recurrent unit. (Source: [30])

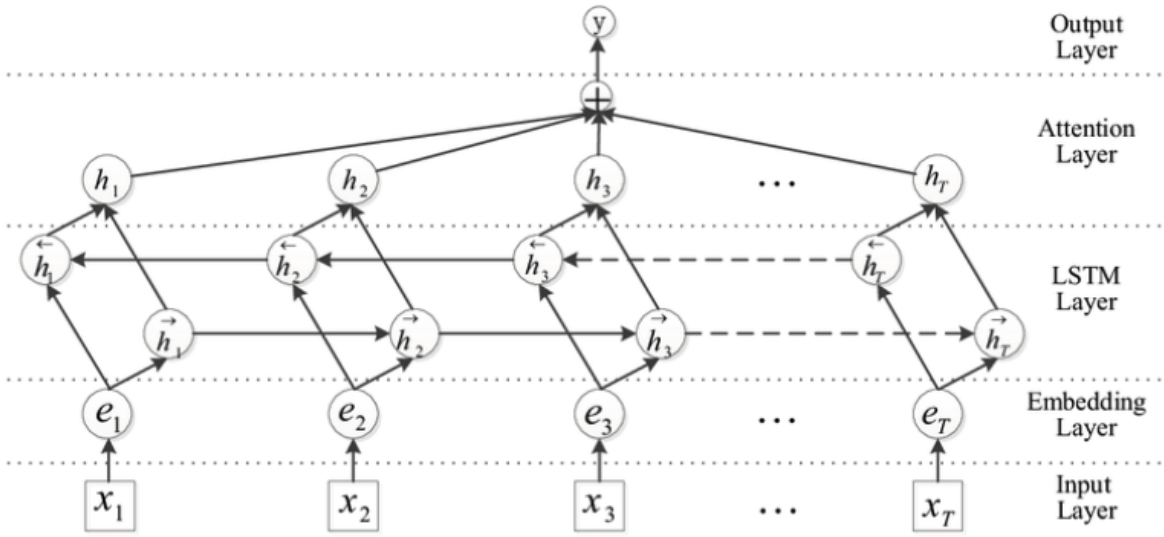


Figure 2.13: Model Architecture of Attention-based Long Short-term Memory Networks (Source: [47])

architecture that is easy to follow and saves some computation cost without losing accuracy. These changes have helped GRU gain popularity in recent days. The following equations were updated to calculate the states in GRU. Figure 2.12 illustrates the architecture of the Gated Recurrent Unit.

$$z_t = \sigma\left(W_z\left[h_{t-1}, x_t\right]\right)$$

$$r_t = \sigma\left(W_r\left[h_{t-1}, x_t\right]\right)$$

$$\tilde{h}_t = \tanh\left(W\left[r_t * h_{t-1}, x_t\right]\right)$$

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

2.3.2 Attention-based Bidirectional LSTM

In order to complete relation categorization tasks at the word level, Zhou et al. [47] suggested an attention-based bidirectional long short-term memory network (biLSTM) in 2016. In contrast to current state-of-the-art systems that derive features from lexical resources, the proposed model captures key aspects utilising attention-based biLSTM. The input layer, embedding layer, LSTM layer, attention layer, and output layer are the five layers that make up the model. Figure 2.13 illustrates the architecture of the biLSTM model.

Original sentence is fed to the model as an input. Every word in the sentence is projected to a low dimension vector by the embedding layer. Strong features from embedded word vectors

are captured by the subsequent LSTM layer. Then, the attention layer creates a weight matrix connecting the LSTM layer's word-level features to a feature vector at the sentence level. The sentence level feature vector is then sent into a softmax-based output layer, which generates probability distributions for each relation class.

2.3.3 BERT

In the field of natural language processing (NLP), the BERT (Bidirectional Encoder Representations from Transformers) architecture has been a significant development in recent years. BERT is a deep neural network architecture that utilizes transformer-based models to capture the contextual relationships between words in a sentence.

One of the key features of the BERT architecture is its bidirectional approach. This technique considers both preceding and subsequent words in a sentence to better understand the text's meaning and represent it effectively. By processing the text in both directions, the architecture can capture long-range dependencies between words and sentences, leading to more accurate representations of the text's meaning.

The BERT architecture is composed of a multi-layer bidirectional transformer encoder that processes the input text. This transformer encoder is trained using an unsupervised learning technique known as pre-training. During pre-training, the model is exposed to a large corpus of text, such as Wikipedia or BookCorpus, and learns to understand the relationships between words and sentences in the corpus.

The pre-training stage consists of two tasks: masked language modeling and next sentence prediction. In masked language modeling, the model is trained to predict the missing words in a sentence with a random mask applied to a percentage of the input tokens. In next sentence prediction, the model is trained to predict whether two sentences follow each other in the text.

After pre-training, the BERT architecture can be fine-tuned on a specific NLP task, such as sentiment analysis or named entity recognition, using supervised learning. In the fine-tuning stage, the model is trained on a labeled dataset specific to the task, allowing it to adapt its learned representations to the task at hand.

The BERT architecture has been used extensively in various NLP tasks, such as question-answering, sentiment analysis, and text classification, and has shown superior performance compared to traditional NLP models. Its ability to capture contextual relationships between words in a sentence has made it a valuable tool for various industries, including healthcare, finance, and e-commerce.

Overall, the BERT architecture's ability to capture contextual relationships between words and sentences has transformed the field of NLP and continues to advance the development of

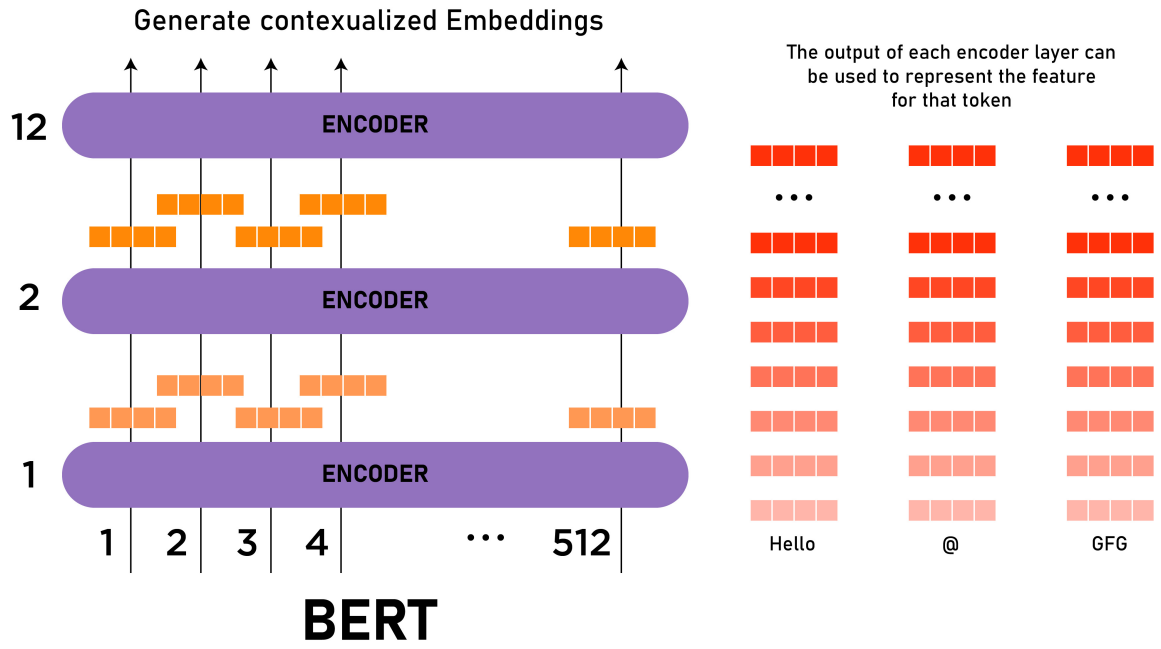


Figure 2.14: Model Architecture of BERT. (Source: [33])

more advanced and specialized models. Its effectiveness in various NLP tasks has made it a valuable tool for improving the accuracy of text-based applications and services.

PubMed-BERT

Pubmed-BERT is a specialized variant of the BERT architecture that has been trained on a large corpus of biomedical text. This architecture has been specifically designed to better understand the complex language used in biomedical literature, which often contains technical jargon and complex terminology.

The Pubmed-BERT architecture is similar to the original BERT architecture, with some notable differences. Firstly, it has been pre-trained on a large corpus of biomedical text, including abstracts and full-text articles from PubMed Central, which contains over 30 million biomedical articles. This training data includes a diverse range of topics such as genetics, epidemiology, and pharmacology, ensuring that the model can capture the nuances of biomedical language.

Secondly, the Pubmed-BERT architecture has been fine-tuned on various biomedical tasks, including named entity recognition, relation extraction, and question answering. This fine-tuning process has allowed the model to adapt its learned representations to specific biomedical applications, improving its performance on these tasks.

The Pubmed-BERT architecture has shown superior performance compared to other NLP

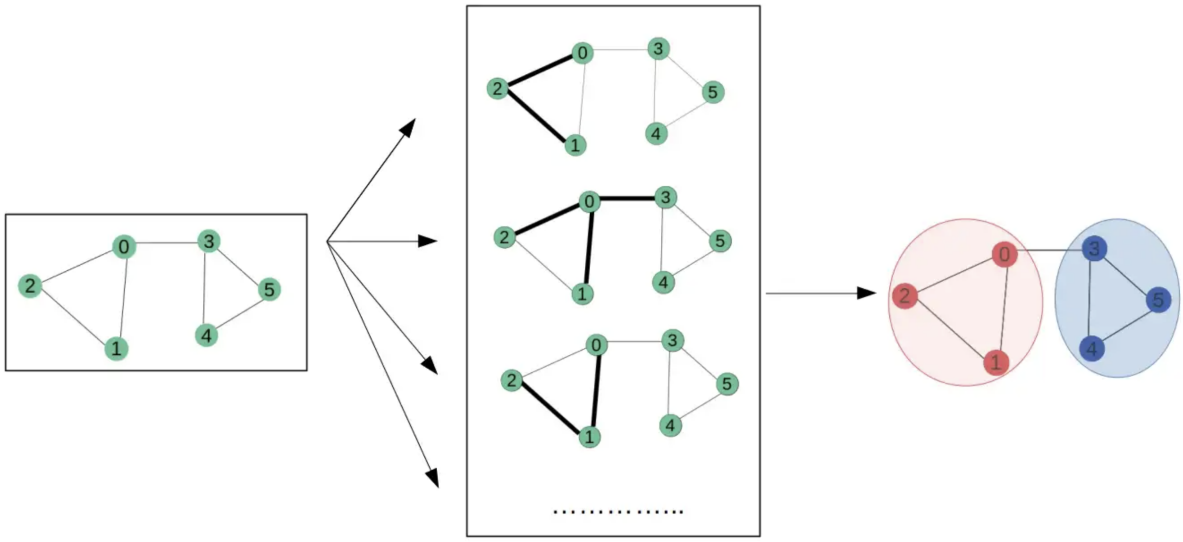


Figure 2.15: Illustration of Graph Convolutional Networks. (Source: [23])

models in various biomedical applications. For instance, it has been used to identify drug interactions, predict disease risk factors, and classify medical images. Its ability to accurately represent biomedical text has made it a valuable tool for researchers and healthcare professionals, enabling faster and more accurate analysis of biomedical literature.

Furthermore, the Pubmed-BERT architecture's effectiveness has led to the development of other specialized variants, such as ClinicalBERT and BioBERT, which have been trained on clinical and biomedical text, respectively. These models have further advanced the field of NLP in the biomedical domain and have opened up new opportunities for improving healthcare and medical research.

In summary, the Pubmed-BERT architecture is a specialized variant of the BERT architecture that has been trained on a large corpus of biomedical text and fine-tuned on various biomedical tasks. Its effectiveness in accurately representing biomedical language has made it a valuable tool for researchers and healthcare professionals, and its success has led to the development of other specialized NLP models in the biomedical domain.

