DEEP NEURAL RANKING MODELS FOR MYANMAR NEWS RETRIEVAL

HAY MAN OO

UNIVERSITY OF COMPUTER STUDIES, YANGON

JULY, 2024

Deep Neural Ranking Models for Myanmar News Retrieval

Hay Man Oo

University of Computer Studies, Yangon

A thesis submitted to the University of Computer Studies, Yangon in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

July, 2024

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research and has not been submitted for a higher degree to any other University or Institution.

…..……………………………   …………........…………………………

Date   Hay Man Oo

ACKNOWLEDGEMENTS

First of all, I would like to thank the Union Minister, the Ministry of Science and Technology for granting me for full facilities support during the Ph. D Course at the University of Computer Studies, Yangon.

I would like to express my special appreciation and deepest gratitude to Dr. Mie Mie Thet Thwin, Former Rector of the University of Computer Studies, Yangon, for giving me kindness and morality supports.

Secondly, I would like to express very special thanks to Dr. Mie Mie Khin, the Rector, the University of Computer Studies, Yangon, for allowing me to do this research and giving me technical supports during the period of my research.

My greatest pleasure and the deepest appreciation to my supervisor, Dr. Win Pa Pa, Professor, Natural Language Processing Lab., University of Computer Studies, Yangon for her kindly support, encouragement, excellent mentorship, constructive comments and practical advice throughout the research period. This research would not be possible without her supervision, invaluable guidance and constructive ideas. I really gratitude for her supports which lead me to this point. She helped me in developing skills to present the progress of research work and enhancing my interests in the area of Information Retrieval.

I would like to thank and respect to Dr. Sabai Phyu, Professor, and former Dean of the Ph.D 11th Batch, University of Computer Studies, Yangon, for her clear guidance, inspiration and encouragement.

I greatly appreciate and acknowledge the kindly support, motivation, encouragement and valuable suggestions received from Dr. Tin Thein Thwel, Professor, University of Computer Studies, Yangon, for her advice and encouragement during my research.

I would like to deeply special thanks and appreciate to Dr. Si Si Mar Win, Professor and Dean of the Ph.D 11th Batch, University of Computer Studies, Yangon, for her great patience, valuable suggestions, motivation and kindly supports in everything.

I deeply and specially thank the external examiner, Dr. Aung Nway Oo, Professor, University of Information Technology, for her patience in critical reading, valuable comments and suggestions in the preparation of the dissertation.

I would like to mention my special appreciation and thanks to Dr. Khin Mar Soe, Professor, Head of Natural Language Processing Lab., University of Computer Studies, Yangon, for her insightful advice and kindly suggestions, and providing me with the institutional resources.

I would like to extend my special thanks to Dr. Aye Mya Hlaing, Associate Professor, Natural Language Processing Lab., University of Computer Studies, Yangon, for her excellent guidance, caring, patience, and providing me with excellent ideas and kindly supports in everything.

I am indeed obliged to Daw Mya Thandar, Associate Professor of English Department., University of Computer Studies, Yangon, for her careful assistance from the language point of view and pointed out the correct usage in my dissertation.

My sincere thanks also go to all my respectful teachers for giving us valuable lectures and knowledge during the Ph.D. course work.

I also thank my friends from Ph.D. 11th batch for providing support and friendship that I needed.

Last but not least, I would like to express my indebtedness and gratitude to my beloved parents, aunts and only one brother for their positive encouragement in morality, eternal love, endless support and always believing in me. They are always supporting and encouraging me during the research journey and also helping the financial supports.

ABSTRACT

This dissertation focuses on enhancing Myanmar Information Retrieval (IR) system to generate more natural text for a given input text. Typical IR systems have two main components: text query (user needs or preferences) and text documents (related to text query). Both text query and documents are important for the clarity and effectiveness of the IR system. Therefore, this research is emphasized on both text query and documents in Myanmar IR system.

In the contemporary era dominated by Information Technology (IT), search engines such as Google have become ubiquitous tools for individuals seeking access to a vast array of information. These platforms serve as indispensable resources, enabling users to effortlessly locate and acquire knowledge on a myriad of topics according to their needs and interests. Searching for News in English or Myanmar has become incredibly convenient, requiring a minimal effort to access a wealth of information.

The structure of IR has been altered dramatically by the inclusion of neural models, facilitating a more refined analysis of textual data. The textual data for Myanmar News dataset has been prepared in this research. In this research, the Myanmar News dataset was collected from Myanmar News website. In this dataset, each document contains two parts: title and contents.

The evaluations on different neural ranking models were conducted and so the results are thoroughly analyzed and discussed. A comprehensive analysis has started, with immersion in the use of various neural ranking models to comprehend intricate semantic connections, ultimately enhancing the effectiveness of IR systems. Pivotal neural ranking models such as DRMM, MP, Duet, KNRM, PACRR, CONV-KNRM, MZ-CONV-KNRM, which have left a profound impact on the field, are delved deep into, investigating their implications for enhancing the precision and efficiency of retrieval systems.

Another evaluation was done using a fine-tuning approach with the pre-trained model, Vanilla-BERT. The superior performance of this model compared to baseline methods, showcasing improvements in MAP, MRR, P@1 and P@3 overall retrieval performance. The implications of these findings extend to retrieve the similarity score results, highlighting the potential for enhanced IR capabilities.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

CHAPTER 1
INTRODUCTION

Text Information Retrieval (IR) systems aim to retrieve text documents that can satisfy the information requirements of their users, usually conveyed through textual queries. Over time, this inherently ambiguous description has been standardized and defined based on the particular characteristics of documents, information requirements, and users. The essence of the formalization revolves around the notion of a document's relevance concerning a query and the methods for assessing relevance. Over time, numerous ranking models have been suggested to gauge the relevance of documents in response to a query. These models rely on the data presented by the queries and documents, which are utilized to generate 'relevance signals'. Numerous ranking models have emerged over time, spanning from Boolean models to probabilistic and statistical language models. These 'bag of words' models utilize the presence or frequency of query terms in documents to deduce their relevance to a query, utilizing manually created functions to aggregate these instances, such as Best Matching 25 (BM25).

As the Internet and social platforms have become more prevalent, additional sources of relevance information about documents have been recognized. Machine learning techniques have demonstrated their effectiveness in handling the multitude of relevance signals. Their utilization to prioritize documents based on relevance estimates relative to a query has led to the development involving numerous Learning-To-Rank (LTR) models. Relevance signals serve as input features in LTR models, and they are frequently crafted manually, which can be a time-consuming endeavor. Inspired by their advancements in various computer vision and Natural Language Processing (NLP) tasks, neural networks currently stand as the state-of-the-art method for ranking documents in terms of query relevance.

Neural Information Retrieval (Neural IR) is centered on the retrieval of text documents that meet the information requirements of users, utilizing deep neural networks. In Neural IR, neural networks are commonly employed in two distinct manners: first, to acquire the ranking functions that amalgamate relevance signals for arranging documents, and second, to acquire abstract representations of documents and queries to encapsulate their relevance information.

Below, an overview of the latest methodologies in Neural IR is presented below the chapter 2. Considering the rapidly evolving nature of research in this field, it is acknowledged that the coverage may not encompass every aspect of Neural IR. However, a structured introduction to the key concepts and current systems within the field are aimed to be offered.

In the contemporary era, an age dominated by information is inhabited, where an unprecedented volume of data is generated by individuals, reaching into the quintillions of bytes on a daily basis. Within the domain of IR systems, serve as efficient tools designed to swiftly retrieve necessary information from vast and extensive data collections. IR systems have seamlessly integrated into the fabric of daily existence, with ubiquitous search engines like Google processing billions of searches daily. Virtually every Myanmar News website incorporates a search bar, allowing individuals to effortlessly locate articles, products, individuals, and more. Beyond their direct impact on users, IR systems play a pivotal role in supporting various Artificial Intelligence (AI) applications, sourcing valuable data for downstream tasks like data analysis, recommendation systems, Question Answering (QA), and dialogue generation. Nevertheless, the exclusive emphasis which is on learning relevance patterns demands extensive training data and still falls short in achieving robust generalization, particularly when faced with tail queries [2] or unexplored search domains [3].

At the heart of IR lies the essential task of evaluating the relevance between a user's query and a document. Contemporary IR systems have traditionally depended on bag-of-words retrieval models, wherein the number of shared words between the query and the document is computed for relevance assessment. This streamlined representation of natural language facilitates the retrieval system in swiftly scanning through vast collections of millions or billions of documents, rendering large-scale retrieval feasible and efficient. Nevertheless, the practice of simply counting shared words provides a limited and superficial method for modeling the relevance in search contexts. At an intuitive level, an improved IR system ought to possess the capability to comprehend the meanings embedded in the text and discern the semantic relationships between queries and documents.

There has been a significant surge in interest and attention towards enhancing language understanding within the realm of IR. Despite concerted efforts, the incorporation of advanced NLP techniques into retrieval has, for the most part, yielded limited success. In recent times, neural networks have emerged as a potent and transformative paradigm for modeling natural language. The focus of this dissertation is to improve language comprehension in IR through the utilization of neural networks. It centers on addressing two core challenges within IR: the representation of textual content and the modeling of relevance. The discussion delves into the inherent difficulties of these challenges, introduces innovative neural network methodologies to overcome them, and showcases the effectiveness of these approaches in surpassing the limitations of earlier state-of-the-art retrieval systems.

This research utilizes different neural network methods applied NLP and Deep Learning (DL) techniques on a collection of documents. Neural ranking models training Myanmar news datasets were enriched.

1.1 Problem Statements

Neural IR is a field that addresses various challenges and problems related to improving the effectiveness of IR systems using neural network-based techniques.

A significant problem in Neural IR as semantic matching is how to develop models that can understand and match the semantics of queries and documents for more accurate retrieval.

Neural IR aims to improve the ranking of documents by developing models as ranking relevance that can accurately and efficiently score the relevance of documents to queries.

In many IR applications, it can be challenging to obtain large amounts of labeled data for training neural models learning from limited data. Developing techniques for effective learning from limited data is a problem, especially when pre-training on large corpora is not feasible.

Struggled with the development and evaluation of Neural IR models, the comparison of different models and techniques is included as model development and comparison:.

Needed access to pre-trained models and datasets to develop, experiment with Neural IR approaches efficiently accesses to pre-trained models and data.

1.2 Motivations of the Research

In the contemporary era dominated by IT, search engines such as Google have become ubiquitous tools for individuals seeking access to a vast array of information. These platforms serve as indispensable resources, enabling users to effortlessly locate and acquire knowledge on a myriad of topics according to their needs and interests. Searching for news and updates has become incredibly convenient, requiring minimal effort to access a wealth of information in Myanmar or English. The structure of IR has been altered dramatically by the inclusion of neural models, facilitating a more refined analysis of textual data.

1.3 Intentions of the Research

The main purpose of this research is to enhance Myanmar IR system that can generate the more relevance results score. For promoting the accuracy of ranking model, neural network architectures such as Deep Relevance Matching Model (DRMM), Match-Pyramid (MP), Duetl, Kernelized Neural Ranking Model (KNRM), Position-Aware Convolutional Recurrent Relevance (PACRR), Convolutional Kernelized Neural Ranking Model (CONV-KNRM) and MatchZoo-CONV-KNRM (MZ-CONV-KNRM) have been applied in Myanmar News Retrieval and the most suitable neural network architecture for Myanmar News has been investigated. The following are the other objectives:

1. To develop deep neural ranking model for Myanmar News Retrieval
2. To analyze different models in Neural IR systems
3. To assess the performance of Myanmar News Retrieval systems by using different evaluation metrics
4. To improve the relevance score of the similarity between user requirements and Myanmar News Dataset
5. To point out the importance of a large amount of Myanmar News corpus required for the development of Myanmar News Retrieval systems
6. To apply neural network architectures such as Deep Neural Ranking Models and Pre-trained Language Model for Myanmar News Retrieval

1.4 Contributions of the Research

This research has four main contributions. The very first contribution of this research is creating a new kind of annotated Myanmar News dataset (title and its related contents). Collection is manually to retrieve from Myanmar News websites.

Specifically in the recent year, IR as desired query or user preferences rarely searches for Myanmar News. Therefore, developing Myanmar News retrieval in IR System is the second contribution.

The third contribution is proposing and applying Myanmar News Dataset for query-document features extraction. Many IR systems give a query and return the documents. Therefore, the Myanmar News Dataset is proposed and applied with neural network extract the query-document features.

The final contribution of this research is applying Deep Neural Ranking Model and Fine-tuned Model for Myanmar News retrieval. Query-document features can be obtained by deep learning form large amount of Myanmar News dataset and applying these features to rank with neural network models and fine-tuned model.

1.5 Organization of Research

This dissertation is comprised with seven chapters including literature review, related work and background theory of IR research, building Myanmar News dataset for retrieval, description of proposed system architecture, nature of text data in Myanmar dataset, feature extraction process, implementing Deep Neural based and fine-tune based ranking models with data retrieving and ranking methods, experimental results, conclusion and future work of research on Myanmar News retrieval.

Chapter 1 describes the introduction, objectives, focus and contributions of the Myanmar News retrieval research work. The literature reviews on Neural IR, related work of this research, applied areas, previous researches of Neural IR system, pre-trained models, fine-tuned model, and evaluation metrics of Myanmar News retrieval are described in Chapter 2. Background theories required for Deep Neural Network process are described in Chapter 3. It includes the ranker models about DRMM, MP, Duetl, KNRM, PACRR, CONV-KNRM, and MZ-CONV-KNRM, likelihood of score computation and finally describes the performance metric. Chapter 4 explains how to collect and prepare the Myanmar News data from Myanmar website sources and building query-document datasets for Myanmar News retrieval. Moreover, the Myanmar information and statistics of Myanmar News datasets are also reported in this chapter. Chapter 5 describes general architectures of Myanmar IR system, and design and implementation of proposed system architecture for Myanmar News retrieval. Chapter 6 describes the performance analysis for Myanmar News retrieval. It includes the different rankers and fine-tuned models, the experimental setup, performance results and discussion about the ranking models. Moreover, the proposed model is compared with baseline models and word embedding features are also explored and investigated in this chapter. Finally, Chapter 7 presents the conclusion extracted from this research work with the advantages and limitations of research work and describes the future research lines to continue it.

CHAPTER 2
LITERTATURE REVIEW AND RELATED WORK

This chapter describes the literature review on Neural IR techniques, related work of this research and previous researches of Neural IR on Myanmar language. Evaluation metrics of Neural IR are also briefly presented in this chapter.

Machine learning plays a role in many aspects of modern IR systems, and deep learning is applied in all of them. The fast pace of modern-day research has given rise to many different approaches for many different IR problems where designing features used to be a crucial aspect and contribution of newly proposed IR approaches, and the focus has shifted to designing network architectures instead. As a consequence, many different architectures and paradigms have been proposed, such as auto-encoders, recursive networks, recurrent networks, convolutional networks, various embedding methods, deep reinforcement and deep-learning, and, more recently, generative adversarial networks, of which most have been applied in IR settings [34].