2.3.4 Graph Convolutional Network

In the past ten years, neural networks have experienced tremendous progress. Although many data sets in the real world contain underlying graph topologies that are not Euclidean, early Neural Network variations could only be built using regular or Euclidean data. New developments in Graph Neural Networks have been made possible by the non-regularity of data structures. Graph Convolutional Networks (GCN) are one of the many variations of Graph

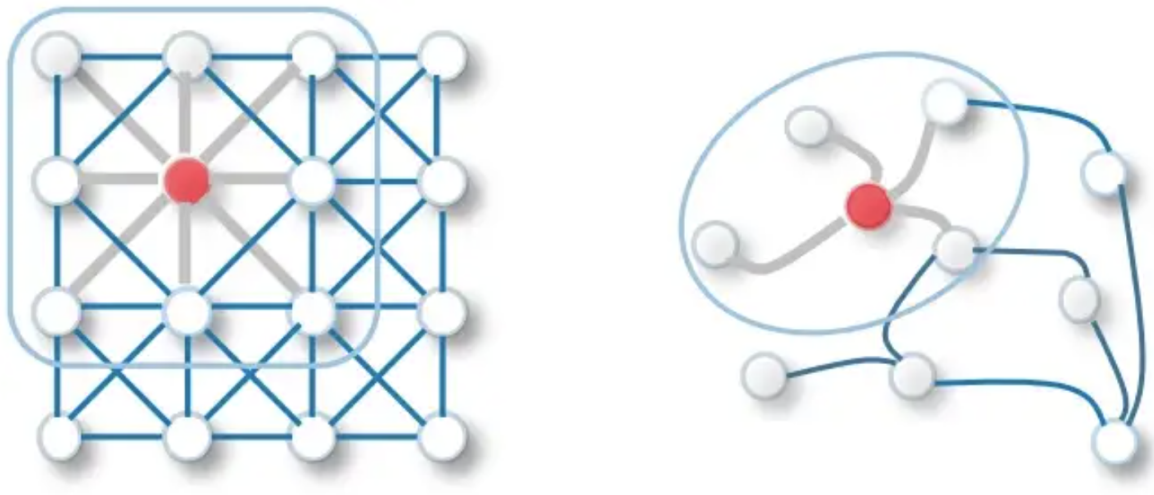


Figure 2.16: Illustration of 2D Convolutional Neural Networks (Left) and Graph Convolutional Networks (Right). (Source: [43])

Neural Networks that have been created over the last several years. GCNs are regarded as one of the fundamental Graph Neural Networks subtypes. Figure 2.15 represents a bare illustration of GCN.

Similar actions are carried out by GCNs, however the model learns the characteristics by looking at the nearby nodes. The primary distinction between CNNs and GCNs is that the former were developed specifically to function on regular (Euclidean) organised data, whilst the latter are generalised CNNs with varying numbers of connections and unordered nodes (irregular or non-Euclidean structured data). Figure 2.16 shows how selecting adjacent nodes differs between regular CNN and a graph structure.

A GCN can be described by the following equation:

$$H^{[i+1]} = \omega\left(W^{[i]}H^{[i]}A^{*}\right)$$

Here, A represents the adjacency matrix which carries the information about the adjacent nodes. Here $H^{[i+1]}$ is the feature representation at the $i+1$ layer. ω is the activation function applied on the weight W, the hidden representation at the current stage $h^{[i]}$ and the bias b.

2.4 Text Representations

In the progression of deep learning models, computers have gained the ability to learn inherent structures of data and make decisions based on data analysis. But, machine learning algorithms are incapable of processing raw string or text data. They only understand numbers. Converting

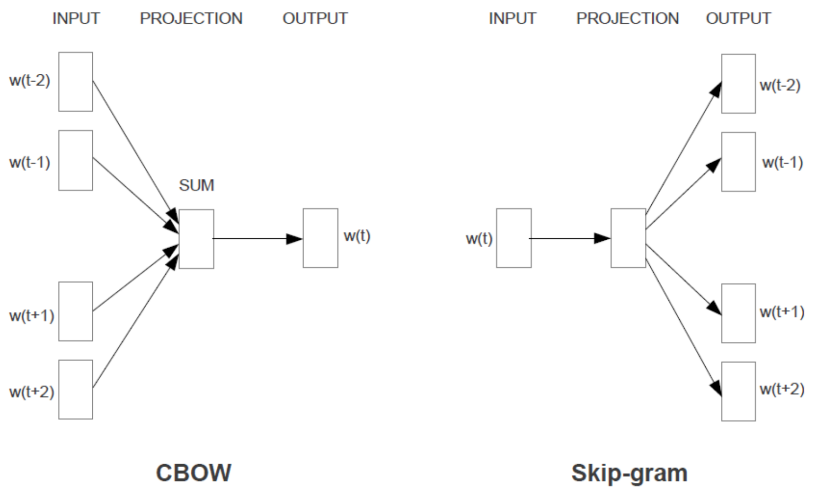


Figure 2.17: Architecture of Word2Vec (Source: [26]).

the text data into numerical vectors allows the machine learning algorithm to extract meaningful patterns and relationships between the words and their contexts. The process of mapping each word in the lexicon to a mathematical representation—more particularly, a particular vector representation—is known as word embedding. One hot encoding is a simple method of turning a set of words into vectors. In one hot encoding a binary vector is created of the same length of the vocabulary. Each vector contains 0 in all places except for the word its representing and contains 1 in the position of the exact word. For example, a vocabulary containing only two words: ['Text','Representations'], the one-hot representation of these words will be 'Text': [1,0], 'Representations': [0,1]. But this technique is unmanageable for a large dataset, as the size of the vectors grows according to the size of the vocabulary. More importantly, the semantic meaning of the sentence is lost in this representation. To overcome these challenges several natural language models are used to represent text. Word2Vec is one of them.

2.4.1 Word2Vec

Word embedding is a technique to represent words with vectors of real numbers. Word2Vec [25, 26] is one of the most popularly used word embeddings in the natural language processing realm. This embedding represents words as vectors that preserve the relationships and meanings of the original words in the vector spaces. Word2Vec combines two model architectures to construct the embedding:

1. **Skip-Gram (SG):** Model is trained to predict the context words from the target words.

2. **Continuous Bag-of-Words (CBOW):** Model is trained to predict the target words from

the context words. This output varies based on the context window size.

The goal of both of these models is to transfer the words' vocabulary-size-dependent one-hot encodings to fixed-size, lower-dimensional vector representations while maintaining semantic information. Each element of this vector is a float value to increase precision because floating point values are better at managing precision. When compared to other models at the time, this shallow model captures more semantic and syntactic information thanks to training on a corpus of 1.6 billion words. Additionally, they assert that their model is capable of capturing the numerous degrees of similarity between words in addition to just bringing them closer together. For instance, nouns can contain words with similar suffixes, and if we look for words with similar endings in the original vector space, we can locate such words [35].

The vectors produced by the Word2Vec model can keep the original meaning and the relationships among the words. It is found that Skip-Gram works well with small datasets, and can better represent less frequent words. However, CBOW is found to train faster than Skip-Gram, and can better represent more frequent words.

2.4.2 BioWordVec

FastText [5] is a variant of Word2Vec. It learns at the subword level rather than the word level. FastText was trained for BioNLP (Biomedical Natural Language Processing) tasks [45] obtaining the BioWordVec word embeddings. The word embeddings produced correspond to the terms present in biomedical texts. The data used came from two sources to train the model: the biomedical literature contained in PubMed, a library of more than 25 million journal article abstracts (at the time that BioWordVec was built), and domain knowledge supplied by Medical Subject Heading (MeSH) terms. A MeSH word graph was initially created for this assignment using the MeSH RDF data. A random sampling technique was used to construct a number of MeSH phrase sequences. After that, this data was used to train the FastText model, teaching it text sequences and MeSH term sequences. This representation holds significant values in the biomedical word domain.

2.4.3 BERT Embedding

BERT embedding is a language modeling technique that utilizes deep neural networks to generate contextualized word embeddings. These embeddings are a distributed representation of words that capture their meaning and usage in a given context.

The BERT embedding process involves pre-training a large neural network on a massive corpus of text data. During this pre-training phase, the model learns to predict the next word

in a sentence given the preceding context. This process allows the model to learn contextual information about words and their usage in natural language.

Once the model is trained, it can be fine-tuned for specific natural language processing (NLP) tasks such as text classification, sentiment analysis, and named entity recognition. During the fine-tuning phase, the model's parameters are adjusted to suit the task at hand, and the model's contextualized word embeddings can be used as input features for downstream NLP models.

The BERT embedding technique has several advantages over traditional word embedding methods such as Word2Vec and GloVe. Firstly, BERT embeddings capture more nuanced contextual information about words, making them more suitable for complex NLP tasks such as question answering and natural language inference.

Secondly, BERT embeddings are pre-trained on a massive corpus of text data, making them more robust and generalizable than other word embedding methods. This means that the embeddings can be used for a wide range of NLP tasks without the need for task-specific training data.

Finally, BERT embeddings can be fine-tuned on small amounts of task-specific data, making them useful for scenarios where labeled data is scarce or expensive to obtain. In our case we used a pretrained BERT model which is trained on Biomedical corpus PubMed.

BERT embedding is a powerful language modeling technique that utilizes deep neural networks to generate contextualized word embeddings. These embeddings capture nuanced contextual information about words and can be fine-tuned for specific NLP tasks, making them a valuable tool for natural language processing.

2.5 Related Work in MeSH Indexing

Automatic MeSH indexing is a challenging task due to the proliferation of articles in MEDLINE and the rising number of MeSH terms every year. The first software that generates MeSH indexing recommendations automatically is the Medical Text Indexer (MTI) [2] created by the National Library of Medicine (NLM) of the United States. If a MEDLINE article's title and abstract are provided, MTI will offer a ranked list of MeSH terms. MTI's original system was created in 2002 and has undergone constant improvement ever since. Figure 2.18 depicts the flow of the present MTI processing. MetaMap [1] and PubMed Related Citations (PRC) [20] are the two fundamental parts of MTI. MetaMap uses the Unified Medical Language System (UMLS) to analyse and annotate texts. UMLS words are mapped to MeSH terms using Restrict-To-Mesh [8]. The PRC method with k-nearest neighbours (kNN) searches for MeSH terms based on document similarity. MeSH indexers can utilise MTI suggestions for the texts

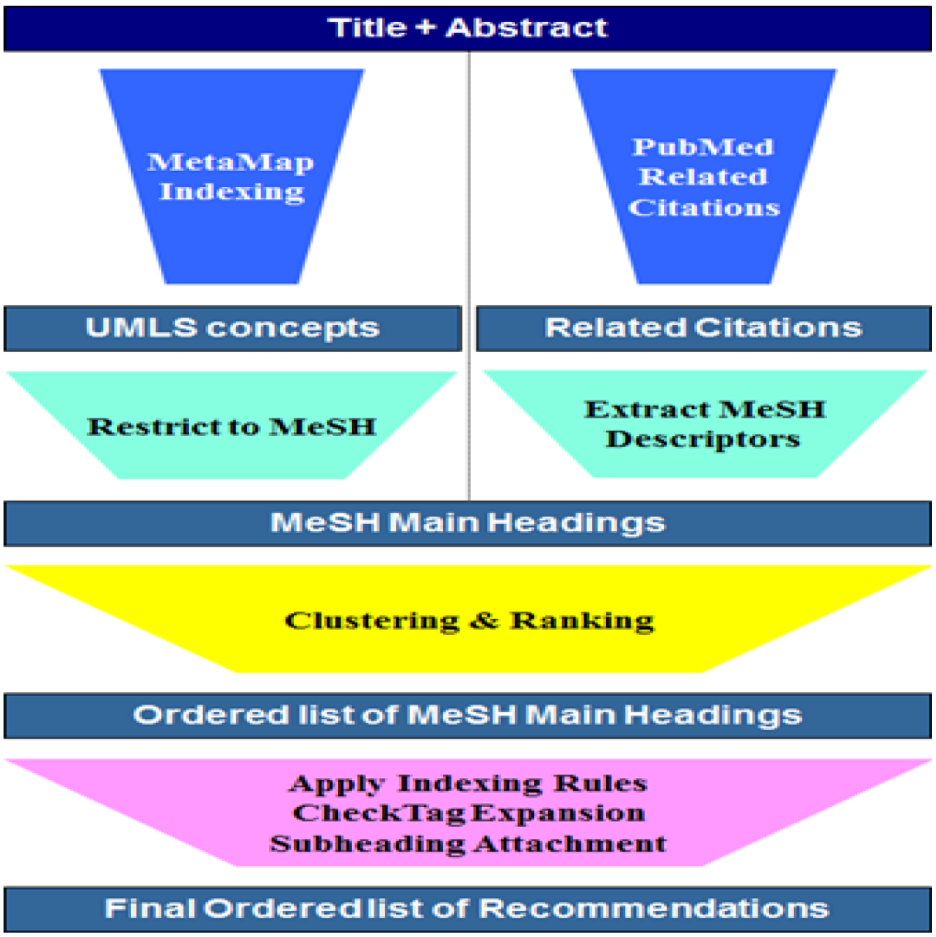


Figure 2.18: Current MTI Processing Flow (Source: `https://ii.nlm.nih.gov/MTI/`)

they are annotating as it is a key tool in MeSH indexing [40].

Since 2013, the European Union-funded initiative BioASQ has held competitions on automated MeSH indexing. Participants are expected to annotate unlabeled PubMed citations with abstracts and titles using their models. For instance, the winning system in 2013 learned two models for ranking and forecasting the amount of related labels using the MetaLabeler algorithm [36]. MeSHLabeler, which consists of MeSHRanker and MeSHNumber, took first place in 2014. A ranked list of potential MeSH phrases is returned by MeSHRanker. The number of output MeSH words is predicted by MeSHNumber [22].