In recent time, machine learning models have surpassed state-of-the-art solutions across various domains, including health monitoring, computer vision and NLP. In particular, deep neural networks have been successfully applied to diverse area including IR [37]. A retrieval model aims to enhance traditional exact-match models like BM25 by incorporating semantic matching signals derived from a neural embedding matching model. This model trains the neural embedding to represent language structures and semantics that cannot be adequately captured by lexical retrieval methods, using a novel residual-based embedding learning approach [16].

The large-scale query-document retrieval problem: given a query (e.g., a question), return the set of relevant documents (e.g., paragraphs containing the answers) from a large document corpus. This problem is often solved by two steps. The retrieval phase first reduces the solution space, returning a subset of candidate documents. The scoring phase then re-ranks the documents. Critically, the retrieval algorithm not only desires high recall but also requires being highly efficient, returning candidates in time sublinear to the number of documents [8].

Ranking models lie at the heart of research on IR. During the past decades, different techniques have been proposed for constructing ranking models from traditional heuristic methods and probabilistic methods, to modern machine learning methods. Recently, with the advance of deep learning technology, it has witnessed a growing body of work in applying shallow or deep neural networks to the ranking problem in IR, referred to as neural ranking models [20].

Nearly three decades have passed since the introduction of BM25, and it has been more than ten years since the inception of learning-to-rank methodologies. Throughout the last few decades, researchers have conducted extensive investigations aimed at integrating advanced NLP techniques into IR. However, the majority of these endeavors has fallen short of practical implementation. The prevailing approach in applying NLP to IR involves utilizing linguistically motivated elements such as part-of-speech tags, grammar-based parsers, word correlations, etc., derived from both documents and queries. These linguistic features are then employed to create representations for retrieval purposes.

As an illustration, research has delved into the exploration of techniques like dependency parsing [71] and part-of-speech tagging [31] to enhance the understanding of queries and documents in IR. While these parsers provided inspiration, their fragility and limitation to well-formed text were notable drawbacks. Moreover, the extracted linguistic signals often necessitated additional complex processing to become valuable for retrieval.

A deep neural network comprises an extensive assembly of elementary mathematical units, referred to as neurons, arranged in layers. These layers can be collectively trained to perform intricate tasks. The hierarchical arrangement of layers in the model allows it to take raw data as input and progressively acquires high-level features through the learning process. One of the key advantages of neural networks over traditional feature-based NLP techniques is their capacity to autonomously uncover intricate features. Neural networks leverage distributed representations, which represent another notable strength of this approach. Nevertheless, within the domain of IR, the enhancements achieved by neural networks seem somewhat modest in comparison to the strides made by traditional techniques.

2.1 Overview of Neural Information Retrieval

The field of IR includes a wide range of content types and tasks. IR is the science of searching for and retrieving relevant information from a collection of documents. It is a fundamental task in many applications, including web search engines, digital libraries, and document management systems. Neural networks are a class of machine learning models motivated by the human brain. They consist of interconnected nodes (neurons) that process and transform data.

Neural IR is one of the important research topics in the field of text processing and is also known as Neural IR. Neural IR is an advanced area of research in the field of IR and NLP. It involves the use of neural networks and deep learning techniques to improve the process of retrieving relevant information from large collections of text documents, such as web pages, books, or other text-based retrieval. In the context of Neural IR, neural networks are used to model text data and relationships between documents and queries.

Neural networks are a class of machine learning models motivated by the human brain. They consist of interconnected nodes (neurons) that process and transform data. In the context of Neural IR, neural networks are used to model text data and relationships between documents and queries. In Neural IR, documents and queries are typically represented as numerical vectors. Each word or phrase in a document is converted into a vector using word embeddings (e.g., Word2Vec, GloVe) or more advanced techniques like transformer-based models (e.g., BERT). Neural IR is a sought out research topic in the IR research community. Since then, the Neural IR paradigm started, and many BERT-based text ranking approaches were developed. These are now usually deployed as re-ranker in multistage search architecture. A condensed multi-stage ranking architecture of this method is shown in Figure 2.1. First, it uses exact matching over inverted index to execute the retrieval step, also known as candidate generation or first stage retrieval, and then the documents that are sorted based on BM25 score. This is a less costly action. After reducing the candidate set to *k* documents, a more costly re-ranking phase is conducted, determining the ultimate ordering of the top *N* documents through the use of BERT-based Contextualized ranking models. The trade-off in search systems is improved by these distinct phases.

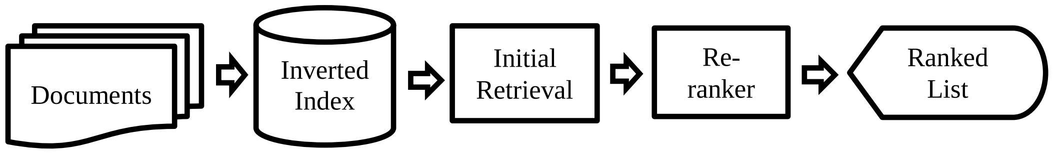

Figure 2.1 Example of Neural Information Retrieval (Neural IR)

2.1.1 Document Understanding

Neural IR aims to improve how well computers understand the content of documents. Instead of relying solely on keywords, it utilizes neural networks to create numerical representations of documents, capturing the context and semantics of the text. Document Understanding in Neural IR is a critical aspect of how search engines and other IR systems process, interpret, and retrieve information from large volumes of text.

Document understanding involves several processes that enable machines to comprehend the content, context, and meaning of documents. Text preprocessing are tokenization, stemming, lemmatization, and removing stop words. Feature extraction is extracting meaningful features from text, such as keywords, entities, and topics. Semantic understanding is the meaning and context of words and phrases using techniques like word embeddings and contextual embeddings.

Key techniques in document understanding of Neural IR have several neural techniques. These are word embedding (such as Word2Vec, GloVe, etc.), contextual embeddings (such as BERT, GPT, etc), sequence models (such as RNNs, LSTM, etc), transformers and document understanding. Word2Vec generates vector representations of words based on their context in the corpus and GloVe creates word embeddings by aggregating global word-word co-occurrence statistics. BERT (Bidirectional Encoder Representations from Transformers) provides deep contextualized word embeddings by considering both left and right context in all layers and GPT (Generative Pre-trained Transformer) uses transformer architecture for understanding and generating human-like text. Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) handle sequential data and capture dependencies over long ranges. Transformers handle long-range dependencies better than RNNs, enabling superior performance in tasks like document classification and summarization. Document understanding in Neural IR is a rapidly evolving field that leverages sophisticated neural network architectures to improve the comprehension and retrieval of information.

2.1.2 Semantic Matching

One of the core objectives is to achieve better semantic matching between user queries and documents. By encoding queries and documents as vectors, Neural IR models can compute similarity scores that go beyond simple keyword matching. Semantic matching in Neural IR is a critical component that aims to understand and match the meaning behind queries and documents rather than relying solely on keyword matching. This approach significantly enhances the accuracy and relevance of search results. Semantic matching involves comparing the meanings of words, phrases, sentences, or documents to determine their relevance to each other. It transcends traditional keyword-based methods by considering the context and semantics of the content.

Semantic matching in Neural IR is a transformative approach that enhances the relevance and accuracy of search results by focusing on the meaning behind words and phrases. By leveraging advanced neural network architectures and embedding techniques, it addresses the limitations of traditional keyword-based matching and paves the way for more intuitive and effective IR systems.

The key concepts in semantic matching are semantics, contextual understanding, and representation learning. Semantics refers to the meaning and interpretation of words and sentences. Contextual understanding captures the context in which words appear to understand their meanings better. Representation learning learns dense vector representations (embeddings) of words, phrases, or entire documents that capture semantic information.

Techniques and models for semantic matching are word embeddings (such as Word2Vec creates vector representations of words based on their context using models like Continuous Bag of Words (CBOW) and Skip-gram, GloVe (Global Vectors for Word Representation) generates word embeddings by aggregating global word-word co-occurrence statistics from a corpus), contextual embeddings (such as BERT, and ELMo). BERT (Bidirectional Encoder Representations from Transformers) provides deep contextualized word embeddings by considering the bidirectional context of words in sentences. It uses transformer architecture to capture complex relationships between words. ELMo (Embeddings from Language Models) generates contextual embeddings by considering the entire sentence in which a word appears. In order to appear a word using bi-directional LSTM networks.

Sentence and document embeddings (such as Universal Sentence Encoder, and Doc2Vec) are used to provide embeddings for sentences that capture their semantic content, using tasks like semantic similarity and text classification, Doc2Vec extends Word2Vec to generate embeddings for entire documents) and neural matching models (such as Deep Relevance Matching Model (DRMM) utilize a deep neural network to model the interaction between query and document terms at multiple levels of granularity. DSSM (Deep Structured Semantic Models) projects queries and documents into a common semantic space where similarities are computed using deep learning techniques).

2.1.3 Relevance Ranking

Neural IR models learn to rank documents based on their estimated relevance to a given query. By training on labeled data, where the relevance of documents is determined by human judgments. Relevance ranking in Neural IR is the process of ordering documents based on their relevance to a given query. This is a core task in IR systems like search engines, where the goal is to present the most pertinent information at the top of the results as shown in Figure 2.2.

Relevance ranking aims to order documents so that the satisfactions the user's query appears first. This involves evaluating the relevance of documents concerning the query and assigning a score that reflects their usefulness. Relevance ranking in Neural IR leverages advanced deep learning techniques to improve the accuracy and relevance of search results. By understanding and implementing these neural approaches, significantly enhance the performance of IR systems, making them more responsive to the user needs and more capable of understanding complex queries and documents.

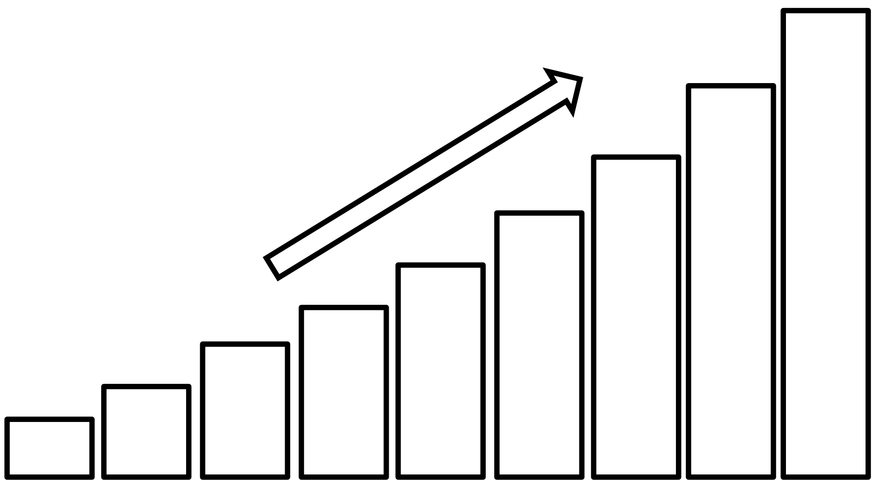

Figure 2.2 Example of Relevance Ranking

2.1.4 Representation Learning

Word embeddings are techniques like Word2Vec, GloVe, and FastText represent words as dense vectors in a continuous space, contextual embeddings are models like BERT, GPT, and ELMo provides context-sensitive embeddings by considering the surrounding words.

2.1.4.1 Text Representations for Ranking

The Probability Ranking Principle [65] states that, under certain conditions, the documents in a collection should be ranked in order of the (decreasing) probability of relevance with respect to the query for a given user's query. This will maximize the retrieval system's overall effectiveness for the user. Ad-hoc ranking's job is to determine which the order of the documents is closest to or identical to the best ordering based on relevance probability for each query. It is typical practice to restrict the set of documents to be sorted to the top *k* documents in the best possible order. For example *Q* represents a log of (text) inquiries and *D* represents a collection of (text) documents. The vocabulary *V* of terms is shared by both queries and documents.

A scoring function, or ranking function, is as follows: $Q \times D \rightarrow R$ calculates a real-valued score according to the log Q's queries for the documents in collection D. It refer to the value *s(q, d)* as the document's relevance score with respect to the query given a query *(q)* and a document *(d)* as in Equation (2.1). The scores of the documents in the collection can be used to order the documents for a given query, with the score values reversed. An IR system based on the scoring system is more effective the closer query that induced ordering.

$$s(q,d) = \quad f(\phi(q), \psi(d), \eta(q,d)) \tag{2.1}$$

where three representation functions, $\phi: Q \rightarrow V1, \psi: D \rightarrow V2 : Q \rightarrow V1, \psi: D \rightarrow V2$, and $\eta: Q \times D \rightarrow V3$, map queries, documents, and query-document pairings into the respective latent representation spaces, *V1*, *V2*, and *V3* [20]. These functions create computationally-friendly abstract mathematical representations of the text sequences of documents and queries. The aggregation function $f\text{: V1} \times \text{V2} \times \text{V3} \rightarrow \text{R}$ calculates the relevance score of the document representation with respect to the query representation. The elements of these vectors indicate the characteristics used to describe the relevant objects. The aggregation function f and the representation functions ϕ, ψ, and η can be computed using machine learning methods or they can be built manually using a few heuristics or axioms. The representation function used in LTR contexts, traditional IR as shown in Figure 2.3.

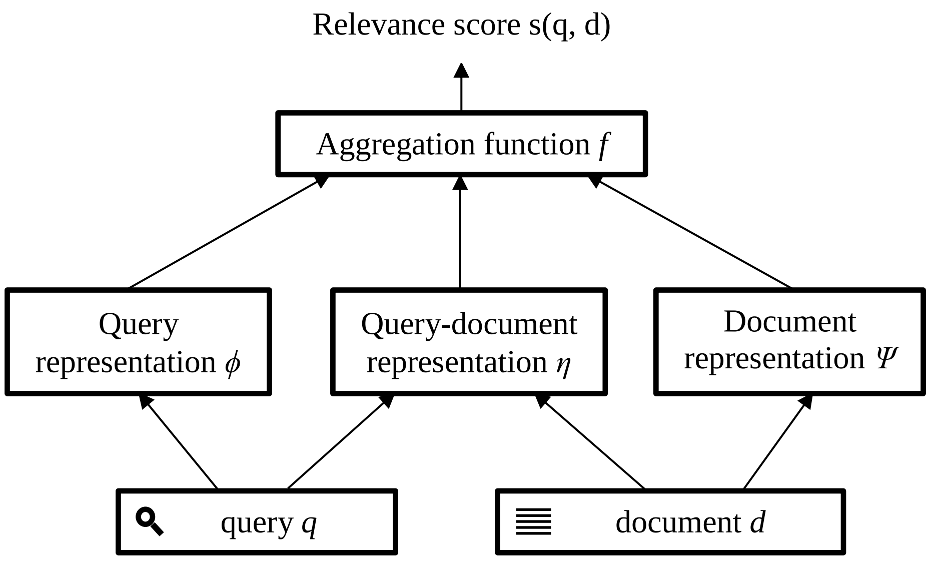
Relevance score s(q, d)
Aggregation function *f*
Query representation ϕ
Query-document representation η
Document representation Ψ
query *q*
document *d*

Figure 2.3 Representation-based decomposition of a ranking function

2.1.4.2 Bag-of-Word (BOW) Encodings

In traditional IR, the representation and aggregation functions are created by hand, incorporating lexical statistics like the quantity of terms that appear in a document or throughout the collections. The bag of words (BOW) model, which is the foundation for traditional IR ranking models such as Vector Space Models (VSM) [67], probabilistic models [66], and statistical language models [59], models queries and documents as a set of terms from the vocabulary V plus the number of occurrences of the corresponding tokens in the text. Formally speaking, queries and documents are represented as vectors $\phi(q)$ and $\psi(d)$ in $N|V|$, which are referred to as BOW encodings. The number of times a term appears is encoded by the i^{th} component of both representations. These ranking functions lack the query-document representation function η. The components of the query and document representations, i.e., the in-query and in-document term frequencies, along with additional document normalization processes, are all taken into consideration by the explicit formula that represents the aggregate function f over these representations. Since the majority of these representations' components equal zero because they reflect tokens that are absent from the query or document, they are known as sparse representations. Sparse representations are easily computed and effectively stored in specialized data structures known as inverted indexes, which serve as the foundation for commercial Web search engines [7]. For additional information on inverted indexes and classical IR ranking models, refer to [6], [48], [75].

2.1.4.3 Learning to Rank (LTR)

Pointwise approach treats the ranking problem as a regression or classification task where each document is scored independently examples include regression-based models, pairwise approach models the relative order of document pairs to minimize ranking errors examples include RankNet, and listwise approach optimizes the order of a list of documents, directly optimizing for ranking metrics examples include ListNet and LambdaMART(combines LambdaRank and Multiple Additive Regression Trees (MART)). New sources of pertinent information on the papers have been available since the introduction of the Web.

Relevance signals include Web page prominence (e.g., PageRank), extra document data (e.g., phrase frequencies in the title or anchor text), and search engine interactions (e.g., clicks). Furthermore, social media and collaborative websites like Facebook, Twitter, and Wikipedia are new sources of relevance signals. By utilizing these relevance signals, LTR's query and document representations have become richer. Features are the relevance signals that are taken out of documents and/or queries. These characteristics fall into a number of classes [34], [46], including: (1) query-only features, or elements of $\phi(q)$*:* query features, like query type, query length, and query performance predictors, that have the same value for every document; (2) query-dependent features, or components of $\eta(q, d)$*:* document features that depend on the query, like different term weighting models on different fields; (3) query-independent features, or components of $\psi(d)$*:* document features with the same value for each query, like importance score, URL length, and spam score.

The representation functions in LTR are manually created. Using feature-specific algorithms, the various parts of query and document representations are produced by taking use of relevance signals from heterogeneous information sources. With respect to vector spaces over *R*, the representations $\phi(q)$*,* $\psi(d)$*,* and $\eta(q, d)$ are thus elements whose dimensions are determined by the quantity of manually created query-only, query-independent, and query-dependent features, respectively. Furthermore, these vectors' various parts are diverse and lack any clear semantic significance. Through the use of various representations, the aggregation function f in LTR is machine-learned, for instance through the use of neural networks [4], logistic regression [17], or gradient-boosted regression trees [5]. For a thorough overview, it is seen [41].

2.1.4.4 Word Embeddings

Although LTR features and BOW encodings are commonly used in commercial search engines, they have various drawbacks. On the one hand, concepts that are semantically linked wind up with entirely distinct BOW encodings. While the names catalogue and directory are interchangeable, their BOW encodings differ significantly, with the single 1 occurring in distinct components.

In a similar vein, two publications pertaining to the same subject may end up with two distinct BOW encodings. Conversely, LTR features use feature engineering to manually generate text representations with heterogeneous components and no explicit notion of similarity. Despite being often employed in commercial search engines, LTR features and BOW encodings have a number of disadvantages.

On the one hand, semantically related concepts result in completely different BOW encodings. Although the terms "catalogue" and "directory" are synonymous, their BOW encodings are very different, with the former occurring in separate components. Similarly, two publications that deal with the same topic could have two different BOW encodings. On the other hand, LTR features do not explicitly consider similarity and instead manually build text representations with heterogeneous components through feature engineering. Moreover, the components of word embeddings are rarely 0: they are real numbers, and can also have negative values. Hence, word embeddings are also referred to as dense representations. Among the different techniques to compute these representations, there are algorithms to compute global representations of the words, i.e., a single fixed embedding for each term in the vocabulary, called static word embeddings, and algorithms to compute local representations of the terms, which depend on the other tokens used together with a given term, i.e., its context, called contextualized word embeddings. Static word embeddings used in Neural IR are learned from real-world text with no explicit training labels: the text itself is used in a self-supervised fashion to compute word representations. There are different kinds of static word embeddings, for different languages, such as Word2Vec [47], Fasttext [30] and GloVe [56].

Based on the training data used to compute the vectors, static word embeddings map terms with multiple senses into an average or most common sense representation; each term in the lexicon is associated with a single vector. Contextualized word embeddings correlate each term in the lexicon with a unique vector each time it appears in a document, based on the surrounding tokens. They map tokens used in a given context to a specific vector. Deep neural networks, such as the Bidirectional Encoder Representations from Transformers (BERT) [12], the Robustly Optimized BERT Approach (RoBERTa) [42], and the Generative Pre-Training models (GPT) [61], are used to learn the most widely used contextualized word embeddings. In Neural IR (nervous IR), word embeddings are utilized to calculate the aggregation function f and the representation functions ϕ, ψ, and η via (deep) neural networks. Neural ranking models fall into two categories: interaction-focused models and representation-focused models, depending on the assumptions made about the representation functions. In models that emphasize interaction, the function that represents the relationship between the query and document contents, known as the query-document representation function $\eta(q,d)$ is both explicitly created and fed into a deep neural network or it is implicitly generated and utilized directly by the deep neural network. The query-document representation function $\eta(q,d)$ does not exist in representation-focused models; instead, deep neural networks independently compute the query and document representations, $\phi(q)$ and $\psi(d)$.

2.2 Interaction-focused Systems

Deep Neural Networks (DNNs) are utilized by the interaction-focused Neural IR systems to model word and n-gram relationships between a query and a document. These systems take in a query (q) and a document (d) as inputs, and produce a query-document representation $\eta(q,d)$ as their output. Convolutional neural networks and transformers are two neural network architectures that have been studied among others to create a representation of these interactions. Convolutional neural networks are among the earliest methods for creating combined representations of documents and questions. The development of pre-trained language models through the use of transformers on textual inputs marked a significant turning point in Neural IR research. In Neural IR, query-document representations are computed using pre-trained language models; BERT and T5 (Text-to-Text Transfer Transformer) are the two primary transformer models utilized for this purpose, respectively. Query-document representations are computed using pre-trained language models and how these models are adjusted to handle lengthy texts.

2.2.1 Convolutional Neural Networks (CNNs)

A Convolutional Neural Network (CNN) is a class of neural networks intended to identify local patterns in structured inputs like texts and images [38]. When combined with the feed forward and pooling layers, the convolution layer forms the fundamental part of a CNN. A convolutional layer is considered as a tiny linear filter that scans the input for proximity patterns.

CNN are used by a number of neural models to generate relevance scores based on the interactions between queries and documents. These models typically include aggregating the word embeddings of the query and document tokens into an interaction matrix, which is then utilized by CNN to learn hierarchical proximity patterns like bigrams, unigrams, and so forth. The relevance score $s(q, d)$ between the query q and the document d is then produced by feeding the final top-level proximity patterns into a feed forward neural network, as shown in Figure 2.4.

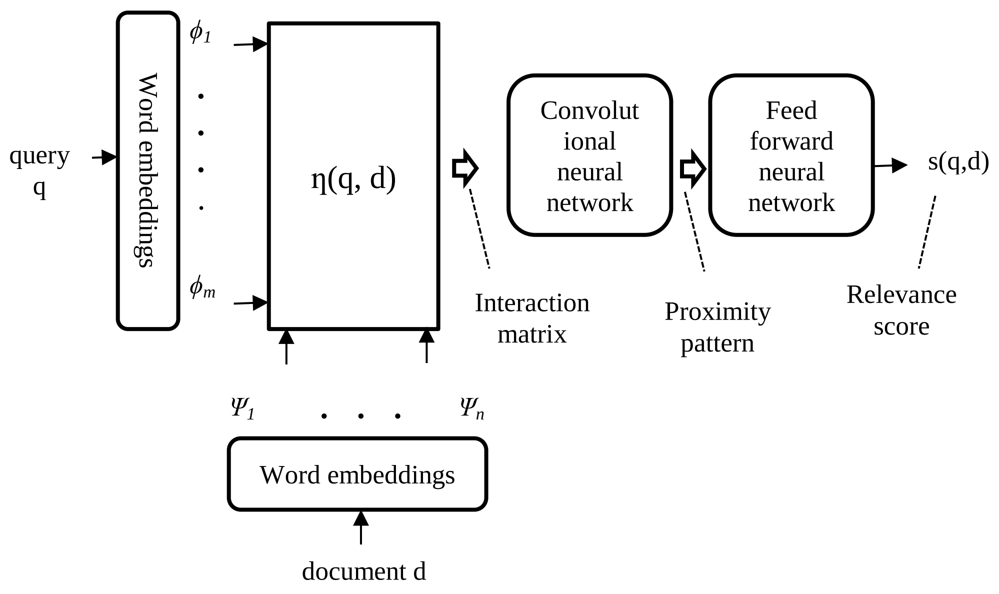

Figure 2.4 Scheme of an interaction-focused model based on convolutional neural networks

Tokenization is used to separate the query q and the document d into m and n tokens, respectively. Each token is associated with a static word embedding. The cosine similarity between a query token embedding and a document token embedding makes up the interaction matrix $\eta(q, d) \in Rm \times n$.

The Deep Relevance Matching Model (DRMM) as one of the earliest neural models that makes use of the interaction matrix [21]. Using hard bucketing, the cosine similarity of each query token with respect to the document tokens in DRMM are transformed into a discrete distribution, or a query token histogram. The final query token-document relevance score is then calculated by feeding the histogram of each query token into a feed forward neural network.

Then, an IDF-based weighted sum across the various query phrases is used to aggregate these relevance scores. The Kernel-based Neural Ranking Model (KNRM) [82] suggests using Gaussian kernels to softly bucket histograms to feed forward neural networks, distributing each cosine similarity's contribution across buckets in a smooth manner instead of using hard bucketing.

While both KNRM and DRMM make use of the interaction matrix, they do not have a convolutional layer. The query and document embeddings are first individually processed using k convolutional neural networks in the Convolutional KNRM model (CONV-KNRM) [11], in order to construct uni-gram, bigram, up to k-gram embeddings. Such convolutions make it possible to construct word embeddings that simultaneously account for several closely related words. After that, between every combination of question and document n-gram embeddings, k^2 cosine similarity matrices are constructed, and KNRM is used to process these matrices. The interaction matrix is processed through many convolutional and pooling layers in the Position-Aware Convolutional Recurrent Relevant model (PACRR) [26] in order to account for word proximity. Other neural models that are similar to this one also include convolutional layers [14], [24], [27], [53], [54].

2.2.2 Pre-trained Language Models

Based on the training data used to generate the vectors, static word embeddings map words with multiple senses into an average or most common-sense representation. A word's vector remains constant regardless of the other words used in the phrase around it. The usage of a unique neural layer dubbed self-attention in combination with feed forward and linear layers, the transformer neural network is able to explicitly consider the context of arbitrary long text sequences. Sequences of input length are mapped to sequences of output length by the self-attention layer. The layer can access all n input elements (bidirectional self-attention) or just the first i input elements (causal self-attention) while calculating the i^{th} output element. The network is able to consider the relationships between several elements in the same input to a self-attention layer. A self-attention layer computes token representations that consider the surrounding words when the input elements are tokens of a specific text. By doing this, the transformer generates contextualized word embeddings, in which the input text as a whole determines how each input token is represented.

Transformers have shown effectively in a variety of natural language processing applications, including question answering, summarization, machine translation, and more. These jobs are all specific examples of a broader objective, which is to convert an input text sequence into an output text sequence. This general task has been addressed by the invention of the sequence-to-sequence model. The two components of a sequence-to-sequence neural network are an encoder model that produces a contextualized representation of each input element given an input sequence, and a decoder model that uses these contextualized representations to produce an output sequence tailored to a job. The components of both types are many stacked transformers. Bidirectional self-attention layers are used by the encoder's transformers on either the input or the output sequence from the preceding transformer. The decoder's transformers use bidirectional cross-attention on the output of the final encoder transformer and causal self-attention on the output of the preceding decoder transformer.

Two particular applications of sequence-to-sequence models have been investigated in Neural IR: encoder-only models and encoder-decoder models. All of the tokens in a particular input sentence are fed into encoder-only models, which then produce contextualized word embeddings for each token in the sentence. The models BERT [12], RoBERTa [42], and DistilBERT [69] are examples of this family of models. Depending on the input sentence, encoder-decoder models produce new output sentences. One token at a time, the decoder model sequentially accesses these embeddings to produce new output tokens, while the encoder model takes all of the tokens of a given sequence as input and creates a contextualized representation. Among these model representatives are BART [39] and T5 [63]. Sequence-to-sequence models can be trained as language models by computing the token probabilities using a softmax operation and projecting each output embedding to a specified vocabulary using a linear layer. A function $\sigma: R^k \rightarrow [0, 1]^k$, which accepts as input $k > 1$ real numbers $z_1, z_2, \dots, z_k$ and transforms each input z_i as in Equation (2.2), is known as the softmax operation.

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \qquad (2.2)$$

The input values are normalized into a probability distribution using the softmax procedure. Within the domain of deep learning, the inputs of a softmax operation are commonly referred to as logits. These logits are the unprocessed predictions produced by a multi-class classification model, which the softmax operation transforms into a probability distribution across the classes.

A sequence-to-sequence model can be trained as a Casual Language Model (CLM), like T5, or as a masked language model (MLM), like BERT, depending on the training goal. While CLM training focuses on predicting the next token in an output sequence given the previous tokens in the input sequence, MLM training teaches learners to predict missing tokens in a sequence given the surrounding tokens. To create pre-trained language models, it is customary to train these models on large amounts of text data in both situations. By doing this, it enables the model to acquire general-purpose language knowledge that may subsequently be applied to a downstream task that is more specialized. This transfer learning strategy uses an initial model that is pre-trained on a smaller, domain-specific training dataset to refine it for the downstream target job. To put it another way, fine-tuning is the process of modifying a pre-trained language model's parameters for the target task and domain data.

Figure 2.5 illustrates the basic requirements for pre-training: large-scale general-purpose training corpus (e.g., Wikipedia or Common Crawl web pages), costly computing resources, and lengthy training periods (e.g., many days or weeks). However, fine-tuning necessitates a small domain-specific corpus that is concentrated on the downstream job, reasonably priced computing resources, and a few more hours or days of training. Two specific instances of fine-tuning are zero-shot learning, in which a pre-trained language model is applied to a downstream job for which it was not fine-tuned, and few-shot learning, in which the domain-specific corpus consists of a relatively small quantity of training data.