In Figure 2.19, the MeSHLabeler workflow system is illustrated. In 2017, DeepMeSH was the best system. It uses a dense semantic representation for texts by converting document to vectors. This step helps to include deep semantic information in the input before sending it to MeSHLabeler . DeepMeSH additionally incorporates another classifier to determine the quantity of MeSH terms returned. A bi-direction recurrent gated unit (BiGRU) is used by AttentionMeSH [17], which was also proposed in 2017. This model collects contextual data

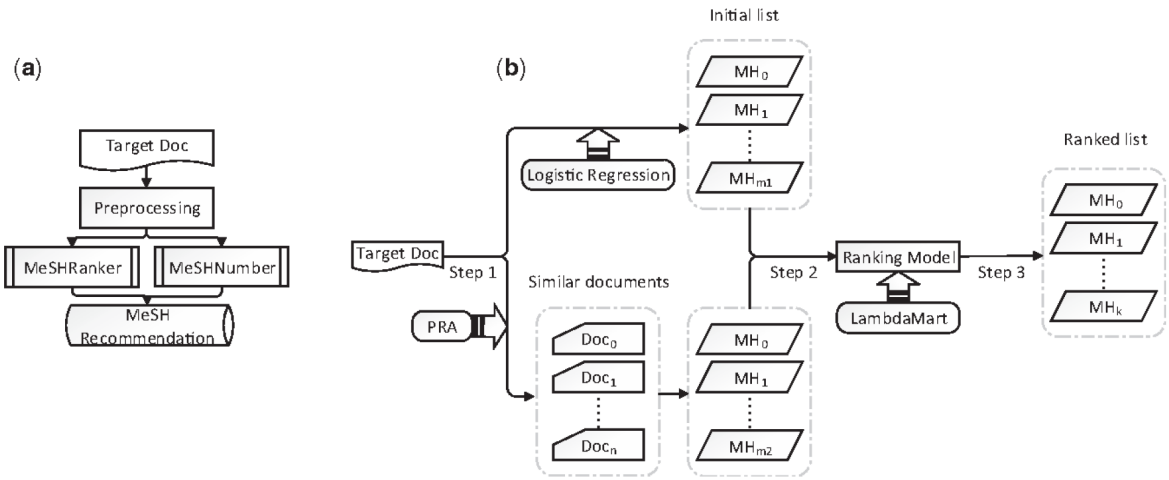


Figure 2.19: A Work Flow of MeSHRanker (a) and MeSHLabeler (b) (Source: [22])

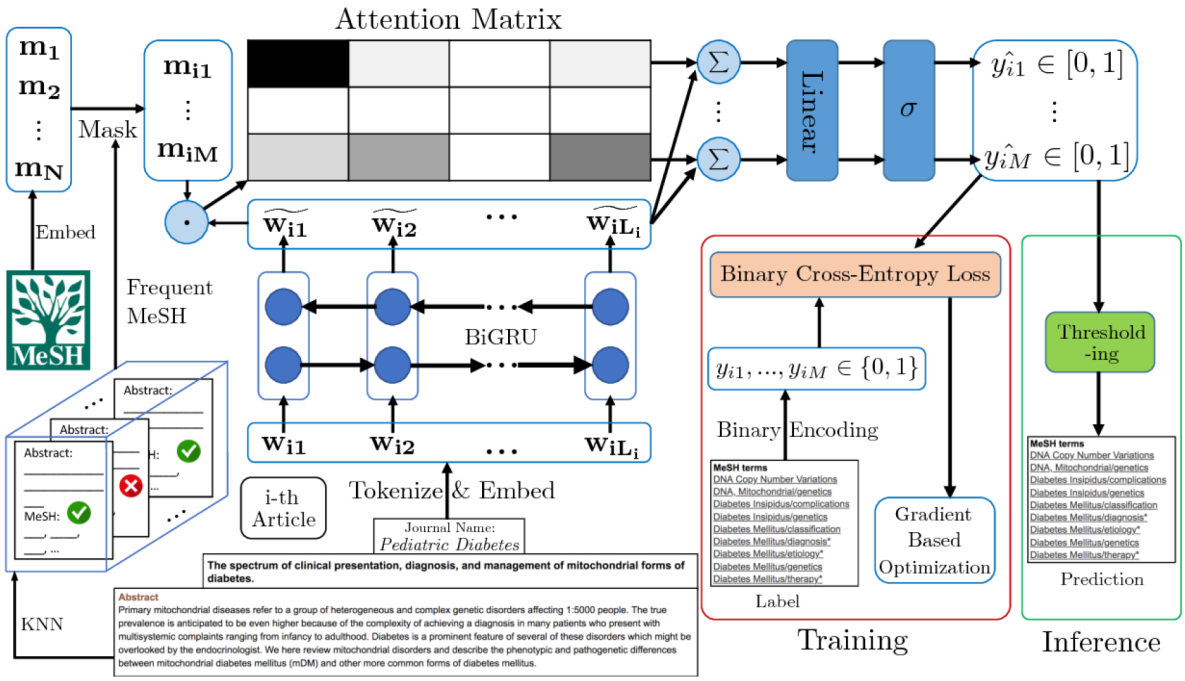


Figure 2.20: Model Architecture of AttentionMeSH (Source: [17])

and attention processes to choose MeSH phrases from the candidate list. Figure 2.20 provides an illustration of the model architecture.

In 2015 the 29 most popular MeSH phrases were classified using a CNN [34] trained on a small sample dataset with 9,000 citations. On another approach, DeepCNN was applied to 1,115,090 publications with titles and abstracts [9]. In addition to deep learning techniques, various machine learning methods have also been researched to potentially solve MeSH indexing tasks. Naive Bayes (NB), support vector machines (SVM), linear regression, and AdaBoost

are a few examples [15] [16].

Chapter 3

Methodology

In this chapter, we will provide an overview of the project setup and the deep learning model architecture selected as the most optimized and best performing one. We will explain how we arrived at this decision by comparing its performance against several other model architectures that we experimented with. Additionally, we will discuss the challenges encountered during the model's implementation, including the software engineering research conducted to optimize memory usage and increase speed. Finally, we will elaborate on the creation of the dataset, which is a significant contribution of this research.

3.1 Problem Statement

In this thesis, we intend to solve the automatic MeSH indexing for biomedical documents. We started our experiment on the state-of-the-art KenMeSH [41] model with an intention to find an upgraded model that will be more dynamic, easy to customize and better in performance. We ended up with a fully new modified model architecture.

MeSH indexing is a multi-label text classification problem. We intend to label each biomedical document with a set of associated MeSH terms. For a set of biomedical documents χ and a given set of MeSH terms ψ the learning function can be defined as $f{:}\chi \rightarrow 2^{\psi}$. The training set would be $\Theta = (x_i, Y_i), i = 1, ..., N$, where N is the total number of documents in the training set χ. For n number of words in such document, x_i is an $n \times e$ dimensional vector, where e is the dimension of the word embedding. $Y_i \subseteq \psi$ is the set of labels associated with instance x_i. The objective of the learning function is to predict a set of labels Y_k for an unseen instance x_k.

There are a few challenges in implementing this framework. First of all, the frequency of MeSH terms occurring for a document is variable. For example, a document can be associated with 30 MeSH terms where as its also possible to have 5 MeSH terms in another document. Secondly, the total number of MeSH terms is large, containing 28,863 MeSH terms. And these

MeSH terms are arranged in a hierarchical structure. On the other hand the frequency of a MeSH term appearing in multiple documents is also a variable. The most frequent MeSH term "Human" appears in 8,152,852 documents where as "Pandanaceae", a rare term, appears in only 31 documents.

Medical subject headings (MeSH) have been utilized to systematically and reliably index biomedical material. MeSH terms are MEDLINE domain-specific medical terms used to label medical documents. Annual updates are made to the hierarchical structure of MeSH words. MeSH keywords distinguish MEDLINE2, which is an excellent resource for indexers and searchers. MeSH keywords are used by indexers at the National Library of Medicine (NLM) to categorize documents based on the information found in journal articles stored in the MEDLINE database. MeSH phrases are used by academics and searchers to speed up subject searches in databases like MEDLINE, PubMed3, and others.

Currently, a significant number of human annotators analyse full text articles and assign appropriate MeSH words to each article as part of the MeSH term indexing process. Human annotation is expensive and time-consuming. According to research, the average cost of annotating one document is $9.40 [27], which is a significant expense when indexing several pages. In the meanwhile, many documents are added daily to the MEDLINE and PubMed databases (approximately 2,000 on average, but as many as 4,000, on a daily basis) 4. It is difficult to annotate all newly released papers in a reasonable amount of time. Therefore, it is extremely desirable to have a system that can index many biological papers.

3.2 Data Set

The dataset used in this project is MeSHup, the corpus provided by Wang et al. [42]. This corpus is derived from the November 2021 BioC-PMC dataset. It contains 1,342,667 full-text articles written in English. In addition to the full text, each document also has metadata that includes the MeSH terms indexed by human annotators. We extracted the PMID (PubMed ID), abstract, title, MeSH terms for each documents. We then arranged them in a JSON format for human readability, investigation and easy loading. An example of a sample document is shown below:

- **PMID**: 19333414
- **Title**: Investigation on the protective effect of α-mannan against the DNA damage induced by aflatoxin B_1 in mouse hepatocytes
- **Abstract**: Aflatoxin B(1) is a contaminant of agricultural and dairy products that can be related to mutagenic and carcinogenic effects. In this report we explore the capacity of

α-mannan (Man) to reduce the DNA damage induced by AFB (1) in mouse hepatocytes. For this purpose we applied the comet assay to groups of animals which were first administered Man (100, 400 and 700 mg/kg, respectively) and 20 min later 1.0 mg/kg of AFB (1). Liver cells were obtained at 4, 10, and 16 h after the chemical administration and examined. The results showed no protection of the damage induced by AFB(1) with the low dose of the polysaccharide, but they did reveal antigenotoxic activity exerted by the two high doses. In addition, we induced a co-crystallization between both compounds, determined their fusion points and analyzed the molecules by UV spectroscopy. The obtained data suggested the formation of a supramolecular complex between AFB(1) and Man.

- **MeSH Terms**: Administration, Oral | Aflatoxins | Animals | DNA Damage | Hepatocytes | Mannans | Mice | Mutagens

3.2.1 Data Processing

To train our model, we need to process the data set in several ways and generate a few supporting data-sets. Below are descriptions of all of the data used in the training and how we arrange them for our experiment.

Mesh Name-Id Mapping

In our MeshUp data set, each document is associated with its human-annotated MeSH terms. But to use those MeSH terms in training, we convert all of the MeSH terms to their corresponding id. These ids are unique to each MeSH term and are defined by PubMed. This helps us build a unique class of MeSH ids to be used for prediction mapping. We can also avoid many problems in string manipulation like capitalization and variable spacing, by using MeSH ids. This data is made available from UMLS [4] and we arrange all them in a dictionary where the MeSH terms are the keys and the corresponding MeSH id is the value. This helps us convert the MeSH terms for each document into their MeSH ids and finally build a class of labels with all of the MeSH ids.

Parent-Child Mapping

MeSH terms are hierarchically arranged medical subject headings. We arrange them in a JSON file in a such a way that each line denotes an edge in the tree and the two MeSH terms are the nodes. This will help us build the tree structure which we will use to build our knowledge-enhanced graph. We are calling it knowledge-enhanced because we will include the word

embedding for each MeSH term.

MeSH Descriptions

Our research builds upon the concept of knowledge-enhanced graph construction introduced in KenMeSH[41]. However, while KenMeSH relied on BioWordVec [45] for MeSH term embeddings, we sought to explore the use of rich BERT embeddings generated for each MeSH term. To further enhance these embeddings, we decided to generate them not only from MeSH terms, but also from their corresponding descriptions available on UMLS. Unfortunately, we faced a significant challenge as there are no readily available datasets or APIs containing the MeSH term and descriptions pair.

We use a data set called UMLS_data_2022AA in UMLS [4] that has all the attribute information, like keywords, entry, parent, and description, about each MeSH term. We extract the descriptions for each MeSH term and compile them into a Python dictionary format. This data file is a crucial contribution of our research, allowing us to generate more comprehensive and informative embeddings for our knowledge-enhanced graph.

Journal Information

Journal information is a collection of MeSH term data that corresponds with each biomedical journal. Each journal entry in the collection contains the number of times each MeSH term appeared in the articles published in that journal and the total count of the individual MeSH term counts for that journal. Then for each journal, we iterate over each MeSH term count and make a list of MeSH terms for that journal that have a normalized count over a threshold of 0.5. We will use this information along with neighbour MeSH information to build a MeSH mask for precise attention. We will discuss more about this in Section 3.4.2.