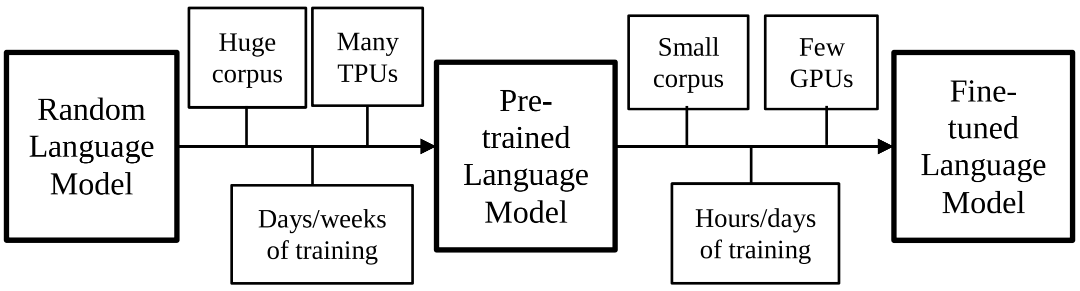

Figure 2.5 Transfer learning of a pre-trained language model to a fine-tuned language model

Cross-encoder models are interaction-focused Neural IR systems that employ pre-trained language models. They take as input a pair (q, d) of query and document strings. Different cross-encoders are fine-tuned in different ways depending on the type of sequence-to-sequence model; nevertheless, generally speaking, their goal is to compute a relevance score $s(q, d)$ to rank documents with respect to a given query. The most popular cross-encoders using both encoder-only and encoder-decoder types are shown in the following.

2.2.3 Ranking with Encoder-only Models

BERT, an encoder-only model, is the transformer architecture that is most frequently used in Neural IR. The WordPiece sub-word tokenizer [79] is used to tokenize the text input. This tokenizer's vocabulary V consists of 30, 522 terms, with the uncommon/rare words (like goldfish) divided into smaller words (like gold## and ##fish). The unique $[CLS]$ token—which stands for "classification"—is always the first input token used in BERT. Other special tokens that indicate the end of a text supplied as input or divide two distinct texts supplied as a single input are accepted as input by BERT. One such token is $[SEP]$. A maximum of 512 tokens can be entered into BERT, and for each token entered, an output embedding in R^l is produced. The most widely used variant of BERT is called BERT base, and it has an output representation space with dimensions of l = 768 and 12 transformer layers stacked. There are two slightly different approaches to fine-tune BERT as a cross-encoder1: [49] and [44].

Both texts in a query-document combination are tokenized into the token sequences $d_1, \ldots, d_n, and\ q_1, \ldots, q_n$. Following that, the tokens and BERT special tokens are concatenated to create the input configuration seen below:

$$[CLS] q_1, \ldots, q_m\ [SEP]\ d_1, \ldots, d_n\ [SEP]$$

This will be the input for the BERT. By doing this, the BERT encoders' self-attention layers are able to consider the semantic relationships between the question and document tokens. For the input $[CLS]$ token, the output embedding $\eta_{[CLS]} \in R^l$ provides a contextual representation of the query-document pair as a whole.

To calculate the query-document relevance score, [50] fine-tune BERT on a binary classification job, as shown in Figure 2.6. In order to obtain the relevance score $s(q, d)$, BERT processes the query and the document to produce the output embedding $\eta_{[CLS]} \in R^l$. This embedding is then multiplied by a learned set of classification weights $W_2 \in R^{2*l}$ to yield two real scores, z_0 and z_1, which are subsequently transformed into a probability distribution p_0 and p_1 over the relevant and non-relevant classes via a softmax operation. The final relevance score is the probability corresponding to the relevant class, which is often assigned to label 1, or p_1. Through the learnt matrix $W_1 \in R^{1*l}$, [44] fine-tune BERT by projecting the output embedding $\eta_{[CLS]} \in R^l$ into a single real value z, which represents the final relevance score as in Equations (2.3) – (2.6).

$$\eta_{[CLS]} = BERT\ (q, d) \tag{2.3}$$

$$[z_0,\ z_1] = W_2 \eta_{[CLS]}\ or\ \ z = W_1 \eta_{[CLS]} \tag{2.4}$$

$$[p_0,\ p_1] = softmax\ [z_0,\ z_1] \tag{2.5}$$

$$s(q, d) = p_1\ \ or\ s(q, d) = z \tag{2.6}$$

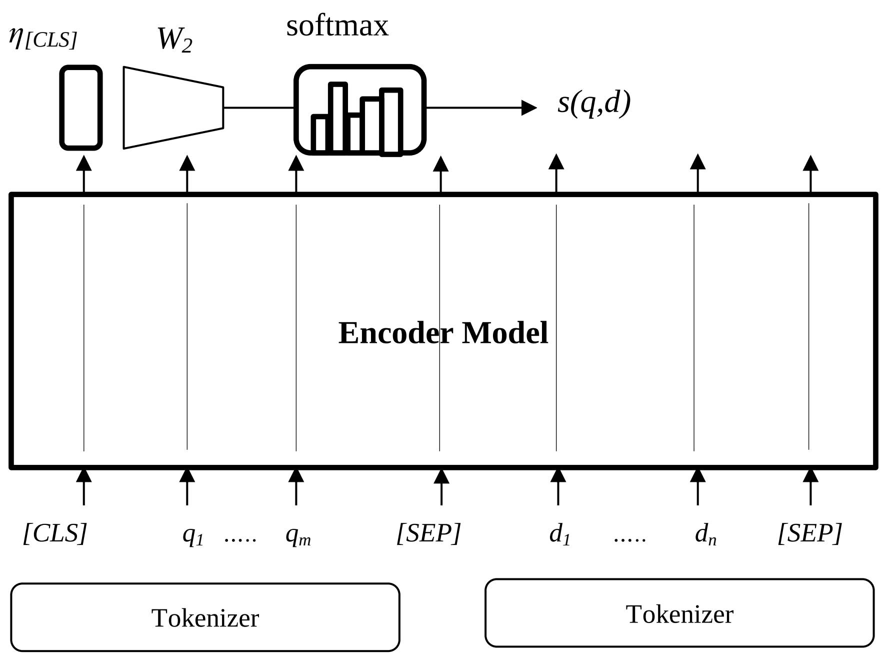

Figure 2.6 BERT classification model for ad-hoc ranking

2.2.4 Ranking with Encoder-decoder Models

It is possible to use an encoder-decoder model [63] with prompt learning in place of an encoder-only transformer model to compute the latent representation of a query-document pair and convert it into a relevance score. This is achieved by converting the relevance score computation task into a cloze test, or fill-in-the-blank problem. In tasks involving the summarization of articles [62] and the building of knowledge bases [58], prompting has been effectively implemented. In quick learning, the downstream goal is recast as a cloze-like problem while the input texts are reformed as a natural language template. For subject classification, for instance, if the phrase text needs to be divided into two classes, c_0 and c_1,, the input template could be:

$$Input : text\ Class : [OUT]$$

Two label terms, w_0 and w_1, have been chosen from the lexicon to represent the classes c_0 and c_1, respectively. It is possible to convert the likelihood that the input token $[OUT]$ will be allocated to the appropriate label token from the likelihood of assigning the input $text$ to a class:

$$p(c_0|text) = p([OUT] = \ w_0|Input: \ text\ Class: [OUT]) \tag{2.7}$$
$$p(c_1|text) = p([OUT] = \ w_1|Input: \ text\ Class: [OUT]) \tag{2.8}$$

As shown in Figure 2.7, a rapid learning strategy for relevance ranking utilizing a T5 model [51]. The following input template is created by concatenating the query with the document texts q and d:

$$Query: q\ Document: d\ Relevance: \ [OUT]$$

A downstream job that takes this input configuration as input and produces an output sequence with the last token equal to True or False, depending on whether the document d is relevant or irrelevant to the query q, is used to fine-tune an encoder-decoder model. The calculation of the query-document relevance score involves applying a softmax operation to normalize solely the *False* and *True* output probabilities, which are calculated throughout the entire vocabulary.

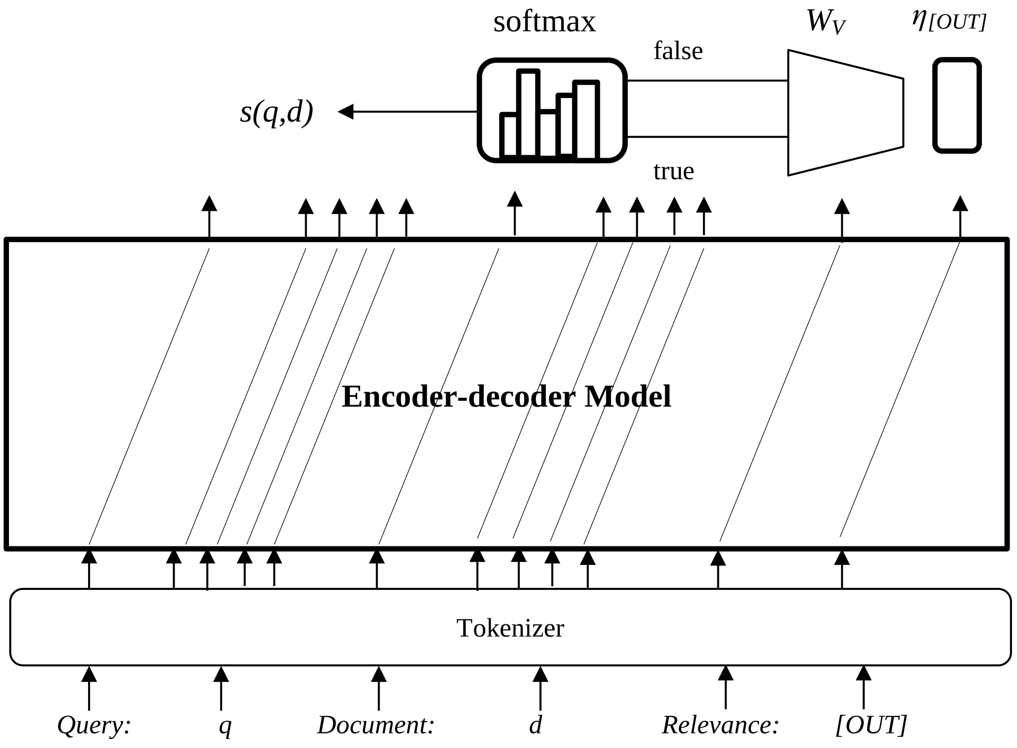

Figure 2.7 T5 model for ad-hoc ranking

The query and the text are processed by T5 to create the output embedding $\eta_{[OUT]} \in R^{l}$, which is projected over the vocabulary V by a learnt set of classification weights $W_V \in R^{|V|*l}$ in order to yield the relevance score $s(q,d)$. In order to get the necessary predictions p_{False} and p_{True} over the "non-relevant" and "relevant" classes, the outputs z_{False} and z_{True}, which correspond to the $False$ and $True$ terms, respectively, are converted into a probability distribution using a softmax operation.

The ultimate relevance score is the prediction, or p_{True}, that corresponds to the relevant class.

$$\eta_{[OUT]} = T5(q,d) \tag{2.9}$$

$$[\dots, z_{False}, \dots, z_{True}, \dots]^T = W_V\, \eta_{[OUT]} \tag{2.10}$$

$$[p_{False}, p_{True}] = softmax([z_{False}, z_{True}]) \tag{2.11}$$

$$s(q,d) = p_{True} \tag{2.12}$$

2.2.5 Fine-tuning Interaction-focused Systems

The pre-trained language models used in IR need to be adjusted for a particular downstream job. A Neural IR model $M(\theta)$, parametrized by θ, evaluates an input query-document pair (q,d) and returns the score of document d with respect to the query q, or $s_\theta(q,d) \in R$. Predicting $y \in \{+,-\}$ from $(q,d) \in Q \times D$ is a task, with denoting non-relevant and $+$ denoting relevant. One way to describe this issue is as a binary classification issue. Assuming a joint distribution p over $\{+,-\} \times Q \times D$, it carry out the classification by selecting appropriate pairs $(+,q,d) \equiv (q,d^{+})$ and $(-,q,d) \equiv (q,d^{-})$. As an example of a metric learning issue [80], it learns a score function $s_\theta(q,d)$ using sampled correct pairs. The score function must provide a high score to a relevant document and a low score to a non-relevant document, as in Equations (2.3) - (2.6) and (2.9) - (2.12). Afterwards, it identifies θ^* such that it minimizes the (binary) cross entropy l_{CE} between the model probability $p_\theta(y|q,d)$ and the conditional probability $p(y|q,d)$:

$$\theta^* = \arg min_\theta\ E[l_{CE}(y,q,d)] \tag{2.13}$$

where the cross entropy is determined as, and the expectation is calculated over $(y|q,d)$ ~ p as in Equations.

$$l_{CE}(y,q,d) = \begin{cases} -\log(s_\theta(q,d)) & if\ y = + \\ -\log(1 - s_\theta(q,d)) & if\ y = - \end{cases} \qquad (2.14)$$

A list of triples $(q, d^+,\ d^-)$ with q representing a query, d^+ representing a relevant document for the query, and d^- representing a non-relevant document for the query typically makes up a dataset T that may be used to fine-tune pre-trained language models for relevance scoring as in Equation (2.15). In this instance, the cross entropies calculated for each triple add up to the predicted cross entropy:

$$E[l_{CE}(y,q,d)] \approx \frac{1}{2|T|} \sum_{(q,d^+,d^-)\,\in\,T} (-\log(s_\theta(q,d)) - \log(1 - s_\theta(q,d))) \qquad (2.15)$$

This method is restricted to considering positive and negative triples independently of each other pairwise. An alternative fine-tuning strategy is frequently applied to representation-focused systems, which accounts for several irrelevant documents for every important document.

2.3 Representation-focused Systems

Document representations can be pre-computed and stored that representation-focused systems construct independent query and document representations. Only the query representation is computed during query processing; the stored document representations are searched to get the top documents. By doing this, representation-based systems which belong to a new class of retrieval techniques known as dense retrieval systems are able to find the pertinent documents among all the documents in a collection as opposed to simply a query-dependent sample. In dense retrieval, two distinct families of representations have so far surfaced.

2.3.1 Single Representations

Because every word in the document can attend to every word in the query and vice versa, interaction-focused systems concatenate the query and document texts before processing them with sequence-to-sequence models. This results in rich interactions between the query context and the document context. Every document must be concatenated with the query and processed through a forward pass of the entire sequence-to-sequence model during the query processing phase. Despite the proposal of certain strategies, such as pre-computation of internal representations [45], interaction-focused systems are not scalable to handle a high volume of documents. A standard CPU can process a query over the entire document collection using an inverted index in a matter of milliseconds [74], whereas [44] found that calculating the relevance score of a single query-document pair using a transformer model may take several seconds.

As shown in Figure 2.8, representation-focused systems use encoder-only models to independently compute query representations $\phi(q)$ and document representations $\Psi(d)$ in the same latent vector space, as opposed to using sequence-to-sequence models to compute a semantically richer but computationally expensive interaction representation $\eta(q, d)$ [76]. Next, using an aggregation function f between these representations, the relevance score between them is calculated:

$$\phi(q) = \phi_{[CLS]}, \phi_1, \dots, \phi_{|q|}] = Encoder(q) \qquad (2.16)$$

$$\Psi(d) = \Psi_{[CLS]}, \Psi_1, \dots, \Psi_{|d|}] = Encoder(d) \qquad (2.17)$$

$$s(q, d) = f(\phi(q), \Psi(d)) \qquad (2.18)$$

The representations functions ϕ and Ψ in neural inference are calculated using refined encoder-only sequence-to-sequence models, such BERT.

Since the question and document representations are computed using the same neural model, this model is also referred to as a dual encoder or bi-encoder [3]. A bi-encoder is a mapping that creates mathematical manipulates representations of queries and documents in the same vector space R^l. Typically, one assumes that the output embedding that corresponds to the $[CLS]$ token represents a specific input text as in Equations. The dot product serves as the score aggregation function when using these single representations:

$$s(q, d) = \phi_{[CLS]}, \Psi_{[CLS]} \qquad (2.19)$$

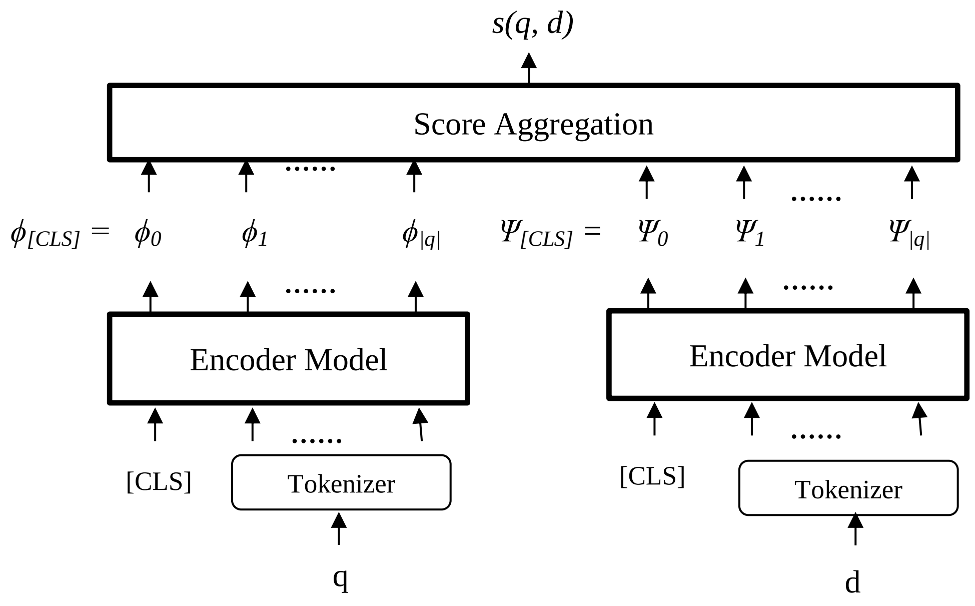

Figure 2.8 Representation-focused system

A variety of single-representation systems have been proposed; the most generally used ones are DPR (Dense Passage Retrieval) [32], [81], and [83].

2.3.2 Multiple Representations

Previously, this attention has been drawn to representation-focused systems, wherein documents and queries are represented by a single embedding into the latent vector space. It is believed that this one representation has all of the meaning text within that one embedding. On the other hand, multiple representation systems, like COIL [15], [36], [43], and [28], use more than one embedding to represent a given text, potentially enabling a richer semantic representation of the content.

Poly-encoders [28] use the first m output embeddings $\Psi_0, \Psi_1, \ldots, \Psi_{m-1}$ to encode a document d rather than simply the first output embedding $\Psi_{[CLS]} = \Psi_0$. While it needs to aggregate the m output document embeddings into a single representation Ψ_* in order to compute the final relevance score using the dot product with the output query embedding, a query q is still represented by a single embedding, $\phi_{[CLS]} = \phi_0$. To do this, poly-encoders use the dot product to first calculate the m similarity $s_0, \ldots, s_{m-1}$ between the query embedding and the first m document embedding.

Using a softmax operation, these similarities are converted into normalized weights $v_0, \dots, v_{m-1}$, and the weighted output embeddings are added up to determine the final document embedding Ψ_* that is employed.

$$[\phi_0, \phi_1, \dots] = Encoder(q) \tag{2.20}$$

$$[\Psi_0, \Psi_1, \dots] = Encoder(d) \tag{2.21}$$

$$[s_0, s_1, \dots, s_{m-1}] = \phi_0.\Psi_0, \phi_0.\Psi_1, \dots, \phi_0.\Psi_{m-1} \tag{2.22}$$

$$[v_0, v_1, \dots, v_{m-1}] = softmax([s_0, s_1, \dots, s_{m-1}]) \tag{2.23}$$

$$\Psi_* = \sum_{i=1}^{m-1} v_i \, \Psi_i \tag{2.24}$$

$$s(q, d) = \phi_0.\Psi_* \tag{2.25}$$

Similarly to poly-encoders, ME-BERT [43] exploits the first m output embeddings to represent a document d (including the $[CLS]$ embedding), but uses a different strategy to compute the relevance score $s(q, d)$ a query q. ME-BERT computes the similarity between the query embedding and the first m document embedding using the dot product, and the maximum similarity, also called maximum inner product, represents the relevance score:

$$s(q, d) = max_{i=0,\dots,m-1} \, \phi_0.\Psi_i \tag{2.26}$$

On the contrary, the relevance scoring function in Equations (2.20) - (2.25), based on a softmax operation does not permit to decompose the relevance scoring to a maximum computation over dot products. ColBERT [36] does not impose a maximum on the number of embeddings that can be utilized to represent a document, in contrast to poly-encoders and ME-BERT. Rather, it represents a document using all of the $1 + |d|$ output embeddings, i.e., one output embedding for each document token, including the $[CLS]$ special token. Additionally, a query q is represented by numerous $1 + |q|$ output embeddings, meaning that each query token, including the $[CLS]$ special token, has an output embedding. To provide "a soft, differentiable mechanism for learning to expand queries with new terms or to re-weigh existing terms based on their importance for matching the query," queries may also be augmented with additional masked tokens, as in other representation-focused systems [36]. Currently, up to 32 query token embeddings are included to queries.

A learnt weight matrix $W \in R^{l'*l}$, with $l' < l$, can project query and document embeddings in a smaller latent vector space without sacrificing generality. ColBERT takes advantage of a modified form of the relevance scoring function in Equations (2.26), where each query embedding adds to the final relevance score by summing, because there are numerous query embeddings.

$$s(q,d) = \sum_{i=0}^{|q|} max_{j=0,\dots,|d|} \phi_i \,.\, \Psi_j \qquad (2.27)$$

In Equation (2.27), also known as sum maxim, ColBERT late interaction scoring carries out an all-to-all computation: every query embedding, including the embeddings of the masked tokens is dot-multiplied with every document embedding, and the maximum computed dot products for each query embedding are then summed up. By matching a different lexical word to the maximum extent possible, a query term might so contribute to the final scoring. The COIL system suggests an alternative method [15]. A learnt matrix $W_C \in R^{l*l}$ is used in COIL to linearly project the query and document $[CLS]$ embeddings. Using another learnt matrix $W_T \in R^{l'*l}$, the embeddings corresponding to normal query and document tokens are projected into a smaller vector space with dimension $l' < l$. Values for ~ 0 typically fall between 8 and 32.

The total of the two elements is the query-document relevance score. The projected query and document $[CLS]$ embeddings are multiplied to create the first component, and the sum of the sub-components, one for each query token, is the second component. The maximal inner product between a query token and the document embeddings for the same token is what makes up each sub component as in Equations:

$$[\phi'_0, \phi'_1, \dots] = [W_C\phi_0, W_T\phi_1, \dots] \qquad (2.28)$$

$$[\Psi'_0, \Psi'_1, \dots] = [W_C\Psi_0, W_T\Psi_1, \dots] \qquad (2.29)$$

$$s(q,d) = \phi'_0 . \Psi'_0 + \sum_{t_i \in q} max_{t_j \in d,\, t_i = t_j} \;\phi'_i .\, \Psi'_j \qquad (2.30)$$

It can pre-compute the projected document embeddings and, for each term in the vocabulary, concatenate the embeddings in the same document and in the entire collection to the COIL's scoring function, which is based on lexical matching between query and document tokens. These embeddings are then arranged in posting lists of embeddings, with a special posting list reserved for the $[CLS]$ token and its document embeddings. This organization uses optimized linear algebra libraries, like BLAS, to handle posting lists efficiently at query time [2]. It should be noted that computation of the projected query embeddings occurs during query processing.

2.3.3 Fine-tuning Representation-focused Systems

Fine-tuning of a bi-encoder is equivalent to learning an inner-product function that is suited for relevance scoring in the ad-hoc ranking challenge. Neural IR model $M(\theta)$, which is parametrized by θ, calculates a score $s_\theta(q,d)$ for a document d in relation to a query q. It now formulates the learning task as an estimation problem using probabilities. In order to achieve this, it applies a softmax operation to transform the scoring function into a suitable conditional distribution as in Equation:

$$p_\theta(d|q) = \frac{\exp(s_\theta(q,d))}{\sum_{d' \in D} \exp(s_\theta(q,d'))} \tag{2.31}$$

where the posterior probability of the document being relevant in light of the query is represented by $p_\theta(d|q)$. It aims to determine the parameters θ^* that minimize the cross entropy I_{CE} between the actual probability $p(d|q)$ and the model probability $p_\theta(d|q)$., assuming that it have a joint distribution p over $D \times \mathrm{Q}$ as in Equation:

$$\theta^* = \arg min_\theta \ E[I_{CE}(d,q)] = \arg min_\theta \ E[-\log(p_0(d|q))] \tag{2.32}$$

where $(d,q) \sim p$ is the computation of the expectation. If the scoring function $s_\theta(q,d)$ is sufficiently expressive, then $p(d|q) = p_\theta(d|q)$ for a given θ. Due to the enormous number of documents in D, it is challenging to optimize the cross entropy loss and computing the denominator, sometimes referred to as the partition function.

In noise contrastive estimation, it maximizes the likelihood of $p_\theta(d|q)$ contrasting $g(d)$ by selecting an artificial noise distribution g over D of negative samples. It defines the conditional distribution for each of the $k \geq 2$ documents $D_k = d_1, \ldots, d_k$ as in Equation.

$$\hat{p}_\theta(d_i|q, D_k) = \frac{\exp(s_\theta(q,d_i))}{\sum_{d' \in D_k} \exp(s_\theta(q,d'))} \tag{2.33}$$

This is much less expensive to calculate than Equation (2.33) if $k \ll |\mathrm{D}||$. The goal now is to identify the values of θ^+ that minimize the noise contrastive estimation loss (l_{NCE}), which is expressed as in Equation:

$$\theta^+ = \arg min_\theta \ E[l_{NCE}(D_k, q)] = \arg min_\theta \ E[-\log(\hat{p}_\theta(d_1|q, D_k))] \tag{2.34}$$

where, for $i = 2, \ldots, k$, the expectation is calculated over $(d_1, q) \sim p$ and $(d_i \sim g)$. The ultimate objective of this fine-tuning is to learn a latent vector space for query and document representations [32] where a query and its relevant document(s) are closer than the query and its non-relevant documents, with respect to the dot product [25]. This fine-tuning approach is also known as contrastive learning. The noise distributions g over D yields negative samples. Its outline a few negative sampling techniques used in Neural IR below.

Random sampling means $q(d) = \frac{1}{|D|}$, or any random document from the corpus is regarded as non-relevant with equal probability. One can sample an infinite number of negative documents. It makes intuitive sense to anticipate that a document selected at random will have a relevance score that is significantly lower than the relevance score of a positive document, with a loss value that is near to zero. The training convergence to determine the parameters θ^+ is not significantly impacted by negative documents with almost minimal loss [29] and [33].

In-batch sampling means in order to speed up training, the queries used to calculate the loss can be arbitrarily combined into batches of size b. The positive passages for the other $b - 1$ inquiries are regarded as negative passages for the particular query in a specific batch [18]. Although the sampling process is faster, this sampling strategy has the same near zero loss issue as random sampling [81].

Hard negative sampling means using a traditional or trained retrieval system, negative documents can be produced. The retrieval system receives each query as input, retrieves the top documents, and treats the documents that do not match the positive results as negatives. Take note that since it presumes to know the pertinent document d_1 for the query, it is assuming a conditional noise distribution$p(d|q, d_1)$. Low-ranking papers that do not affect the user experience or cause loss are given priority over high-ranking documents in this way. The BM25 relevance model, as in DPR [32], and the neural model that is presently being trained, as in ANCE [81], can be utilized by the retrieval system that was used to mine the negative documents, or another fine-tuned neural model that has been refined, such STAR [83].

2.4 Retrieval architectures

Although they are relatively costly to compute, pre-trained language models successfully increase the efficiency of IR systems in the ad hoc rating task. The interaction-focused methods are not employed directly on the document collection, that is, to rank all documents that match a query, because of these processing costs. They are used in a pipelined architecture as shown in Figure 2.9, where a more costly neural re-ranking system, such the cross-encoders, is used after a preliminary ranking stage that retrieves a restricted number of candidates, usually 1000 documents.

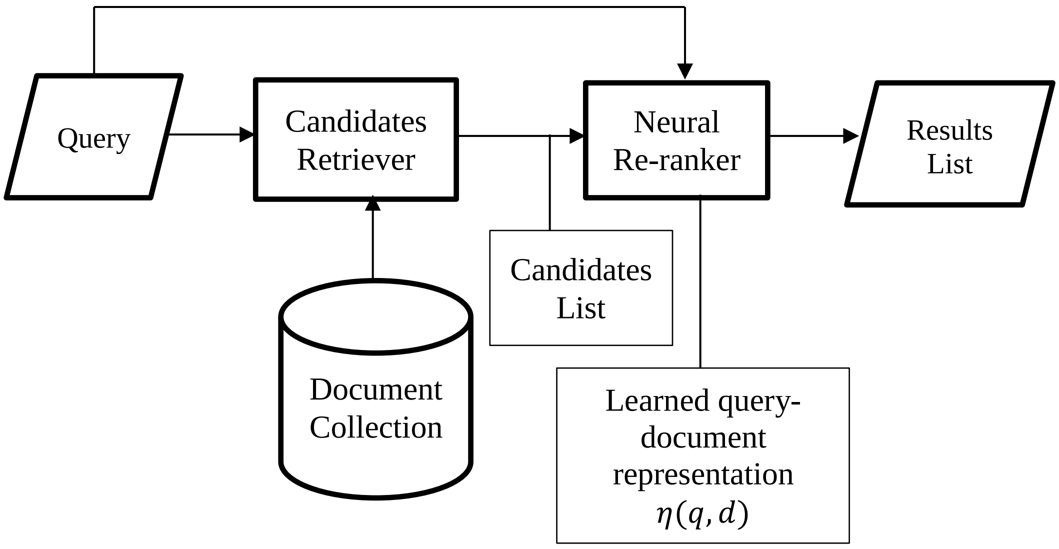

Figure 2.9 Re-ranking pipeline architecture for interaction-focused Neural IR systems

The ability to pre-compute and cache the representations of a sizable corpus of documents using the learnt document representation encoder $\psi(d)$ is the primary advantage of bi-encoders.