Neighbour MeSH

To enhance our MeSH mask, we calculated the K-nearest neighbours for each MeSH term from our dataset. To create this, we calculated a tf-idf vector for each MeSH term and generated the neighbour dictionary. We will discuss more about this in Section 3.4.2.

Final Data Set

We created a final data set, which has the structure of our original data set but it also has a new attribute for each document, which is the MeSH mask. The MeSH mask is a vector of 28,415 dimensions. Here 28,415 is the total number of MeSH terms. The vector has a value of 1 in the index of each MeSH term that is present and 0 otherwise.

3.3 Overview of KenMeSH

Since KenMeSH [41] is the basis of BERTKenMeSH we first provide a brief overview of this deep learning model, focussing on those aspects which we modify to build BERTKenMeSH.

KenMeSH is a deep learning model designed and implemented in our Cognitive Engineering Laboratory at The University of Western Ontario. At the time of publication, it was the state-of-the-art automatic MeSH term indexer. Figure 3.1 shows the KenMeSH architecture.

KenMeSH is composed of three main components. The first component is a two-channel document representation module that takes as input the title and abstract of an article. It uses BioWordVec [45] as the word embedding layer, sends the word embeddings to two BiLSTM layers, and the abstract channel finishes by feeding the BiLSTM hidden values to a dilated CNN. The last hidden value from the abstract channel BiLSTM and the final value from the dilated CNN are then given to the attention layer. The second component is a masked attention layer. It calculates the label-specific attention vectors used for predictions in the output layer. The MeSH masks are generated using journal information and k-nearest neighbour information. The third component is a 2-layer Graph Convolution Network (GCN) that creates label vectors. It uses information about the MeSH terms converted to embeddings using BioWordVec to form the nodes of the graph and the hierarchical organization of MeSH to provide the edges in the graph. The masked GCN output is used in the attention layer and the GCN output is used in the output layer.

Three modifications of the original KenMeSH will be done. First, BioWordVec word embeddings will be replaced with PubMedBERT word embeddings everywhere that BioWordVec occurs in KenMeSH. Second, the MeSH embeddings that form the nodes in the GCN graph will be generated from full descriptions of the MeSH terms. Third, the dilated CNN will be modified.

In Chapter 4 these modifications are done in a sequence of changes. Phase I incorporates the full descriptions of the MeSH terms and the use of PubMedBERT word embeddings to generate the MeSH term embeddings for the GCN graph. The rest of the KenMeSH architecture remains unchanged. The results reported for Phase I will be called KenMeSH with BERT Embedding. Phase II sees the replacement of the BioWordVec word embedding layer and the BiLSTM layer in the two-channel document representation module with a PubMedBERT word embedding layer. The Phase II results are called BERTKenMeSH (3 layer CNN). The dilated CNN is modified in Phase III. The Phase II results are called BERTKenMeSH (7 layer CNN).

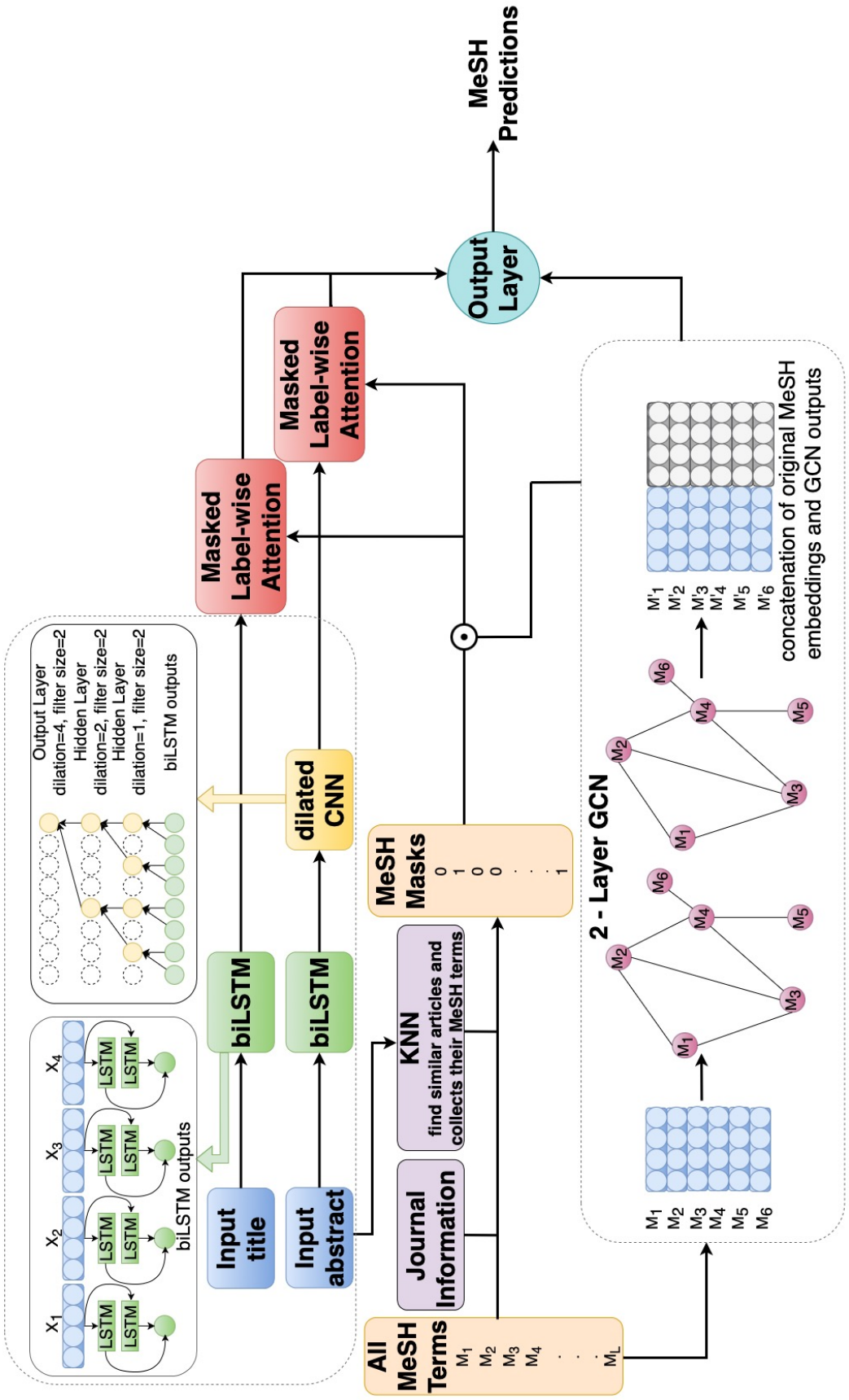


Figure 3.1: KenMeSH Architecture (Source: [41])

3.4 Model overview

A complete architecture of our model is illustrated in Figure 3.2. Our model can be divided into three sections, they are a label feature learning module, a dynamic semantic mask attention module, and a classification module.

3.4.1 Label Features Learning Module

We utilize the two-layer Graph Convolutional Network (GCN) that was designed in KenMeSH to combine the hierarchical parent and child information among MeSH labels since the MeSH hierarchy is crucial for our purpose. To create the label feature vectors for each MeSH phrase that are used as the nodes in the GCN graph, we use the MeSH embeddings generated from their descriptions. This modification to the feature vectors that are used in the GCN graph will be described at the end of this section.

In the graph structure, each node is considered a MeSH label and edges denote their parent-child connection. At each GCN layer, the node feature is aggregated by its parent and children to form the new label feature for the next layer:

$$h^{l+1} = \sigma\left(A.h^{l}.W^{l}\right)$$

Here, h^l and h^{l+1} indicate the node presentation of the lth and $(l+1)^{th}$ layers. $\sigma(.)$ represents an non-linear activation function and A denotes an adjacency matrix of the MeSH hierarchical graph. W^l denotes a layer-specific trainable weight matrix. The label feature vectors are then added to the node as the node feature.

We modify this module to use enhanced embeddings for each MeSH term. As mentioned above, we now describe the modification that we have made to the MeSH term feature vectors. KenMeSH generates embeddings for MeSH terms by averaging the word embeddings for the MeSH term (which can be a word or a phrase). For the word embeddings, KenMeSH uses BioWordVec which is a 200-dimension embedding trained on PubMed [1] and MIMIC III [2] data.

We have made two modifications to the KenMeSH embeddings. First, to generate the embeddings we use PubMedBERT [11], a pretrained BERT model which is trained on the abstracts of all documents from PubMed and the full-text of all documents from PubMed Central. This model has achieved state-of-the-art performance in many NLP related tasks in the biomedical field. This model is also currently the top scorer in the Biomedical Language Understanding and Reasoning Benchmark.

[1] https://www.ncbi.nlm.nih.gov/pubmed/
[2] https://physionet.org/works/MIMICIIIClinicalDatabase/access.shtml

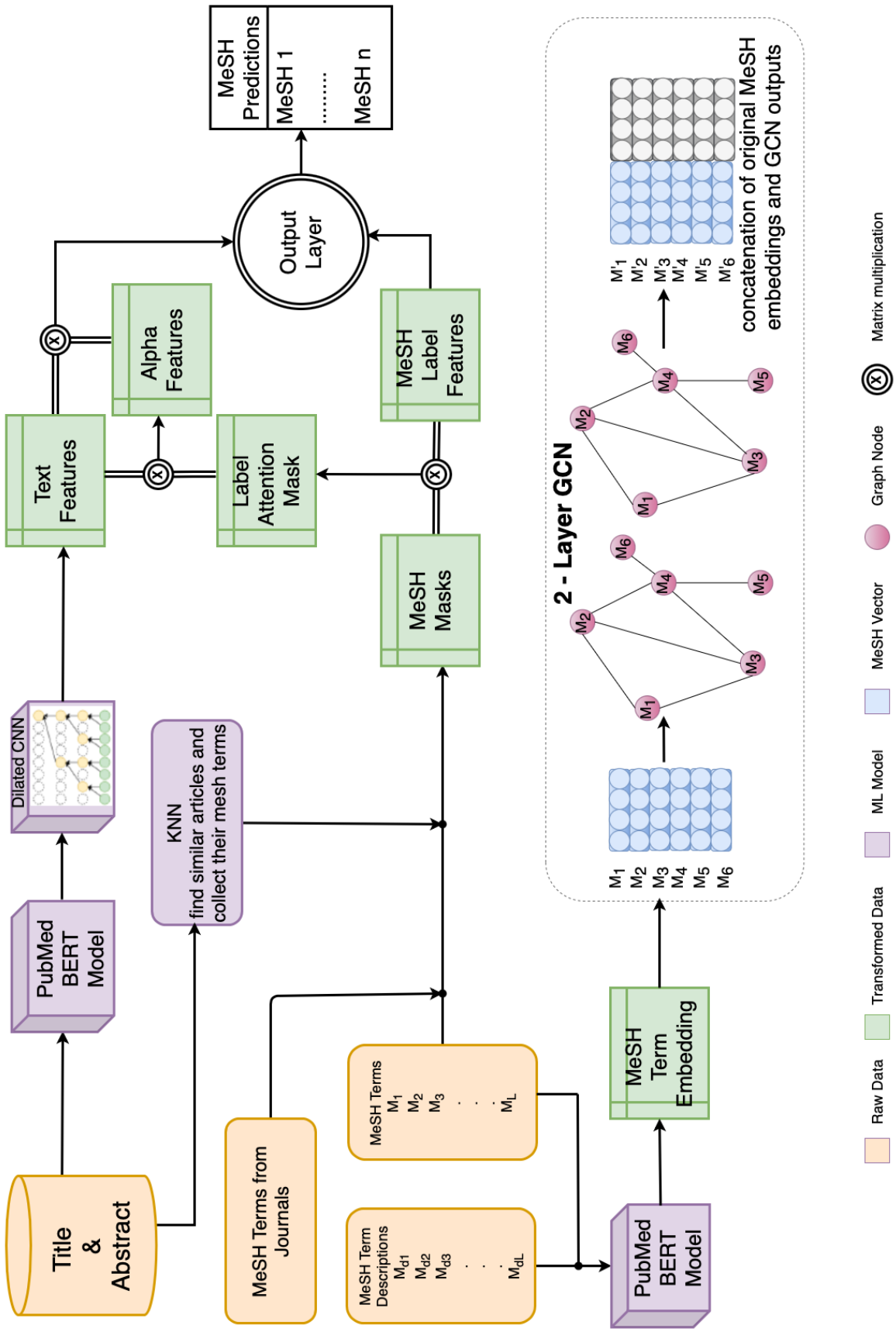


Figure 3.2: BERTKenMeSH Architecture

Second, to generate the BERT embeddings we use the MeSH descriptions data set that we have created. We use the BERT tokenizer-to-tokenizer encoder to generate the key encodings. We then generate the output from the BERT model using the key encodings. BERT generates output in various formats. We take the last hidden layer which provides the word embedding for each token in the MeSH term description. Then we take the mean of all the token embeddings to create an embedding of size 768 dimension for each MeSH term.