During the query processing phase, the user receives the top k documents whose embeddings have the largest inner product of the query embedding. The learned query representation encoder only needs to compute the query representation $\psi(q)$. After that, the documents are ranked based on this inner product as shown in Figure (2.10).

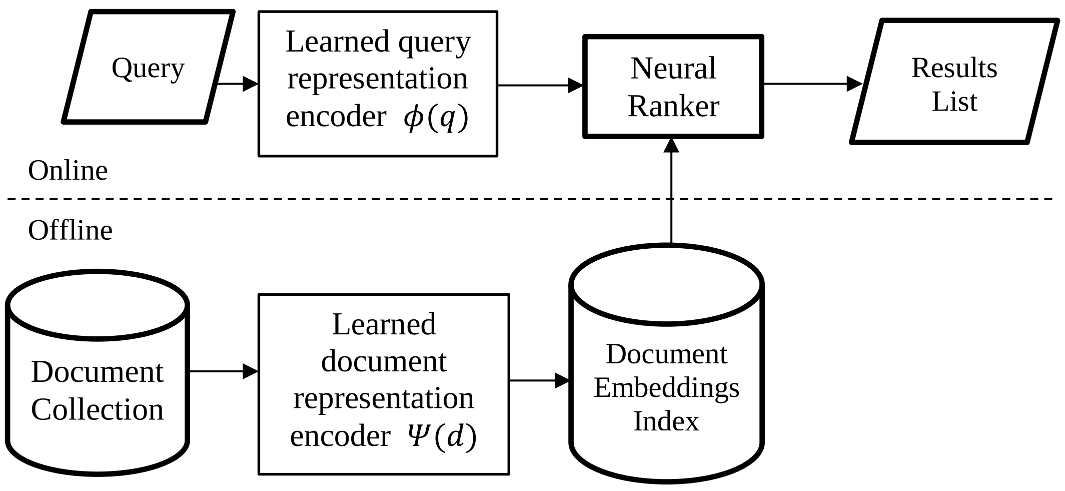
Query
Learned query representation encoder $\phi(q)$
Neural Ranker
Results List
Online
Offline
Document Collection
Learned document representation encoder $\Psi(d)$
Document Embeddings Index

Figure 2.10 Dense Retrieval Architecture for Representation-focused Neural IR systems

2.5 Application Areas of IR and Neural IR

IR has a broad range of applications across various domains, leveraging the power of neural networks to enhance the retrieval of relevant information. The following are the example applied areas from some of them. Fine-tuning models to improve the ranking of documents based on relevance to a query. Adapt models to retrieve accurate answers from a corpus in response to user queries. Enhancing recommendation algorithms by fine-tuning models to understand user preferences and provide relevant suggestions.

Neural IR has a broad range of applications across various domains, leveraging the power of neural networks to enhance the retrieval of relevant information. The following are the example applied areas from some of them.

Neural IR models like BERT and DPR enhance the relevance of search results by better understanding user queries and document content in web search engines improved search relevance. Neural models help in expanding and reformulating user queries to improve retrieval performance in web search engine as query expansion and reformulation.

QA systems are open-domain QA, and conversational agents. Open-domain QA systems like DPR are used to retrieve relevant passages from large corpora to answer user queries. Conversational agents mean virtual assistants and chat-bots use Neural IR to fetch relevant information and provide accurate responses to user questions.

E-commerce are includes the product search, and recommendation systems. Neural IR enhances product search by understanding user queries and matching them with product descriptions, reviews, and specifications in product search. Recommendation systems combine IR with recommendation algorithms to provide personalized product recommendations based on user behavior and preferences.

Healthcare includes the clinical decision support, and patient query systems. Neural IR helps healthcare professionals retrieve relevant medical literature, clinical guidelines, and patient records are patient query systems assisting patients in finding relevant health information and resourced by understanding natural language queries.

Legal IR is case law and legal documents, and e-discovery. Neural IR aids in retrieving relevant case laws, statutes, and legal documents based on complex legal queries in case law and legal documents and E-discovery enhances the efficiency and accuracy of the electronic discovery process by retrieving relevant documents from large datasets.

Academic Research and Digital Libraries are research paper retrieval, and metadata extraction. Research paper retrieval assists researchers in finding relevant academic papers, articles, and citations, but Metadata extraction extracting and retrieving metadata information from academic articles are for better organization and search ability.

Social Media and Content Moderation is content retrieval, and content moderation. Content retrieval helps finding relevant posts, comments, and user-generated content based on specific queries or interests. Content moderation **u**ses Neural IR to identify and retrieve harmful or inappropriate content for review and moderation.

Customer support is including the automated customer service, and help desk systems. Automated customer service enhances automated customer support systems by retrieving relevant information from knowledge based on answer customer queries. Desk systems assisting support agents in finding relevant solutions and documentation to resolve customer issues quickly.

Multimedia retrieval is including the image and video search, and content-based retrieval. Neural IR models are used to retrieve relevant images and videos based on textual or visual queries. Content-based Retrieval leveraging neural networks to match multimedia content based on content features rather than just metadata.

Enterprise Search is internal document retrieval, and knowledge management. Internal document retrieval helps employees retrieve relevant documents, reports, and internal communications from large corporate databases. Knowledge management enhances knowledge management systems by retrieving relevant information and documents for decision-making and operational efficiency.

Personal Assistants and Smart Devices are voice search and command, and contextual awareness. Voice search and command enhance voice-activated search and command functionalities by understanding natural language queries and retrieving relevant information. Contextual awareness using Neural IR to provide contextually relevant information based on user interactions and preferences.

Media and Entertainment are content recommendation, and semantic search. Content recommendation enhances content recommendation systems for movies, music, and articles based on user preferences and behavior. Semantic search enables semantic search capabilities for large media databases to find relevant content based on complex queries.

The applications of Neural IR are vast and continuously expanding as neural network models become more sophisticated and capable. These applications significantly enhance the efficiency, accuracy, and relevance of IR across various domains, leading to improve user experiences and operational efficiencies.

2.6 Overview of Deep Neural Networks

A Deep Neural Network (DNN) is a type of Artificial Neural Network (ANN) with multiple layers between the input and output layers. These networks are capable of learning complex patterns and representations in data through hierarchical processing. DNNs represent a significant advancement in artificial intelligence, enabling machines to perform complex tasks with high accuracy. Their development and applications continue to expand, driven by both academic research and industry innovations.

2.6.1 Applications of DNNs in IR

DNNs can learn complex ranking functions that score documents based on their relevance to a query. Models like RankNet, LambdaRank and LambdaMART, are designed for document ranking.

By understanding the context and semantics of a query, DNNs can suggest additional relevant terms to improve retrieval performance in query expansion.

DNNs can match queries with documents at a semantic level rather than relying solely on keyword matching. This is particularly useful for understanding synonyms and contextually similar phrases in semantic matching.

DNNs can model user preferences and search behaviors to provide personalized search results. This involves learning user profiles and adapting the retrieval process accordingly personalization.

DL models can understand and generate natural language, making them suitable for building systems that answer user queries directly instead of retrieving a list of documents in QA systems.

2.7 Overview of Pre-trained Model

To get pre-trained language models, it is standard practice to train these models on large amounts of textual data in both scenarios. By doing this, it enables the model to acquire general-purpose language knowledge that may subsequently be applied to a downstream task that is more specialized. This transfers learning strategy uses an initial model that is already learned to refine it on a smaller, domain-specific training dataset for the downstream target job.

Another way is fine-tuning that is the process of modifying a pre-trained language model's parameters for the target task and domain data. Pre-training language models typically requires a huge general-purpose training corpus, such as Wikipedia or Common Crawl web pages, expensive computation resources and long training times, spanning several days or weeks. On the other side, fine-tuning models requires a small domain-specific corpus focused on the downstream task, affordable computational resources and few hours or days of additional training. Special cases of fine-tuning are few-shot learning, where the domain-specific corpus is composed of a very limited number of training data, and zero-shot learning, where a pre-trained language model is used on a downstream task that it was not fine-tuned on. Cross-encoder models are interaction-focused Neural IR systems that employ pre-trained language models. They take as input a pair *(q, d)* of query and document strings. Different cross-encoders are fine-tuned differently depending on the type of sequence-to-sequence model; nonetheless, they all strive to rank documents according to a given query by computing a relevance score *s(q, d)*.

2.7.1 Overview of Fine-tuned Model

Fine-tuning begins with an existing model that has already been trained on a large, diverse data set to gain a wide range of characteristics and patterns. During this initial training, the pre-trained model learns to generalize by finding underlying patterns and characteristics in the training data. Over time, the model gains the ability to effectively comprehend new data. The process of fine-tuning a machine learning model involves pre-trained it and then retraining it on a more focused, smaller collection of data. The goal of fine-tuning is to preserve a pre-trained model's initial capabilities while modifying it to fit more specific use cases. Machine learning developers can more quickly and effectively design models for particular use cases by fine-tuning an already complex model. When there is a shortage of pertinent data or computational resources, this strategy is quite helpful.

On the tasks for which it was fine-tuned, a fine-tuned model's performance can outperform the initial pre-trained model. On the other hand, fine-tuning refers to methods for training a model again after its weights have already been adjusted by previous training.

By training the underlying model on a smaller, task-specific dataset, fine-tuning adapts the model by starting with its prior knowledge. Although it was theoretically possible to train a huge model from scratch on a small dataset, doing so runs the danger of overfitting, the model may learn to perform well on the training instances but may not translate well to new data. This would negate the benefit of model training and make the model inappropriate for the task at hand.

Fine-tuning offers the benefits of both worlds: refining the model's comprehension of more particular, detailed concepts while utilizing the wide knowledge and stability obtained via pre-training on a large set of data. Owing to the growing effectiveness of open source foundation models, pre-training can frequently be benefited from without incurring any additional costs or difficulties with calculation or logistics. When fine-tuning, a pre-trained model's weights are used as a basis for additional training on a smaller dataset of instances that more closely match the particular tasks and use cases the model will be applied to. Although supervised learning is usually involved, it can also involve semi-supervised, self-supervised, or reinforcement learning. When the situation necessitates supervised learning but there are few appropriate labeled instances, semi-supervised learning a type of machine learning that combines both labeled and unlabeled data is beneficial. For NLP tasks, semi-supervised fine-tuning has demonstrated promising results and eases the difficulty of obtaining a sufficient amount of labeled data. The weights of the entire network can be updated by fine-tuning; however this is not usually the case due to practical considerations. When a company uses generative AI for customer assistance, for instance, it might train a Large Language Model (LLM) using data from previous customer interactions, policies, and product information.

2.7.2 Large Language Model (LLM)

A crucial step in the LLM development cycle is fine-tuning, which enables the basic foundation models' linguistic capabilities to be modified for a range of applications, including coding, chat-bots, and other creative and technical fields. Using a vast corpus of unlabeled data, self-supervised learning is used to pre-train LLMs. Autoregressive language models are trained to predict the next word or words in a sequence until it is finished. Examples of these models are OpenAI's GPT (Generative Pre-trained Transformer), Google's Gemini, and Meta's Llama models.

Pre-training involves giving models a sample sentence's beginning from the training data and asking them to forecast each word in the sequence until the sample's end. The real word that follows in the original example sentence acts as the ground truth for each forecast. Although this pre-training produces strong text production capabilities, it does not produce a true grasp of the intent of the user. Fundamentally, autoregressive LLMs merely append text to a prompt rather than responding to it.

A pre-trained LLM (that has not been refined) only predicts, in a grammatically coherent manner, what might be the next word(s) in a given sequence that is launched by the prompt, without particularly explicit direction in the form of prompt engineering. In response to the question, "Teach me how to make a resume," and then LLM would say, "using Microsoft Word." Although it is an acceptable approach to finish the sentence, it does not support the user's objective. The model may already possess a great deal of resume writing expertise derived from pertinent information. The fine-tuning process thus serves a crucial role in not only tailoring foundation models for business’s unique tone and use cases, but in making them altogether suitable for practical usage.

The model's core knowledge can be expanded or customized, and it can be made to work in whole new jobs and domains by fine-tuning. Models can be adjusted to better represent the intended tone of a business. This might involve anything from subtle changes like starting each conversation with a kind greeting to more intricate behavioral patterns and unique visual styles. LLMs' general language skills can be refined for certain activities. For instance, Meta's Llama 2 models were made available as code-tuned (Code Llama), chatbot-tuned (Llama-2-chat), and base foundation models. Despite having undergone extensive pre-training on a vast corpus of data, LLMs lack omniscience includes domain-specific information. Legal, financial, and medical environments sometimes need the usage of specialized, esoteric language that may not have been well represented in pre-training, making it more crucial to apply extra training samples to augment the basic model's understanding in these domains. Including proprietary data can have a pipeline of confidential data that is extremely pertinent to the particular use case. This information can be fed into the model through fine-tuning, saving the need to start from scratch throughout training. Few-shot learning: Using relatively few demonstrative instances, models with good generalized knowledge may typically be fine-tuned for more specific categorization texts. Handling edge cases model respond in a particular way to circumstances that were not covered in pre-training. One good technique to make sure these kinds of events handled correctly is to fine-tune a model on labeled samples of them.

When compared to its more generic pre-trained cousin, the refined model generates more relevant and valuable replies because of this enterprise-specific training. An existing model that has already been trained on a sizable, varied data set to acquire a broad variety of features and patterns is the starting point for fine-tuning. Fine-tuning can be viewed as a subset of the more general transfer learning technique, which is the process of using the prior information that an existing model has acquired to begin learning new tasks. The idea behind fine-tuning is that, in general, it is less expensive and easier to refine the capabilities of a basic model that has already been trained and has gained general knowledge relevant to the task at hand than it is to train a new model from the beginning for that particular use. This is particularly true for deep learning models with millions or even billions of parameters, such as the complex CNNs and Vision Transformers (ViTs) used for computer vision tasks like object detection, image segmentation, or LLMs that have gained popularity in the field of NLP. Fine-tuning can lower the quantity of costly processing resources and labeled data required to produce huge models customized to specialized use cases and business objectives by utilizing previous model training through transfer learning. Fine-tuning, for instance, can be used to simply change the illustration style of a pre-trained image generation model or the conversational tone of a pre-trained LLM. It can also be used to add proprietary data or specialized, domain-specific knowledge to the learnings from a model's original training dataset. As a result, fine-tuning is crucial to the practical use of ML (Machine Learning) models, facilitating more widespread access to and modification of complex models. Although fine-tuning is supposedly a method used in model training, it is a separate procedure from what is typically referred to as "training." In this context, data scientists usually refer to the latter as pre-training for clarity's sake.

The model has not "learned" anything at the start of training or, pre-training. The first step in training is to randomly initialize the model's parameters, which are the different weights and biases applied to the mathematical operations carried out at each neural network node. Training takes place iteratively in two stages: during backpropagation, an optimization algorithm typically gradient descent is used to adjust model weights across the network to reduce loss. In a forward pass, the model makes predictions for a batch of sample inputs from the training dataset, and a loss function measures the difference (or loss) between the model's predictions for each input and the “correct” answers (or ground truth). The model "learns" through these weight modifications. The procedure is carried out again over several training. Labeled data is used in conventional supervised learning, which is typically used to pre-train models for Neural IR. The knowledge imparted by these pretext tasks is helpful for tasks that follow. Usually, they use one of two strategies: Self-prediction involves hiding a portion of the initial input and giving the model the challenge of piecing it back together. For LLMs, this is the most common training method. Training models to acquire similar embeddings for related inputs and distinct embeddings for unrelated inputs is known as contrastive learning.

2.7.3 Parameter-Efficient Fine-Tuning (PEFT)

Other fine-tuning techniques that update only a subset of the model parameters are widely available and are commonly referred to as Parameter-Efficient Fine-Tuning (PEFT). PEFT approaches, which help to reduce catastrophic forgetting (the phenomena where fine-tuning results in the loss or destabilization of the model's essential information) and computing demands, typically without causing significant performance sacrifices. Because there are so many different fine-tuning techniques and variables that come with them, it is frequently necessary to go through several iterations of training strategies and setups in order to achieve optimal model performance. These iterations involve adjusting datasets and hyper-parameters, such as batch size, learning rate, and regularization terms, until a satisfactory result is reached, as determined by the metrics that are most pertinent to use case. To fine-tune, just updating the complete neural network is the most conceptually simple method. The only significant distinctions between the pre-training and complete fine-tuning procedures are the model's initial parameter state and the dataset being used. This straightforward methodology essentially looks like the pre-training process. Certain hyper-parameters model attributes that impact learning but are not themselves learnable parameters might be adjusted in relation to their specifications during pre-training to prevent destabilizing changes from the fine-tuning process. For instance, a smaller learning rate (which lowers the magnitude of each update to model weights) is less likely to result in catastrophic forgetting.

Similar to pre-training, full fine-tuning is highly computationally intensive. It is typically too expensive and impracticable for contemporary deep learning models with hundreds of millions or even billions of parameters. A variety of techniques are combined under the umbrella of PEFT to minimize the number of trainable parameters that must be modified in order to successfully tailor a sizable pre-trained model to particular downstream applications. By doing this, PEFT dramatically reduces the amount of memory and processing power required to produce a model that is successfully fine-tuned. Especially for NLP use cases, PEFT approaches have frequently been shown to be more stable than full fine-tuning methods.

2.7.3.1 Partial Fine-tuning

Partial fine-tuning techniques, also known as selective fine-tuning, are designed to lower computational demands by adjusting only the specific subset of pre-trained parameters that are the most important to the model's performance on pertinent downstream tasks. The remaining settings are guaranteed not to be altered because they are "frozen". Updating the neural network's outer layers solely is the most logical partial fine-tuning method. For example, in a CNN used for image classification, early layers typically discern edges and textures; each subsequent layer discerns progressively finer features until final classification is predicted at the outermost layer. In most model architectures, the inner layers of the model (closest to the input layer) capture only broad, generic features. In general, the pre-trained weights of the inner layers of the model will already be more beneficial for this new, related work and the fewer layers need to be updated the more similar the new task (for which the model is being fine-tuned) is to the original goal. Additional partial fine-tuning techniques include changing the model's layer-wide bias terms alone (as opposed to the node-specific weights) and "sparse" fine-tuning techniques that modify just a portion of the model's total weights.

2.7.3.2 Additive Fine-tuning

Additive approaches add additional parameters or layers to a pre-trained model, freeze the pre-trained weights, and train only those new components, as opposed to fine-tuning the existing parameters of a pre-trained model. This method preserves the model's stability by guaranteeing that the initial pre-trained weights do not change. Because there are fewer gradients and optimization stages to store, this can lengthen training durations but also drastically lower memory requirements. Training a whole model set of parameters uses 12–20 times more Graphics Processing Unit (GPU) memory than training the model weights alone [40]. The six quantization weights of frozen model can save even more memory by reducing the precision with which model parameters are represented. This is conceptually comparable to reducing the bitrate of an audio file. Prompt tuning is one area of additive approaches. From a conceptual standpoint, it is comparable to prompt engineering, which is the process of customizing "hard prompts," or human-written prompts in natural language, to direct the model toward the intended result, including defining a specific tone or offering instances that help with few-shot learning. AI-authored soft prompts, or learnable vector embeddings concatenated to the user's hard prompt, are introduced by prompt tweaking. Prompt tuning involves freezing model weights and training the soft prompt itself, as opposed to retraining the model. Prompt, effective adjustment reduces interpretability but makes it easier for models to transition between different tasks.

2.7.3.3 Adapters

An additional subset of additive fine-tuning involves injecting and training adaptor modules, which are new, task-specific layers introduced to the neural network, instead of fine-tuning any of the frozen pre-trained model weights. The original paper measured the outcomes on the BERT masked language model, and found that adapters trained only 3.6% more parameters while achieving performance comparable to full fine-tuning.

2.7.3.4 Re-parameterization

Techniques that rely on re-parameterization, such as Low Rank Adaptation (LoRA), utilize the low-rank transformation of matrices with high dimensions (such as the large matrix of trained model weights in a transformer model). To capture the underlying low-dimensional structure of model weights, these low-rank representations exclude irrelevant higher-dimensional information, resulting in a significant reduction in the number of trainable parameters. This significantly minimizes the amount of memory required to store model updates and speeds up fine-tuning. Instead of directly optimizing the matrix of model weights, LoRA inserts a matrix of updates to model weights, or delta weights, into the model and then optimizes it. The number of parameters that need to be updated is significantly decreased by representing that matrix of weight updates as two smaller (i.e., lower rank) matrices. This decreases the amount of memory required to store model changes and speeds up fine-tuning. The weights of the pre-trained models themselves stay fixed. An additional advantage of LoRA is that different task-specific LoRAs can be "swapped in" as needed to adapt the pre-trained model whose actual parameters remain unchanged to a given use case. This is because what is being optimized and stored with LoRA are not new model weights, but rather the difference (or delta) between the original pre-trained weights and fine-tuned weights. Numerous variations of LoRA have been created, including Quantized Low Rank Adaptation (QLoRA), which quantizes the transformer model before Low Rank Adaptation (LoRA), so reducing computational complexity even further.

2.8 Summary

In this chapter, overview of Neural IR system, pre-trained model, fine-tuned model and some applied areas of Neural IR are described. By reviewing the related work, it can be seen that these techniques can promote the quality of Neural IR for other languages such as English, but there is no research on Neural IR system for Myanmar Language are described in this chapter. Queries and documents are represented by a single embedding in single representations systems, multiple representations systems express queries and/or documents using several embeddings, and the fine-tuning of representation-focused systems by using noise-contrastive estimation. The primary distinction between these systems used the method based on BERT model fine-tuning.

CHAPTER 3

DEEP NEURAL RANKING MODELS

This chapter describes the theoretical backgrounds and methodologies to DNN in detail. It describes about the overviews of DNN as well as the mechanism for its development and representation. It also describes the Deep Neural Ranking Model and Fine-tuned Model.

3.1 Models of Neural IR System

Neural IR models aim to understand the relevance between a user's query and a document by learning a function that scores the similarity between vector representations. This matching score is used to rank documents based on their relevance to the query. Neural IR models are trained on labeled data, where documents are labeled as relevant or non-relevant to specific queries. These labels are used to optimize the model's parameters so that it can better predict relevance in the future. The ultimate goal of Neural IR is to rank documents in order of their likely relevance to a given query. This ranking can be used in search engines to display the most relevant results to users. The performance of Neural IR models is typically evaluated using metrics like Precision, Recall, F1-score, and MAP to assess how well they retrieve relevant documents. Neural IR models often utilize deep learning architectures like CNNs or Transformers to capture complex patterns in text data and improve the accuracy of relevance ranking. Some Neural IR models are pre-trained on large text corpora (e.g., BERT) and then fine-tuned on specific retrieval tasks to make them more effective at understanding the nuances of IR.

3.2 Deep Neural Networks (DNNs) in Information Retrieval

DNNs are a class of machine learning algorithms inspired by the human brain's neural networks. They consist of multiple layers of artificial neurons (nodes), which process input data to extract features and make predictions. In IR, DNNs are employed to improve the effectiveness and efficiency of retrieving relevant information from large datasets.

3.2.1 Key Components and Architectures

The input layer represents the raw data fed into the network. In IR, this data could be text documents, queries, user profiles, or other forms of structured and unstructured data. Hidden layers are composed of multiple neurons that perform transformations on the input data. These layers capture various levels of abstractions and features from the data. In the context of IR, hidden layers can capture semantic meanings, contextual information, and latent relationships between terms and documents. The output layer provides the final prediction or classification. In IR, this could be a relevance score, a ranking position, or a classification of documents into relevant and non-relevant categories.

3.2.2 Popular DNN Architectures in Information Retrieval

Modeling local patterns and spatial hierarchies are used in text data on CNNs. An example is CNNs which can be used for document classification, where the convolutional layers detect features such as important keywords or phrases.

RNNs and LSTM capture long-term dependencies and suitable for sequential data. Tasks like query understanding and text generation are used. An example is LSTMs which can model the sequence of words in a query and predict the relevance of documents based on the entire query context.

State-of-the-art architecture for various NLP tasks due to their ability in transformers captures contextual information through self-attention mechanisms. An example is transformers (like BERT) which can be used for query expansion, where the model understands the context and retrieves more relevant documents.

3.2.3 Advantages of Using DNNs in Information Retrieval

DNNs automatically learn features from raw data, eliminating the need for manual feature engineering as feature learning:.

DNNs are effective in processing unstructured data such as text, images, and audio, making them versatile for various IR tasks handling unstructured data.

The multiple layers in DNNs allow them to capture complex patterns and relationships in the data that traditional models might miss capturing complex relationships.

DNNs can learn complex ranking functions that score documents based on their relevance to a query. Models like RankNet, LambdaRank, and LambdaMART are designed for this purpose in document ranking.

DNN have significantly advanced the field of IR by enabling more accurate, efficient, and context-aware retrieval of information. Their ability to learn from vast amounts of data and model complex relationships has led to substantial improvements in tasks like document ranking, query expansion, and semantic matching. However, challenges such as data requirements and computational demands need to be addressed to fully leverage their potential in IR applications.

3.2.4 Training Process of DNN

Forward propagation input data is passed through the network layer by layer to generate an output. Loss function measures the difference between the predicted output and the actual output. Common loss functions are Mean Squared Error (MSE) and Cross-Entropy Loss. The error is propagated back through the network to update the weights and biases using the gradient descent algorithm in backpropagation. Methods used to minimize the loss function, including Stochastic Gradient Descent (SGD), Adaptive Moment Estimation (Adam), and Root Mean Square Propagation (RMSprop) in optimization algorithms.

3.2.5 Types of DNN

Feedforward Neural Networks (FNNs) data flows in one direction from input to output without cycles. CNNs specialized for processing grid-like data such as images. They use convolutional layers to automatically detect spatial hierarchies of features. Convolutional Layers apply filters to extract features from the input. Pooling Layers reduce the dimensionality of the data, helping to prevent overfitting. RNNs designed for sequential data. They maintain information across time steps using loops. LSTM is an advanced type of RNN that can capture long-term dependencies. Gated Recurrent Units (GRUs) is a simplified version of LSTMs. Autoencoders such as unsupervised learning models that compress and then reconstruct the input data. Variational Autoencoders (VAEs) introduce a probabilistic approach to data encoding and decoding. Generative Adversarial Networks (GANs) consist of two networks, a generator and a discriminator, that compete to improve data generation.

The key concepts of DNN are regularization, hyper-parameters, and transfer learning. Regularization techniques prevent overfitting, such as dropout and L2 regularization. Hyper-parameters settings that control the learning process, like learning rate, batch size, and number of epochs. Transfer learning using a pre-trained model on a new but related task to leverage learned features.

3.3 Deep Neural Ranking Models

Many neural ranking models have been proposed primarily to solve IR tasks. Several approaches to ranking are based on traditional machine learning algorithms using a set of hand-crafted features. Recently, researchers have leveraged deep learning models in IR. These models are trained end-to-end to extract features from the raw data for ranking tasks, so that they overcome the limitations of hand-crafted features. A variety of DL models have been proposed, and each model presents a set of neural network components to extract features that are used for ranking [75]. Developing efficient and effective retrieval models have always been at the core of IR [72]. Modern search engines use a multi-stage cascaded architecture for ranking documents in response to each query [9].

DNNs play a pivotal role in advancing the capabilities of IR systems, enabling them to deliver more accurate, relevant, and personalized results to users across various domains and applications. Ranking models are the main components of IR systems. In this research, it has applied the following ranking models: Deep Relevance Matching Model (DRMM) [21], Match-Pyramid (MP) [53], Duetl [48], Kernelized Neural Ranking Model (KNRM) [82], Position-Aware Convolutional Recurrent Relevance (PACRR) [26], Convolutional Kernelized Neural Ranking Model (CONV-KNRM) [11], MatchZoo-CONV-KNRM (MZ-CONV-KNRM) [19].

3.3.1 Deep Relevance Matching Model (DRMM)

A neural network architecture called the Deep Relevance Matching Model (DRMM) was created with IR tasks in mind. It focuses on relevance matching between documents and queries. DRMM, created by researchers at Microsoft Research Asia, attempts to increase the efficiency of retrieval systems such as search engines by better determining the relevance of documents to specific queries.

The main attributes and elements of DRMM are interaction-based approach, and matching histograms. DRMM places a strong emphasis on the interactions between query terms and document terms, in contrast to conventional approaches that process queries and documents independently. By capturing the relevance signals between every pair of query and document terms, it generates interaction matrices. Using the interaction matrices, the DRMM creates matching histograms. The distribution of term-level matches and non-matches between the queries is shown by these histograms. The DNN that analyzes the matching histograms is the central element of the deep reinforcement machine model.

The purpose of this network is to use the properties of the histogram to identify meaningful, complicated patterns. It is made up of several layers of brain units with varying degrees of matching information. The DRMM has a term gating network that gives query terms varying relevance weights as term gating network. By prioritizing more significant terms in the query, the model is able to improve overall relevance matching. To optimize the model and rank relevant articles higher than irrelevant ones, DRMM is trained using a ranking loss function as ranking loss function. This loss function is essential to ensure that the model successfully learns to discriminate between different levels of relevance. DRMM [21] is a neural model designed for document ranking. It focuses on modeling the interaction between query terms and document terms using a histogram-based approach. It is known for its effectiveness in capturing local term-matching patterns. It represents documents and queries as term frequency histograms and computes a relevance score.

$$match(T1, T2) = F(\Phi(\mathrm{T1}), \Phi(\mathrm{T2})) \tag{3.1}$$

where two texts $T1$ and $T2$, the degree of matching is typically measured as a score produced by a scoring function based on the representation of each text, where Φ is a function to map each text to a representation vector, and F is the scoring function based on the interactions between them by the Equation 3.1. Such a text matching problem is considered general since it also describes many NLP tasks. These elements work together to give DRMM a strong framework for relevance matching in IR tasks. It has been demonstrated to significantly outperform conventional retrieval models, particularly in situations where it is essential to comprehend the subtle relationships between query and document terms.