If we consider M is the set of all MeSH terms and M_D the set of mesh descriptions, we generate the tokenizer as follows:

$$
\begin{aligned}
key\_encoding_i &= BERT\_Tokenizer(M_{D_i}) \\
bert\_output_i &= BERT\_model(key\_encoding_i) \\
MeSH\_embedding_i &= \frac{1}{L}\sum_{j \in L} w_{ij}, i = 1, 2, ..., N
\end{aligned}
$$

Here $i = 1, 2, \ldots, N$ where N is the total number of MeSH terms, L is the number of words in its descriptor and w_i denotes the *BERT_output* for the ith MeSH term. In this case, we are taking the *last_hidden_layer* of the BERT output. For j words in the descriptor, we will get j word embeddings for any MeSH term M_i.

3.4.2 Dynamic Knowledge-enhanced Mask Attention Module

As was done in KenMeSH, we incorporate knowledge from external sources to produce a distinct mask for each article dynamically in the dynamic knowledge-enhanced mask attention module. For the following two reasons, we take into account only a portion of the entire MeSH list and use a masked label-wise attention method that computes the element-wise multiplication of a mask matrix and an attention matrix. First, MeSH terms have a broad range of occurrence rates. As a result, there are far more bad examples than good ones for each MeSH classification. Down-sampling of the negative instances is accomplished for each article by choosing a subset of MeSH labels, or a MeSH mask, which drives the classification module to focus on a smaller set of candidate labels. The second problem with the standard attention method is that it lacks pertinence because the classification module concentrates on finding pertinent information for all anticipated labels [3]. Instead, by using a masked label-wise attention, the classification module can now locate more relevant information for each label inside the MeSH mask.

Being dynamic makes sure that the module creates a distinct MeSH mask for each individual sample. We use journal information and document similarity as the two external knowledge

sources to create the MeSH masks. The term "journal information" refers to the title of the publication venue for an article, which often identifies a particular area of research. We anticipate that MeSH keywords that are pertinent to the journal's research emphasis will frequently be indexed with papers that have been published in the same publication. Using conditional probability, we create a co-occurrence matrix between journals and MeSH labels, i.e., $P(L_i|J_j)$, which denotes the probability of occurrence of label L_i when journal J_j appears.

$$P(L_i|J_i) = \frac{C_{L \cap J}}{C_j}$$

where $C_{L \cap J}$ denotes the occurrences of L_i and J_j. C_j denotes the number of occurrences of L_i in the training set. To avoid noise we introduce a threshold τ.

$$M_j = \{L_k | P(L_k|J_j) > \tau, k = 1, ..., L\}$$

where M_j denotes the MeSH mask for journal j.

Then, based on document similarity, we choose a subset of certain MeSH keywords for each article using k-nearest neighbours (KNN). Using the article's abstract, we represent each article as the IDF-weighted sum of word embeddings:

$$D_{idf} = \frac{\sum_{i=1}^{n} IDF_i \times e_i}{\sum_{i=1}^{n} IDF_i}$$

where e_i is the word embedding, and IDF_i is the inverse document frequency of the word.

Next, we utilise KNN to identify the k closest neighbours for each article and compute the cosine similarity between abstracts. After that, we collect MeSH terms from neighbours to form M_n. The MeSH mask is the union of the journal and neighbours set.

$$M = M_j \cup M_n$$

where $M \in \mathbb{R}^L$, M_j and $M_n \in [0, 1]$ is the MeSH mask. Then we apply masked label-wise attention.

$$H_{masked} = H_{label} \odot M$$

$$\alpha_{title} = \text{softmax}(H_{title} \cdot H_{masked})$$

$$\alpha_{abstract} = \text{softmax}(D_{abstract} \cdot H_{masked})$$

where $\odot$ denotes element-wise multiplication, H_{masked} denotes the masked label features and α_{title} and $\alpha_{abstract}$ measure the importance of a text fragment based on their importance. Then label-specific title and abstract representations are generated:

$$c_{title} = \alpha_{title}^{T} \cdot H_{title}$$

$$c_{abstract} = \alpha_{abstract}^{T} \cdot D_{abstract}$$

where, $c_{title} \in \mathbb{R}^{L \times 2d}$, and $c_{abstract} \in \mathbb{R}^{L \times 2d}$. The document vector is formed by summing the title and abstract representations:

$$D = c_{title} + c_{abstract}$$

3.4.3 Classification Module

We leverage PubMedBERT, a pre-trained BERT model which was trained on PubMed data obtained from PubMed Central. Since BERT can handle sentences with a maximum of 512 tokens, we concatenate the document title and abstract and generate *input_ids* and *attention_mask*. The *input_ids* and *attention_mask* are then utilized to generate output from the BERT model. We extract the last hidden layer from the BERT model.

The reason for taking the last hidden layer from BERT as opposed to other layers such as the pooling layer or the CLS token is that the last hidden layer captures the most contextualized information about the input sequence. This is because BERT uses a transformer architecture that processes the input sequence in a bidirectional manner, allowing each token to access contextual information from both its left and right context. The last hidden layer has been found to be the most informative layer for many downstream natural language processing tasks, including text classification.

Let D be the document consisting of the document title and abstract. We concatenate the document title and abstract as follows: $D = title\ abstract$. We generate *input_ids* and *attention_mask* as follows:

$$input\_ids, attention\_mask = BERT\_tokenizer(D, max\_length = 512, padding = True, truncation = True)$$

where *BERT_tokenizer* is the BERT tokenizer function. We feed the *input_ids* and the BERT tokenizer *attention_mask* to the pre-trained BERT model as follows:

$$outputs = BERT\_model(input\_ids, attention\_mask)$$
$$hidden\_state = outputs \cdot last\_hidden\_state$$

Next, we apply a 7 layer dilated CNN with a kernel size of 5 and dilation rate [1,2,4,8,4,2,1]. The dilated CNN architecture has been shown to have improved performance over standard CNNs, particularly in scenarios where the input has a large receptive field. We experimented with different dilations and layer combinations, but found out that this combination works the best in our context.

The output of the dilated CNN was then passed through a label-wise attention mask to generate alpha features, with the label attention mask being generated by multiplying the MeSH mask and label feature. The label feature itself is generated from the GCN graph in a previous step, as detailed earlier. The dilated CNN output and alpha features were multiplied to obtain focused output features, which were then multiplied by the label feature to produce a prediction matrix of size $[1, 28415]$, corresponding to the total number of MeSH terms.

For each MeSH term i:

$$\hat{y}_i = \sigma\left(D \odot H_{label}\right), i = 1, 2, ..., L$$

where $\sigma(.)$ represents the sigmoid function. The binary cross-entropy loss used to train the model is:

$$L = \sum_{i=1}^{L} \left[-y_i . \log(\hat{y}_i) - (1 - y_i).log(1 - \hat{y}_i)\right]$$

where, $y_i \in [0, 1]$ is the ground truth of label i, and $\hat{y}_i \in [0, 1]$ denotes the prediction of label i.

3.5 Environmental Experiments

One of the major challenges encountered during the training process was memory management. We have been training with a large data set of size 43.8GB. After generating mesh_mask, the data set grew to 117GB, and loading the data to the data loader consumed the full 256GB memory available in our system and still failed. We did elaborate research to find out an optimized solution.

3.5.1 Python Memory Management

Swap Memory and Memory Management

To address the issue with memory overflow, swap memory blocks of various sizes were created, and their sizes were increased until sufficient memory was available. Subsequently, research was conducted, and it was revealed that when a file is read, data is parsed into int, string, and other primitive data types, and they are organized into a DOM-like hierarchy that is ready to be read and changed later. It was observed that loading and processing a file using the json.load() function could take up to three times the size of a file to complete.

To investigate the memory usage of json.load() we did research on the use of json.load(). We noticed that it loads the full file to memory before parsing. We also noticed the usage of Python objects to store data, which takes up a significant amount of memory. Reference-type variables are an issue, because Python doesn't delete the object/variable and release the memory while there is at least one reference to the object/variable. This reference can be from a variable, but it can also be from another data structure (Example: Class). Our project uses vectors, which is a special data type that starts at the NULL pointer and grows via a capacity-doubling strategy. This alone could produce 2x bloat. This technique is used to give vectors amortized performance.

We discovered that Python's latest versions have optimized memory consumption of primitive types like strings. The process is applicable if the string can be represented in ASCII. In this case, each character might take one byte of memory. However, if there are special characters and depending on other parameters, each character can take up to 4 bytes. This means that the same length of strings can consume different sizes of space in the memory.

To explore optimization options, a script was developed to load the dataset.json file, write a batch size of 32 document objects into a new file, and run using the filprofiler library. The output from the script revealed excessive swapping, indicating why the program's progress was so slow. However, there was no memory leak detected as the data was loaded and stored in lists.

Garbage Collection

Python manages memory allocation and garbage collection to optimize the process of allocating memory to a process. The allocation flow includes Python Runtime/GC, userland allocator, and the operating system. While the garbage collector releases memory when an object has no more references to it, it does not guarantee when it will occur. The suggested solution is to allocate more memory than required and leave the unused memory as a buffer for later use. Additionally, Python assumes that some of the space might be needed in the future for alloca-

tion, so it may not reclaim all memory. This is done to optimize the allocation time of memory, and the observation made from the investigation can be useful in similar software experiments. Manually collecting garbage did not show significant changes in memory usage, and there was no evidence to support why Python is not collecting garbage.

We concluded that, the only solution to deal with a large file is the use of a generator to make the data iterable rather than loading the full data at once. And ijson is a Python library which exactly does this. It returns a generator instead of the full document. So, we decided to incorporate ijson, instead of the json library.

3.5.2 Sparse Matrix

The project ceased to function during the first epoch despite possessing a memory capacity of 1.5TB (RAM and swap included). This amount of memory consumption is considered excessive. To address the issue, we planned to implement sparse matrices, and I conducted some research on this topic. However, I initially deemed implementation in our case to be difficult due to the fact that the conversion of a SciPy sparse matrix returns a class. Our *mesh_mask* is generated from a pre-training stage, and the data is saved in a JSON file. Therefore, saving a class within an object in a JSON file is not possible.

Upon encountering issues with the project, I revisited the idea of sparse matrices and researched how to implement them in our case. As a result, I developed two utility functions. The first function is used to convert the matrix into a dictionary format of sparse matrices, while the second function converts the dictionary to a matrix of predefined length of the total number of MeSH terms. However, dictionaries cannot be saved in an object within a list. Therefore, I converted the dictionary to a string format and saved it in our JSON file under the documents object, which successfully resolved the issue.

This approach had a significant impact on the project. The data set, which was originally 110GB in size, was converted into a 42GB JSON file. The loading time of the data file was reduced from 2.30 hours to 1.39 minutes. To further enhance the training process's speed and reduce memory consumption, I only converted the *mesh_mask*s to dense matrices in batches within the dataloader. As a result, the memory consumption has been reduced to 393GB from 1.5TB. The dataloading process was completed within 15 minutes.

3.5.3 Code Refactoring

During our experimentation with the KenMeSH codebase, we encountered several issues with the code structure and hard coded values. After attempting to make modifications, we found that the code was breaking and required significant debugging efforts. The debugging was even

more time consuming and cumbersome due to the complicated data structure. To address this issue, we decided to refactor the entire codebase.

The original KenMeSH implementation was in PyTorch. We opted to refactor it using the PyTorch Lightning framework, which provides several useful features for streamlining the model training process. PyTorch Lightning is a lightweight wrapper for PyTorch that simplifies the code structure, making it more readable and easier to debug. It separates the training loop from the model, allowing for more flexibility and scalability. It provides features such as automatic batching, automatic checkpointing, and logging, which help streamline the training process.

The framework also simplifies the implementation of complex models, by providing a standardized template for defining the model architecture, data processing pipeline, and training loop. Overall, PyTorch Lightning simplifies the implementation of deep learning models, making it easier to write high-quality code and accelerate research progress.

Chapter 4

Experiments

This chapter focuses on the practical aspects of the multi-label text classification task and the subsequent analysis of the experimental outcomes. The chapter is comprised of four parts. The first part delves into the data set used in the experiment and the second part discusses the parameter configurations applied to our deep learning model. We will then discuss several evaluation techniques used to measure the performance of the models we experimented on. Finally, we will discuss the result of our experiments on various parameter settings for each model to determine the most optimal configuration. We will then provide an extensive discussion of the models' performances, using the identified optimal parameters.

4.1 Dataset

The dataset used in this project is MeSHup, the corpus provided by Wang et al. [42]. This corpus is derived from the November 2021 BioC-PMC dataset. It contains 1,342,667 full-text articles written in English and MeSH indexed by human annotators. In addition to the full text, each document also has metadata that includes the MeSH terms. We used 78% of the data to train our model, 2% for validation and the remaining 20% for testing. We trained our BERTKenMeSH model for 20 epochs when we recorded the results for this report.

We also created two experimental data sets from the original data set to save time in our experiments. One data set contains only a hundred documents from the original data set. Another data set contains ten thousand documents in the same format as it is in the original data set. We used these two data sets sequentially in our research and development. We first experimented with each model on the hundred data set. When we reached a certain stable level, we started training our model on the ten thousand data and measured the performance on various hyperparameter settings for the certain model architecture on variable epochs and batch sizes. Once we found the optimal environmental setting, we then ran the model on the full data.

4.2 Implementation Details

Our experiments and implementation is a gradual process and it can be divided into 3 phases. We started our experiments on the KenMeSH project setup. We will discuss all of the phases and their parameters below.

4.2.1 Phase I

One important part of KenMeSH is the graph label feature attention discussed in Section 3.4.1. KenMeSH uses averaged BioWordVec [45] word embeddings of the words that form the MeSH term as the embedding for each MeSH term. These MeSH term embeddings are used as the node features in the GCN graph. In the graph, every MeSH term is represented as a node and each edge represents the parent-child relationship between two MeSH terms in the MeSH hierarchy. BioWordVec provides pretrained biomedical word embeddings of dimension 200.

In our first phase, we decided to replace the BioWordVec word embeddings with richer word embeddings generated from a pretrained BERT model. The BERT model we use is PubMedBERT [11]. This BERT model is trained on the PubMed dataset which aligns with our training data and our purpose to classify biomedical documents. It is expected that the pretrained BERT will provide a much richer embedding for each MeSH term than BioWordVec.

To make the MeSH term embeddings even more precise for our work, we use our generated MeSH terms description file (see Section 3.2.1) and generated each MeSH term embedding as the average of the BERT word embeddings of the description of that MeSH term. The embedding generated from the description of each MeSH term is of size 768 dimension. The rest of the model architecture is identical to KenMeSH. But unfortunately, despite using a rich word embedding, thus a rich label feature attention, we do not see a major upgrade in the model performance. One of the main factors for the suboptimal outcome is due to the fact that the BERT embedding has a dimensionality of 768. However, in order to align with the output dimension of KenMeSH's Bi-LSTM and CNN architecture, we had to decrease the label feature vector embedding size of the GCN to 200. This is why we are missing a lot of important information from the rich BERT embeddings. This insight leads us to our second phase.

4.2.2 Phase II

We researched the experimental results thoroughly from our first phase and listed all of the probable reasons for not seeing a better performance of the model. The conclusion that we reached is that as a pre-trained deep learning model, BERT is highly effective in helping to

generate high-quality MeSH term embeddings that can capture the underlying features of the text data. Unfortunately, we did not obtain the desired results, as the Classification module architecture could not fully capture the rich features of the BERT-enhanced label feature embeddings that are applied after the dilated CNN.

In our Classification module that existed at the beginning of this phase, we used BioWordVec word embeddings for the title and abstract and a Bi-LSTM followed by a 3-layer dilated CNN, with a dilation rate of [1, 2, 3]. The Bi-LSTM generates hidden vectors that capture contextual information for each word in a sentence. The dilated CNN layers utilized in the Classification module can be thought of as filters that detect and extract specific features from the context-enhanced text data. However, the BERT-enhanced MeSH term embeddings are inherently rich and complex, and this architecture was not sophisticated enough to fully capture all the necessary features. This mismatch led to poor results.

To overcome this challenge, we decided to explore an alternative Classification module architecture that was better suited for the BERT-enhanced MeSH term embeddings. Since BERT is a pre-trained model designed for generating word embeddings, we hypothesized that using the same word embedding model would lead to better performance. In particular, we opted to use the PubMedBERT model, which is a variant of the original BERT model pre-trained on PubMed articles, specifically designed for natural language processing in the biomedical domain. By using PubMedBERT, we could leverage the same word embeddings as used in the label features, ensuring a better match between the output of the dilated CNN and the label features. Because BERT models use context to generate the word embeddings, the Bi-LSTM is no longer needed and is removed from the Classification module. Changing the core of the model architecture of KenMeSH needed major refactoring of the codebase. At the end of the modification, we established a new model architecture along with an optimized code base that is loosely coupled and dynamic to adapt to further changes. We have named our new model BERTKenMeSH which denotes that we use BERT and also employ the knowledge-enhanced MeSH label attention from KenMeSH.

Unfortunately, we did not see a good accuracy with the model after fine tuning the BERT model on our dataset. So, we had to dive deep again into debugging and researching how a neural network extracts features and what are the possible reasons for the failure. That brings us to our third and final phase. In addition, the problem is that BERT has 109M parameters and it's computationally expensive. It took us 12 days to train the model on our full data set on a 48-core CPU and 1 NVIDIA RTX A6000, 48 GB GPU. The memory consumption was 256GB of main memory in the CPU and 1.8TB Swap memory.

4.2.3 Phase III

Up until now, we have replaced the BioWordVec word embeddings with PubMedBERT word embeddings to enhance the MeSH graph node features using the MeSH term descriptions. However, in Phase I, we reduced the dimension of the BERT embeddings from 768 to 200, resulting in important information loss. To mitigate this information loss, in Phase II, we decided to replace the BioWordVec embedding layer and the Bi-LSTM module of KenMeSH with PubMedBERT. Despite the promising theoretical underpinnings of this approach, we observed poorer results than those obtained previously. Consequently, we delved into the internal mechanics of neural networks for feature extraction. We scrutinized our label feature matrices and experimented with different permutations of applying label attention to the output of our model. We were aware that we had already employed the most optimal method for extracting label-wise attention.

Then, we decided to experiment with different neural nets. We attempted to replace the dilated CNN layers with Fully-connected NN dense layers, but this did not prove to be effective. After experimenting with various settings, we realized that the root of the issue was the 3-layer dilated CNN architecture, which seemed not capable of capturing and filtering the rich BERT output features, thus removing important features from the predictions.

To overcome this challenge, we conducted extensive experiments with different CNN models, such as ResNet, VGG, and Inception, to find a suitable architecture that could accurately filter and extract the relevant features from our dataset. Ultimately, we found that the Inception architecture, with its multiple layers of convolutions, to be the most effective for capturing the complex features generated by the BERT embeddings. We started experimenting with various dilated CNN architectures varying in the combination of the number of layers, kernel size, and dilation rate. We ran our experiments on 4, 5, and 7 layer CNN models with dilation rates of [1,2,4,8,16], [1,2,4,2,1], [2,4,8,4,2], [1,2,4,8,4,2,1], and kernel sizes of 3, 5, and 7. We ran this experiment on a small data set consisting of 10,000 data samples to reduce execution time. Finally, we chose one combination based on its evaluation result and decided to include it in our BERTKenMeSH model. The results for each of these configurations are provided in Table 4.1. With this new architecture, we are able to achieve significantly improved performance in our multi-label text classification task, but the performance is still less than what we expected.

Having made this final change to the model, we started training on the full dataset. We provide a short summary of the configuration below, but more details are given in Section 4.4.

We employed dropout with a rate of 0.05 immediately after the embedding layer and batch normalization to prevent over-fitting. We employed early halting methods and the Keras AdamW optimizer. The initial learning rate is 2e-5 and we employed a linear scheduler with a warmup to adjust the learning rate during training. For our final training, we decided to

freeze BERT because PubMedBERT is already trained on PubMed data. We took this decision to save some training time at each epoch. The dilated CNN layer is larger now, and we ran into memory issues again despite having 2TB of swap memory and 256GB of main memory available on our machine. We reduced the batch size to 8 (KenMesh and earlier versions of BertKenMeSH used batch size 16) to update the model compromising some training time. We have listed all the hyperparameter settings used in our experiment and bolded the values used for the final result in Table 4.2.

This final phase of software development saw a further refactoring of the code. KenMeSH uses PyTorch [32] to implement the model. We have designed our code with a new framework called PyTorch Lightning. This is an updated framework wrapper on PyTorch. It automates and saves time by optimizing the data loading and maintaining the training cycle.

4.3 Model Evaluation Techniques

There is no widely recognized benchmark for assessing multi-label classifications. To assess multi-label classification effectively, evaluation metrics adapted from multi-class classification and binary classification are utilised. Some of the popular evaluation techniques can be found in the articles by Tsoumakas et al. [38] and Kosmopoulos et al. [19]. In this section we present three groups of measures, namely bipartition-based, ranking-based and hierarchy evaluation. We used these evaluation mechanisms in our project to evaluate the quality of our classification models.

For the following description of the three evaluation techniques, we have taken $\{x_i, y_i\}_{i=1}^{N}$ pairs of the test set. Here x_i is the document text set. y_i is the vector of true labels, $y_i \in \{0, 1\}_{i=1}^{L}$. For each document i in y_i (0 denotes the label is not in the set, and 1 denotes the presence of the label). N denotes the number of test examples and L is the total number of labels. Given a document x_i, predicted labels by the classifier are $\{\hat{y}\}_{i=1}^{N}$, where $\hat{y} \in \{0, 1\}_{i=1}^{L}$, and ranking of the predicted labels indexes among the top k is denoted as $r_k(\hat{y})$, where $\hat{y} = \{\hat{y}\}_{i=1}^{N}$.

4.3.1 Bipartition-based Evaluation

The two types of bipartition assessment are example-based and label-based. Example-based measures produce the top-5, top-10, and top-15 rated labels across all of the test set's documents in terms of Hamming loss, precision, recall, and F-score (in our evaluation). Each label in the label set is used to produce the label-based evaluation metrics, such as the macro-average precision, micro-average precision, macro-average F-score, and micro-average F-score [40].

Example-based Evaluation

Hamming loss is used to determine how frequently a true label is incorrectly predicted, that is, whether a label that belongs to the document is predicted or one that does not. It is the proportion of incorrectly categorised labels to all other labels. The model performs better the lower the Hamming loss. Following is a definition of hamming loss:

$$Hamming\ Loss = \frac{1}{N \cdot L} \sum_{i=1}^{N} \sum_{j=1}^{L} (y_i \oplus \hat{y}_i)$$

where N is the number of test examples, L is the total number of labels, and $\oplus$ denotes exclusive-or (same as XOR in logic operation). $(y_i \oplus \hat{y}_i) = 1$ if $y_i \neq \hat{y}_i$, and 0 otherwise.

The percentage of pertinent labels that are returned for each test document is determined by example-based precision (EBP). It is defined as follows and is expressed by the proportion of properly identified positive predictions:

$$EBP = \frac{1}{N} \sum_{i=1}^{N} \frac{|y_1 \cap \hat{y}_i|}{|\hat{y}_i|}$$

Example-based recall (EBR) is the ratio of actual positive cases. It shows the fraction of relevant labels that have been successfully retrieved. The formula is:

$$EBR = \frac{1}{N} \sum_{i=1}^{N} \frac{|y_1 \cap \hat{y}_i|}{|y_i|}$$

Example-based F-score (EBF) is the harmonic mean of example-based precision and example-based recall, and is defined as:

$$EBF = \frac{1}{N} \sum_{i=1}^{N} \frac{2 \times |y_1 \cap \hat{y}_i|}{|y_i| + |\hat{y}_i|}$$

Label-based Evaluation

Two averaging processes, namely macro-average and micro-average, are taken into consideration in order to assess the efficacy of averaging across labels. Macro-average approaches calculate global averages for each label and treat each label equally. When calculating average metrics, micro-average approaches combine the scores of all labels and give high frequency labels greater weight [44].

Here TP_j, FP_j and FN_j are the true positives, false positives and false negatives, respectively, for each label l_j. L is the set of all labels.

$$Macro\text{-}average\ Precision = \frac{1}{L}\sum_{j=1}^{L}\frac{TP_j}{TP_j + FP_j}$$

$$Micro\text{-}average\ Precision = \frac{\sum_{j=1}^{L} TP_j}{\sum_{j=1}^{L} TP_j + \sum_{j=1}^{L} FP_j}$$

$$Macro\text{-}average\ Recall = \frac{1}{L}\sum_{j=1}^{L}\frac{TP_j}{TP_j + FN_j}$$

$$Micro\text{-}average\ Recall = \frac{\sum_{j=1}^{L} TP_j}{\sum_{j=1}^{L} TP_j + \sum_{j=1}^{L} FN_j}$$

$$Macro\text{-}average\ F\text{-}score = 2 \times \frac{Macro\text{-}average\ Recall \times Macro\text{-}average\ Precision}{Macro\text{-}average\ Recall + Macro\text{-}average\ Precision}$$

$$Micro\text{-}average\ F\text{-}score = 2 \times \frac{Micro\text{-}average\ Recall \times Micro\text{-}average\ Precision}{Micro\text{-}average\ Recall + Micro\text{-}average\ Precision}$$

4.3.2 Ranking Based Evaluation

The goal of ranking-based evaluation is to rate the relevant labels higher than the irrelevant ones for the expected labels. It is a reliable assessment that takes into account outliers in the returning projected set. In this thesis, the performance of classification models is assessed using accuracy at k and normalised discounted cumulative gain.

By taking into account the most pertinent labels produced by the classification procedure, precision at k ($p@k$) analyses labels that have been recovered at a specific cutoff rank. $p@k$ is a localised ranking method as a result that assesses the quality of ranking based on the top k most significant retrieved labels [24]. This can be defined as:

$$p@k = \frac{1}{k}\sum_{l \in r_k(\hat{y})} y_l$$

4.4 Results and Discussion

We started our experiment to improve MeSH indexing by replacing the BioWordVec MeSH term embeddings with PubMedBERT MeSH term embeddings generated from both MeSH terms and their respective descriptions. This extra information from the MeSH descriptions and

the rich embedding generated from PubMedBERT was meant to improve MeSH term indexing with these enhanced label features. But, after training on the MeSHup corpus we saw a drop in precision and subsequently all other evaluation metrics. After investigating this outcome, we concluded that reducing the dimension of the PubMedBERT embedding from 768 to 200 has resulted in a loss of relevant information and has failed to give the label features the required context to enhance the MeSH indexing task. The dimension reduction was necessary to match with the other parts of KenMeSH. The results of this phase, which we have called Phase I, are presented in Table 4.3 in the row titled KenMeSH with PubMedBERT Embeddings.

To tackle this issue, we decided to replace the BioWordVec and Bi-LSTM layers with a PubMedBERT layer to align the remaining part of the project with the rich label features generated in Phase I of our experiment. This step needed a major refactoring of the KenMeSH source code and eventually we developed a new model architecture. We named our new model BERTKenMeSH. We than trained our BERTKenMeSH model with the MeSHup corpus. This setup was looking very promising in theory, but it ended up with some extremely low precision and recall values. This low precision and recall result affected all other evaluation metrics because they are highly dependent on precision and recall. From the results in Table 4.3 labelled BERTKenMeSH (3 layer CNN), we can see that we got *nan* values for most of the evaluation metrics. *nan* stands for not a number. Dividing any number by zero results in a zero division error and is returned as a *nan* value. From the evaluation metric equations described in Section 4.3 we know that the calculations depend heavily on the number of correct predictions. In our case the precision and recall were so low that when they were being calculated in the overall context for measuring Micro-average and example based precision and recall they resulted in 0 or *nan*.

We then started investigating thoroughly each part of our model including the application of label features and label wise attention, which was one of the major contributions of KenMeSH, in an effort to make sure that the label wise attention is working properly on our newly designed BERTKenMeSH model. We examined outputs from each step of our models randomly. After in-depth examination, our findings revealed that the resulting feature vectors were too complex for a 3-layer dilated CNN to effectively capture deeper semantic information. As a result, we observed poor precision, which had a significant impact on the example-based precision, recall, and F1-score. Notably, the values of these measures are highly dependent on the precision, which further compounded the challenges we faced in obtaining accurate results. The poor performance of our approach is reflected in all measures, including MiP, MiR, and MiF, underscoring the need for further research to optimize and refine the use of advanced techniques such as BERT in the MeSH indexing process. This underscores the importance of carefully evaluating the performance of new models and techniques and considering the

Kernel size	Dilation	p@1	p@3	p@5	EBP	EBR	EBF
3	[1,2,4,8,16]	0.588	0.229	0.146	0.005	0.534	0.010
3	[1,2,4,2,1]	0.544	0.214	0.136	0.003	0.648	0.006
5	[1,2,4,2,1]	0.587	0.229	0.146	0.004	0.647	0.007
5	[1,2,4,8,16]	0.028	0.021	0.020	0.003	0.188	0.006
3	-	0.073	0.074	0.073	0.001	0.821	0.001
4	[1,2,4,2,1]	0.587	0.231	0.147	0.003	0.648	0.006
4	[1,2,4,8,4,2,1]	0.567	0.227	0.145	0.005	0.705	0.010
5	[1,2,4,8,4,2,1]	**0.595**	**0.232**	**0.147**	0.199	0.130	**0.157**
7	[1,2,4,8,4,2,1]	0.582	0.228	0.145	**0.562**	0.053	0.106
7	[2,4,8,16,8,4,2]	0.572	0.222	0.140	0.152	**0.016**	0.028
5	[2,4,8,16,8,4,2]	0.593	0.230	0.145	0.002	0.161	0.005

Table 4.1: Comparison among experimented models

limitations and challenges that can arise in their application to complex problems like MeSH indexing.

In light of our previous findings, we started experimenting on the dilated CNN module of our model. We used a small data set of 10,000 data and evaluated our model for various dilated CNN varying on layer depth, kernel size and dilation rate. We present a detailed comparison of various model architectures that we have experimented with to determine the most effective and accurate model configuration for our task in Table 4.1. We have conducted thorough investigations to identify the optimal combination of hyper-parameters and model architectures that yielded the best results for our task. By performing these experiments, we aimed to gain a deeper understanding of the impact of different configurations on the performance of our model, and ultimately determine the most suitable model settings for our specific task.

We compiled a comparison of the models we experimented with to find the most optimised dilated CNN settings for our task in Table 4.1. Our experiment uses a subset of 10,000 documents from our data set, with 20% for testing and 2% for validation. However, this data set is relatively small for the task of multi-label classification involving 28,415 MeSH terms. Due to the limited data, some MeSH terms are not represented in the data set, while other common terms like “Humans” are repeated 10,699 times. This has led to an imbalance in the data set, resulting in poor performance in our evaluation metrics. Nevertheless, this experiment has provided us with insights into which models could potentially perform better when trained on the full data set. We will mainly focus on the precision at 1, 3, and 5 scores for each model. Example-based precision, recall, and F1-score depend on the threshold and we can get different results by tweaking the threshold. Here we used thresholds 0.5, and 0.05 at different times to calculate the EBP, EBR, and EBF scores and included the best result for the model.

The findings of our Phase III experiment are now ready for review and analysis. We start

	Parameters	KenMeSH	KenMeSH with BERT embedding	BERTKenMeSH with 3 layer CNN	BERTKenMeSH with 7 layer CNN
CNN	Kernel size	**3**, 5, 10	**3**,5	**3**,5	3,**5**,7
	Filter size	**128**, 256	128	128	128
	Dropout	**0.2**, 0.5	0.2	0.2	0.05
	Dilation	[1,2,3]	[1,2,3]	[1,2,3]	[1,2,4,8], [1,2,4,8,16], [1,2,4,2,1], **[1,2,4,8,4,2,1]**
GCN	Hidden units	200	200	768	768
General	Epochs	**10**,20	10	10	20
	Batch size	4,8,**32**	32	8	8
	Learning rate	**1e-4**, 5e-4, 1e-5	1e-4	1e-4	dynamic starting from 1e-4

Table 4.2: Hyperparameters

with a detailed comparison of various hyper-parameter settings and model architectures that we experimented with in different phases of our experiments. Throughout the experiments, we aimed to gain a deeper understanding of the impact of different configurations on our model's performance and ultimately determined to use the most suitable model settings for our specific task for that phase. We have compiled all the hyper-parameter combinations we experimented with into Table 4.2 for your reference. The ones finally chosen are in bold.

We will now compare the evaluation of our model against the state-of-the-art model, KenMeSH, and another successful model, BERTMeSH. Even though BERTMeSH is trained on full PMC articles, we will include this in our comparison because of it uses the BERT model for extracting features from the input. Our proposed model is trained on the title and abstract of each of the 1,052,177 documents in the MeSHup corpus [42], whereas, KenMeSH is trained on 1,284,308 documents from PubMed. The rest of the articles were divided into training and validation sets. For testing, we used the 268,412 latest articles from the corpus. We used 21,473 examples for validation.

In Table 4.3, we compared our model to earlier analogous systems using micro-average measure and example-based measure. The best score for each evaluation metric is shown bolded in each row of the table, which includes all metrics for a particular approach.

KenMeSH has the state-of-the-art micro-average precision score in MeSH indexing. KenMeSH with BERT embedding has dropped a little on the micro-average precision from KenMeSH but it is still performing better than other MeSH indexing models like BERTMeSH, even though these models were trained on the full text. The decrement in the micro-average precision in KenMeSH with BERT embedding has led us to our next experiment of replacing the Bi-LSTM layer with PubMedBERT. But it has led to a failure because of the original 3-

Methods	Micro-average Measure			Example Based Measure		
	MiF	MiP	MiR	EBF	EBP	EBR
BERTMeSH (Full)	0.685	0.713	**0.659**	0.675	0.717	**0.667**
KenMeSH	**0.745**	**0.864**	0.655	**0.738**	**0.863**	0.644
KenMeSH with BERT Embedding	0.601	0.771	0.491	0.554	0.505	0.609
BERTKenMeSH (3 layer CNN)	nan	nan	0.0	0.0	0.0	nan
BERTKenMeSH (7 layer CNN)	0.091	0.323	0.053	0.246	0.396	0.179

Table 4.3: Comparison to previous methods across two main evaluation metrics. Methods marked as Full are trained on entire PMC articles; others are trained on abstracts and titles only. Bold: best scores in each column.

Ranking Base Measure	Methods			
	KenMeSH	KenMeSH with BERT embedding	BERTKenMeSH 3 layer CNN	BERTKenMeSH 7 layer CNN
$p@1$	0.993	0.972	0.090	1.0
$p@3$	0.972	0.921	0.020	0.674
$p@5$	0.937	0.857	0.025	0.526

Table 4.4: Comparison to KenMeSH across ranking-based measures. Bold: best scores in each row.

layer CNN structure which was not able to extract features from the rich BERT feature outputs. We then experimented with the new updated model architecture that we developed from our experiments shown in Table 4.1. The precision evaluation for 1, 3, and 5 predictions for each model are compiled in Table 4.4.

Our results reveal that we have a notable increment in precision, recall, and all other evaluation metrics from the previous BERTKenMeSH model with 3-layer CNN in Phase II. The $p@1$ value for our latest BERTKenMeSH model is 1.0, which denotes that our model is predicting at least one right label for most of the articles, if not for all of them. The *EBP*, *EBR*, and *EBF* values of 0.396, 0.179 and 0.246, respectively, indicate the huge improvement from Phase II, even though they have not yet reached the desired benchmark. Recall indicates how many of the true labels are present in our predicted result and we examined that we can achieve a example based recall of 1.0 if we choose a very low threshold for our predictions at the cost of very low precision. The example based measures indicate that even though our model is predicting many True Positives it is also predicting many False Positives. We also ranked very low on the Micro-average measures. The difference among $p@k$ where $k = 1, 3$, and 5 and

Epoch	Train Loss	Valid Loss
0	-	-
1	1.2	0.993
2	0.699	0.678
3	0.433	0.477
4	0.15	0.208
5	0.0954	0.089
6	0.126	0.046
7	0.0377	0.0254
8	0.0285	0.0172
9	0.191	0.013
10	0.0142	0.0103
11	0.0113	0.00854
12	0.00954	0.00742
13	0.00856	0.00653
14	0.00693	0.00582
15	0.00602	0.00534
16	0.00512	0.00485
17	0.00508	0.00457
18	0.00454	0.0043
19	0.004	0.00421

Table 4.5: Training and validation loss throughout 20 epochs

example based measures, and micro-average measures gives us the context that, our model is biased towards some MeSH terms that are most commonly found throughout the dataset.

In order to further our understanding, we have made two investigations. First, we manually scrutinized randomly picked documents from our test data set and compared the original labels and the predicted labels. We found that our assumption based on the evaluation metrics is correct. In most of the examples, the true labels are present in the predictions, but the model is predicting many more False Positives. We can conclude from the above findings that we have overcome the major challenge in our BERTKenMeSH model. In Phase II, when we first introduced BERTKenMeSH, our model was not learning anything. In our current model, we have evidence that our model is learning and predicting the true labels. Second, with the goal of further investigating, we analyzed the training and validation loss for each epoch throughout the training. We have noted a continual decrease in the training and validation loss, this being a positive indication that learning is happening and the trend would suggest that learning will continue for some time. We present the gradual decrement of the loss from our training in Table 4.5. These two investigations indicate that our model will keep improving if we train the model for some more epochs.

Our model is currently performing poorly compared to KenMeSH in all other measures except for $p@1$. But the progress in the result indicates that the model is learning at every epoch. This gives us optimism that our BERTKenMeSH model is continuously improving and it may show promising results with further improvement in its learning.

Chapter 5

Conclusions and Future Work

Medical Subject Heading (MeSH) terms are used to index the 29 million journal articles in MEDLINE. This thesis has investigated some improvements to a state-of-the-art deep learning model, KenMeSH [40], that labels biomedical journal papers with multiple Medical Subject Heading terms. The improved model is called BERTKenMeSH.

In this concluding chapter we will discuss the key topics of our thesis and demonstrate some of the future plans that we have regarding this topic.

5.1 Conclusions

In conclusion, the KenMeSH project underwent significant improvements during our thesis. We were able to develop a new model architecture along with a more efficient and maintainable code. The MeSH term embeddings were also updated by replacing BioWordVec with BERT embeddings generated from the MeSH term descriptions, resulting in more accurate and informative representations of the terms. Moreover, we replaced the BioWordVec word embeddings and the Bi-LSTM with BERT to generate feature vectors, which proved to be a better approach for extracting features from the MeSH terms. We also conducted extensive research on various neural net structures, including different kernel sizes and dilation rates of the dilated CNN, to find the optimal solution for the problem. The result found at the time of writing this thesis shows an indication that the model can improve but needs further training. Overall, our findings contribute to the advancement of MeSH indexing methods and demonstrate the potential of using BERT embeddings and neural net structures in future research on this topic. These explorations led to the following contributions:

- We have taken the concept of a Graph Convolutional Network (GCN) from KenMeSH and have modified it to generate a rich label feature attention graph. GCN helps to iterate

to the adjacent nodes from a graph structure input data of MeSH terms. In the end we successfully exported a graph structure with all necessary information. In this process we replaced averaged BioWordVec word embeddings of the MeSH term phrase as node features with averaged BERT word embeddings of the description of the MeSH term.

- To generate BERT embeddings we used MeSH descriptions. We found a UMLS [4] data file that contains information on MeSH terms. We have carefully curated necessary information from the data file and created a corpus consisting of the MeSH description for each MeSH term.

- We altered the output dimension in Phase I (Section: 4.2.1) to match with the BioWordVec embedding and operated a maxpooling to train and build the graph. This complicated process takes almost a day and a half to train and generate the output. Our experiment shows that, reducing dimension from a BERT embedding may reduce efficiency due to the loss of information.

- Further, we took the implementation of K-Nearest Neighbours from KenMeSH and modified it to align with our new BERT word embedding. We used this model to find relationships among MeSH terms and their relevance in appearing together in journals and articles. The major challenge was dealing with such a large amount of data. We had to study Python memory management and how to run the model in an efficient way to reduce computational costs.

- We simplified the KenMeSH model architecture and refactored the code base. We also changed the implementation with a new Python framework called PyTorch Lightning [32]. This framework simplifies data loading and training cycles.

- The KenMeSH model architecture contains a BiLSTM and a 3-layer dilated CNN. We replaced this part of the architecture and redesigned it with BERT word embeddings and a pyramid structure 7 layer dilated CNN. This part of the architecture extracts features from the title and abstract of research articles. We introduce a new MeSH indexing model BERTKenMeSH.

- We took the concept of creating a masked label-attention from label features, and compared it against the MeSH masks. But in our case, the label feature was enhanced with BERT word embeddings.

- Our prepared MeSH corpus and curated dataset along with the implementation code is available to the public in the following location: `https://github.com/xdwang0726/KenMeSH/tree/anurag-dev`.

The implementation has some limitations due to the huge amount of time needed to deal with such a large dataset. Debugging and training the framework from the beginning is a time-costly task. On the other hand, our dataset is biased towards data available in the PubMed central database. Some biomedical articles never release free versions. Our framework might be biased and not perform as expected for those articles.

5.1.1 Future Work

In this thesis, we mainly focused on enhancing the semantic meaning of MeSH terms to increase the accuracy of tagging new research articles. We updated MeSH label features, we updated the classifier model structure through extensive research on the performance change, and we present a new refactored more maintainable code base. In the future, we will train our model for more epochs and evaluate the model to see the progress with the training. We also intend to train our model on the KenMeSH data set to get a clearer comparison with KenMeSH. We will also research more on various neural nets to increase the accuracy of our model.

We also have the full text for each document available in our MeSHup[41] data set. In our experiment, we were stuck to only the title and abstract because of the limitation of the BERT model to take a maximum of 512 tokens at once. In the future, we will like to upgrade the model to generate predictions from the full text of a document.

To calculate the output of the model, we manually set the number of MeSH terms returned from the model. We decided on the value of k to predict the top k MeSH terms, where $k \in \{5, 10, 15\}$. In the future, we would also like to add a ranking system after the classifier that can dynamically decide the number of MeSH terms relevant to the document to give a more accurate result.

The focus of this project is to ease the work of human annotators and not to replace them. In the future, we wish to develop a model that can automatically suggest important sentences or paragraphs in a research article to human annotators. This will result in easing the work for human annotators and a more accurate MeSH indexing.

Bibliography

[1] AR Aronson and FM Lang. An overview of MetaMap: historical perspective and recent advances. *J Am Med Inform Assoc*, 17(C):229–36, 2010 May-Jun. [PMID: 20442139; PMCID: PMC2995713.].

[2] AR Aronson, JG Mork, CW Gay, SM Humphrey, and WJ Rogers. The NLM Indexing Initiative's Medical Text Indexer. *Stud Health Technol Inform*, 107((Pt 1)):268–72, 2004. [PMID: 15360816].

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, (ICLR 2015)*, 2014.

[4] O Bodenreider. The Unified Medical Language System (UMLS): integrating biomedical terminology. *Nucleic Acids Research,*, 32(Database issue):D267–D270, 2004.

[5] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[6] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, 2014.

[7] CodeEmporium. Lstm networks - explained! `https://www.youtube.com/watch?v=QciIcRxJvsM&t=240s&ab_channel=CodeEmporium`, 2018. [Online; accessed 19-December-2022].

[8] KW Fung and O Bodenreider. Utilizing the UMLS for semantic mapping between terminologies. *AMIA Annu Symp Proc*, 8:266–70, 2005. [PMID: 16779043; PMCID: PMC1560893.].

[9] Francesco Gargiulo, Stefano Silvestri, and Mario Ciampi. Deep convolution neural network for extreme multi-label text classification. In *Proceedings of the 11th International Joint Conference on Biomedical Engineering Systems and Technologies (HEALTHINF 2018)*, pages 641–650, 2018.

[10] Linyuan Gong and Ruyi Ji. What does a textcnn learn? *arXiv preprint arXiv:1801.06287*, 2018.

[11] Yu Gu, Robert Tinn, Hao Cheng, Michael Lucas, Naoto Usuyama, Xiaodong Liu, Tristan Naumann, Jianfeng Gao, and Hoifung Poon. Domain-specific language model pretraining for biomedical natural language processing. *ACM Transactions on Computing for Healthcare*, 3(1):1–23, 2020.

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[13] Jeremy Howard. Lesson 4: Practical deep learning for coders. `https://www.youtube.com/watch?v=V2h3IOBDvrA&t=343s&ab_channel=JeremyHoward`, 20-Dec-2016. [Online; accessed 19-December-2022].

[14] Md. Zabirul Islam, Md. Milon Islam, and Amanullah Asraf. A combined deep CNN-LSTM network for the detection of novel coronavirus (COVID–19) using X-ray images. *Inform Med Unlocked*, 2020.

[15] Antonio Jimeno-Yepes, James Mork, Dina Demner-Fushman, and Alan Aronson. A one-size-fits-all indexing method does not exist: Automatic selection based on meta-learning. *Journal of Computing Science and Engineering*, 6(2):151–160, 2012.

[16] Antonio Jimeno-Yepes, James Mork, Dina Demner-Fushman, and Alan Aronson. Comparison and combination of several MeSH indexing approaches. In *AMIA Annual Symposium Proceedings*, pages 709–718, 2013.

[17] Qiao Jin, Bhuwan Dhingra, William Cohen, and Xinghua Lu. AttentionMeSH: Simple, effective and interpretable automatic MeSH indexer. In *Proceedings of the 6th BioASQ Workshop A challenge on large-scale biomedical semantic indexing and question answering*, pages 47–56, 2018.

[18] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, 2014.

[19] Aris Kosmopoulos, Ioannis Partalas, Eric Gaussier, Georgios Paliouras, and Ion Androutsopoulos. Evaluation measures for hierarchical classification: a unified view and novel approaches. *Data Mining and Knowledge Discovery*, 29(3):820–865, sep 2014.

[20] Jimmy Lin and W. John Wilbur. PubMed related articles: a probabilistic topic-based model for content similarity. *BMC Bioinformatics*, 8:1471–2105, 2007.

[21] Jingzhou Liu, Wei-Cheng Chang, Yuexin Wu, and Yiming Yang. Deep learning for extreme multi-label text classification. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, pages 115–124, 2017.

[22] Ke Liu, Shengwen Peng, Junqiu Wu, Chengxiang Zhai, Hiroshi Mamitsuka, and Shanfeng Zhu. MeSHLabeler: improving the accuracy of large-scale MeSH indexing by integrating diverse evidence. *Bioinformatics*, 31(12):i339–347, 2015.

[23] Inneke Mayachita. Understanding graph convolutional networks for node classification. `https://towardsdatascience.com/understanding-graph-convolutional-networks-for-node-classification-a2bfdb7aba7b`, 2020. [Online; accessed 20-December-2022].

[24] Brian McFee and Gert R. G. Lanckriet. Metric learning to rank. In *Proceedings of the 27th International Conference on Machine Learning*, 2010.

[25] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *1st International Conference on Learning Representations, (ICLR 2013)*, 2013.

[26] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, volume 26, 2013.

[27] James G. Mork, Antonio Jimeno-Yepes, and Alan R. Aronson. The NLM medical text indexer system for indexing biomedical literature. In *Proceedings of the First Workshop on Bio-Medical Semantic Indexing and Question Answering, a Post-Conference Workshop of Conference and Labs of the Evaluation Forum 2013 (CLEF 2013)*, volume 1094 of *CEUR Workshop Proceedings*, 2013.

[28] National Library of Medicine. Medline citation counts by year of publication. `https://www.nlm.nih.gov/bsd/medline_cit_counts_yr_pub.html`.

[29] National Library of Medicine. Pubmed search engine. https://pubmed.ncbi.nlm.nih.gov.

[30] Christopher Olah. Understanding lstm networks. https://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2018. [Online; accessed 19-December-2022].

[31] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[32] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *33rd Conference on Neural Information Processing Systems (NeurIPS 2019),*, 2019.

[33] pawangfg. Explanation of BERT model — NLP. https://www.geeksforgeeks.org/explanation-of-bert-model-nlp/.

[34] Anthony Rios and Ramakanth Kavuluru. Convolutional neural networks for biomedical text classification: Application in indexing biomedical articles. In *Proceedings of the 6th ACM Conference on Bioinformatics, Computational Biology and Health Informatics*, BCB '15, pages 258–267, 2015.

[35] Sudipta Singha Roy. Investigating citation linkage as a sentence similarity measurement task using deep learning. Master's thesis, The University of Western Ontario, 2020.

[36] Lei Tang, Suju Rajan, and Vijay K. Narayanan. Large scale multi-label classification via metalabeler. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, page 211–220, 2009.

[37] Easy TensorFlow Team. Vanilla RNN for digit classification. http://www.easy-tensorflow.com/tf-tutorials/recurrent-neural-networks/vanilla-rnn-for-classification, 2018. [Online; accessed 19-December-2022].

[38] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. Mining multi-label data. In *Data Mining and Knowledge Discovery Handbook*, pages 667–685. Springer, 2009.

[39] Krina Vasa. Text classification through statistical and machine learning methods: A survey. *International Journal of Engineering Development and Research*, 4:655–658, 2016.

[40] Xindi Wang. Incorporating figure captions and descriptive text into mesh term indexing: A deep learning approach. Master's thesis, The University of Western Ontario, 2019.

[41] Xindi Wang, Robert Mercer, and Frank Rudzicz. KenMeSH: Knowledge-enhanced end-to-end biomedical text labelling. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2941–2951, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[42] Xindi Wang, Robert E. Mercer, and Frank Rudzicz. Meshup: A corpus for full text biomedical document indexing. In *Proceedings of the 13th Conference on Language Resources and Evaluation (LREC 2022)*, page 5473–5483, 2022.

[43] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021.

[44] Yiming Yang. An evaluation of statistical approaches to text categorization. *Information Retrieval*, 1:69–90, 1999.

[45] Zhang Yijia, Qingyu Chen, Zhihao Yang, Hongfei Lin, and Zhiyong lu. BioWordVec, improving biomedical word embeddings with subword information and MeSH. *Scientific Data*, 6, 52, 2019.

[46] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *4th International Conference on Learning Representations (ICLR 2016)*, 2016.

[47] Peng Zhou, Wei Shi, Jun Tian, Zhenyu Qi, Bingchen Li, Hongwei Hao, and Bo Xu. Attention-based bidirectional long short-term memory networks for relation classification. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 207–212, 2016.

Curriculum Vitae

Name:	Anurag Bhattacharjee
Post-Secondary Education and Degrees:	University of Western Ontario London ON, Canada Master's Studies in Computer Science, 2021-2022 Supervisor: Dr. Robert E. Mercer
Honours and Awards:	Western Graduate Research Scholarship 2018-2019
Related Work Experience:	Teaching Assistant The University of Western Ontario 2018-2020