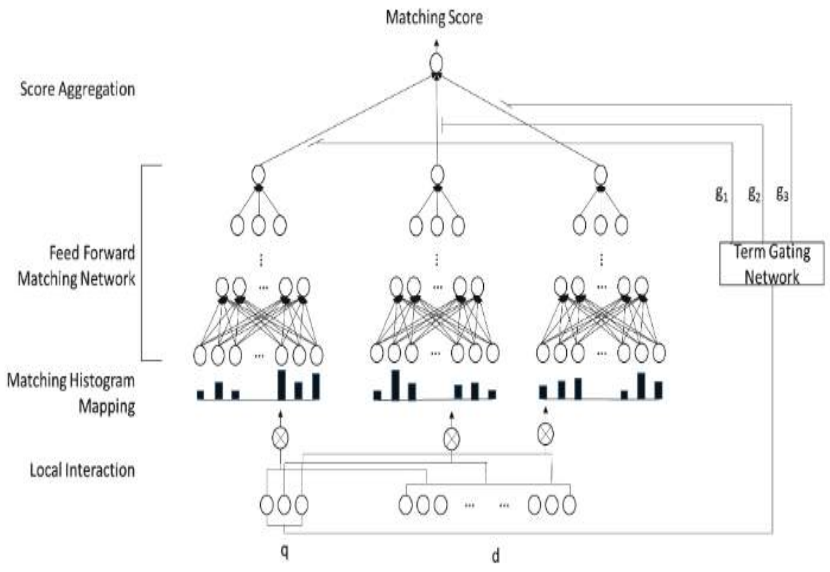

Figure 3.1: Architecture of the Deep Relevance Matching Model

As in Figure 3.1, *q* means a user's search input, typically component of a sequence of words or tokens and *d* means collection of documents. A *local interaction* captures interactions layer (eg. Convolution layer) to measure the similarity between individual terms in the query and documents. *Matching histogram mapping* constructs matching histograms based on the local interactions and captures the distribution of term overlap. *Feed-forward matching network* to process the matching histograms and generate a relevance score. *Term gating network* to assign weights to individual terms based on their relevance. *Score aggregation* process combined the weighted terms and generated the final relevance score for the query-document pair. Finally, outputs are matching score which results to relevant user queries.

3.3.2 Match-Pyramid (MP)

A deep learning architecture called MatchPyramid (MP) was created for text matching applications, including IR, QA, and paraphrase recognition. Inspired by the success of CNNs in image processing, it was proposed to capture the hierarchical matching patterns between two text sequences (e.g., a query and a document). The main elements and functionalities of MP are interaction matrix, convolutional layers, and local matching patterns. MP begins by creating an interaction matrix, just like DRMM does. The similarity scores between every word pair from the two text sequences are displayed in this matrix. This matrix can be filled with different similarity metrics, like dot product or cosine similarity. MP modifies the interaction matrix by applying convolutional layers. Local matching patterns between the two texts are captured by these layers, which function as feature extractors.

Convolutional filters are used to detect various n-gram matching patterns by swiping over the interaction matrix. Pooling layers are applied after the convolutional layers in order to extract the most prominent matching patterns and lower the dimensionality of the feature maps. Pooling facilitates the creation of a more condensed representation and improves model generalization. Fully connected (dense) layers receive the output from the pooling layers. In order to determine if the two texts match, these layers must combine the features that were extracted.

The model can learn to its fully connected layers. Hierarchical Structure by gradually implementing convolutional and pooling processes, MP is able to capture hierarchical matching patterns. Accurate text matching depends on the model's ability to learn both local and global matching information, which is made possible by its hierarchical structure. Because of the way it is built, MP works especially well for activities where it is crucial to comprehend the subtle relationships between two text passages. MP can automatically extract pertinent characteristics from the interaction matrix by utilizing CNNs, which improves performance in a variety of text matching applications.

MP [53] is a neural model that encodes both the query and document as matrices and computes their similarity through keep consistency: CNN. It is effective at capturing local and global matching patterns between queries and documents. It converts text into matrices and applies convolutions to find matching patterns.

$$M_{ij} = w_i \otimes v_j \tag{3.2}$$

Matching Matrix is a two-dimension structure where each element M_{ij} denotes the similarity between the i^{th} word w_i in the first piece of text (user query) and the j^{th} word v_j in the second piece of text (documents), where $\otimes$ stands for a general operation to obtain the similarity by the Equation 3.2.

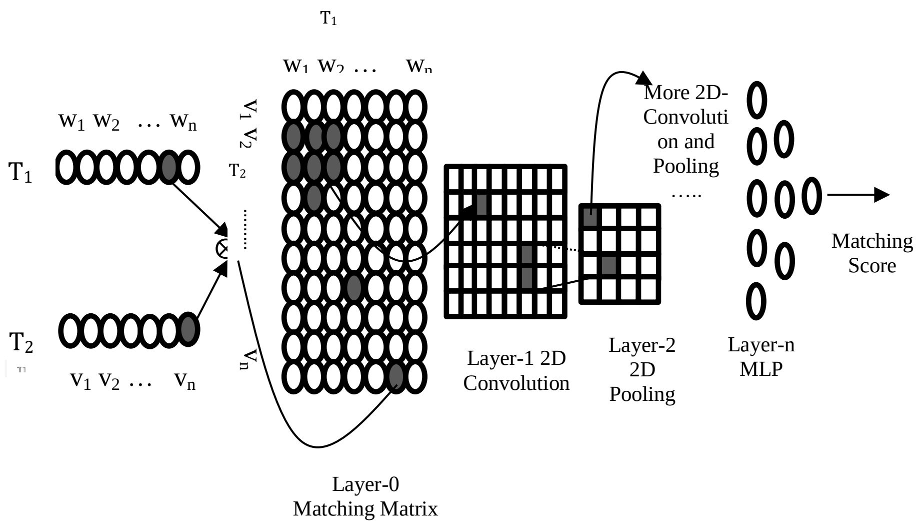

Figure 3.2: Model structure of the Match-Pyramid

As shown in Figure 3.2, T_1 and T_2 take two text sequences are composed of words or tokens as input representing user query and collection of documents. *Level-0 matching matrix* captures the local similarity information at the word level. *Level-1 2D-convolution* capture local matched patterns within a window of adjacent word pairs. *Level-2 2D-pooling* capture the most relevant information from the feature maps. *Layer-n MLP (Multilayer Perceptron)* captured higher-level semantic information and relationships. Finally, outputs are *matching score* results to relevant user query.

3.3.3 Duetl

Duet is a neural ranking model that makes use of both distributed and local representations of documents and queries in order to enhance IR. Microsoft Research's Duet leverages deep learning-based semantic matching to complement traditional term-based matching, with the goal of improving search engine and other retrieval system efficacy. The main elements and characteristics of Duet are dual representation, local representation, and distributed representation. Dual representation means Duet represents queries and documents via two distinct channels. Local Representation means using precise matches between query and document words, this channel concentrates on conventional term-based matching.

To represent local relevance signals, it makes the use of convolutional layers and interaction matrices. Distributed representation captures semantic similarities between query and document terms, it makes the use of distributed word embeddings (e.g., word2vec, GloVe). To understand semantic matching patterns, it makes the use of dense layers.

MP and DRMM-like interaction matrix is built by Duet in the local representation channel. This matrix captures local matching patterns by displaying the precise term matches found in the document and the query. To extract features that capture signals of local importance, the local representation channel applies convolutional and pooling layers to the interaction matrix. These layers aid in locating significant patterns of query and document phrase matching.

The word embeddings of the query and document terms are processed by the dense layers of the distributed representation channel. As a result, high-level interactions and semantic similarities between the query and the content can be captured by the model. To create a single representation, the outputs from the distributed and local representation channels are merged. By combining the benefits of term-based and semantic matching, this approach offers a thorough comprehension of the relationship between the query and the document.

Ranking loss function to optimize the model and rank pertinent articles are higher than irrelevant ones, Duet is trained using a ranking loss function. This loss function makes sure the model which effectively learns to discriminate between different levels of relevance. In IR problems, Duet offers a strong foundation for relevance matching by merging local and distributed representations. It has been demonstrated to significantly outperform conventional retrieval methods, particularly in situations where determining the relevance between queries and documents depends on both exact term matches and semantic similarities.

Duetl [48] is a novel document ranking model composed of two separate deep neural networks, one that matches the query and the document using a local representation, and another that matches the query and the document using learned distributed representations. The two networks are jointly trained as part of a single neural network. This combination or ‘duet’ performs significantly better than either neural network individually on a Web page ranking task, and also significantly outperforms traditional baselines and other recently proposed models based on neural networks.

$$f(Q,D) = f_l(Q,D) + f_d(Q,D) \tag{3.3}$$

where both the query and the document are considered as ordered list of terms, $Q = [q_1, \ldots, q_{nq}]$ and $D = [d_1, \ldots, d_{nd}]$. Each query term q and document term d is an m × 1 vector where m is the input representation of the text (e.g. the number of terms in the vocabulary for the local model) by the Equation 3.3.

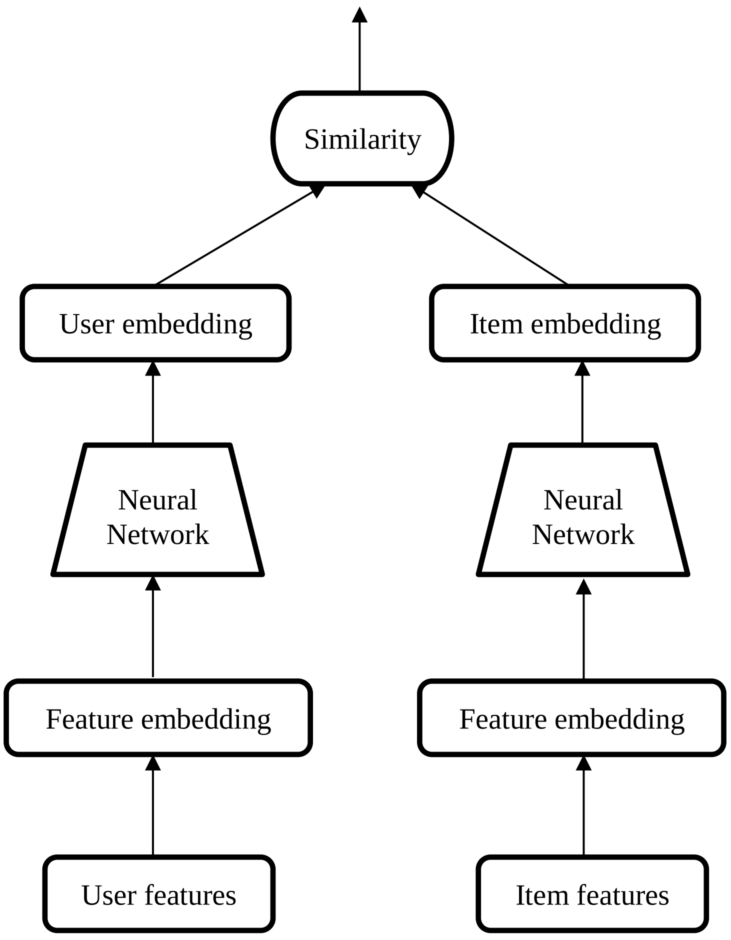

Figure 3.3: Architecture of the Duetl

As shown in Figure 3.3, *user/item features* include information about users (e.g., represents the user's preferences, queries, or historical interactions such as a set of words or tokens that reflect the user's current interests or preferences) and items (e.g., content features such as collection of documents or textual descriptions). *Feature embedding* converts textual user and item features convert into continuous vectors using word embeddings and an embedding layer.

Neural Network includes multiple layers allow the model to learn complex patterns and representations from the feature embeddings. *User and item embedding* are continuous vectors that encode the characteristics, preferences, or features of users and items. *Similarity* Calculation calculated using cosine similarity, dot product or another distance metrics. Output is a relevance or similarity score, indicating the predicted preference or likelihood of interaction between the user and the item.

3.3.4 Kernelized Neural Ranking Model (KNRM)

Neural network architecture called the Kernelized Neural rating Model (KNRM) was created for IR tasks. It focuses primarily on the relevance rating of documents in response to a query. In order to capture both exact matches and soft semantic matches between query and document terms, KNRM blends kernel-based methods with deep learning. KNRM, created by academics at the University of Massachusetts Amherst, intends to use neural networks' and kernel methods' advantages to increase search engines' efficacy. The main elements and characteristics of KNRM are word embeddings, and interaction matrix. KNRM represents the terms in the query and the document using pre-trained word embeddings (such as word2vec, GloVe). These embeddings aid in finding soft matches between keywords and capture the semantic meanings of the words. The word embeddings of the query and document terms are used to build an interaction matrix. The similarity between a pair of terms from the query and the document is represented by each row in this matrix, and is usually calculated using a similarity measure like the dot product or cosine similarity.

Gaussian kernels converts the interaction matrix into a set of kernel scores, KNRM uses several Gaussian kernels. Each kernel captures varying degrees of similarity between the query and document terms are parameterized by a mean and a variance. The model can capture both precise matches and semantic matches to these kernels, which are tailored to concentrate on particular similarity ranges. Kernel pooling throughout the query and document, the matching signals are aggregated by pooling the kernel scores. A fixed-length feature vector that condenses the relevant signals that each of the many kernels was able to collect is produced by this pooling operation. Fully connected (dense) layers receive the pooled kernel scores. In order to determine the final relevance score for the query-document pair, these layers learn to aggregate the kernel scores.

The model may learn intricate relationships between the kernel scores and determine a final relevance assessment to the deep layers. A ranking loss function like pairwise hinge loss or cross-entropy loss, is used to train KNRM. In order to ensure that the model learns, this loss function optimizes the model to rank pertinent documents higher than irrelevant ones. Through the integration of neural networks and kernel approaches, KNRM offers an effective framework for relevance ranking in IR applications. While the neural network layers allow the model to learn intricate relevance patterns, the Gaussian kernels allow the model to capture both exact and soft matches between query and document terms. It has been demonstrated that KNRM significantly outperforms conventional retrieval models, especially in situations where capturing fine-grained relevance signals is essential.

KNRM [82] is a neural ranking model that uses a CNN to learn term-to-term matching signals and applies a kernelized function to measure the importance of terms in the matching process. A CNN learns the matching signals between these terms. Then, kernelized functions measure the importance of these matching signals. KNRM uses CNN to learn matching signals between query and document terms. It then applies kernelized functions to measure the importance of terms in the matching process.

$$KNRM = \sum_{i=1}^{L} \sum_{j=1}^{J} f_{kernel}\left(q_i, d_j\right) * Soft - TF(q_i) * Soft - TF(d_i) \tag{3.4}$$

where L represents the number of terms in the query, J is the number of terms in the document, q_i and d_j denote the embeddings of the i^{th} term in the query and j^{th} term in the document, respectively. Additionally, f_{kernel} is a kernel function assessing the similarity between terms by the Equation 3.4.

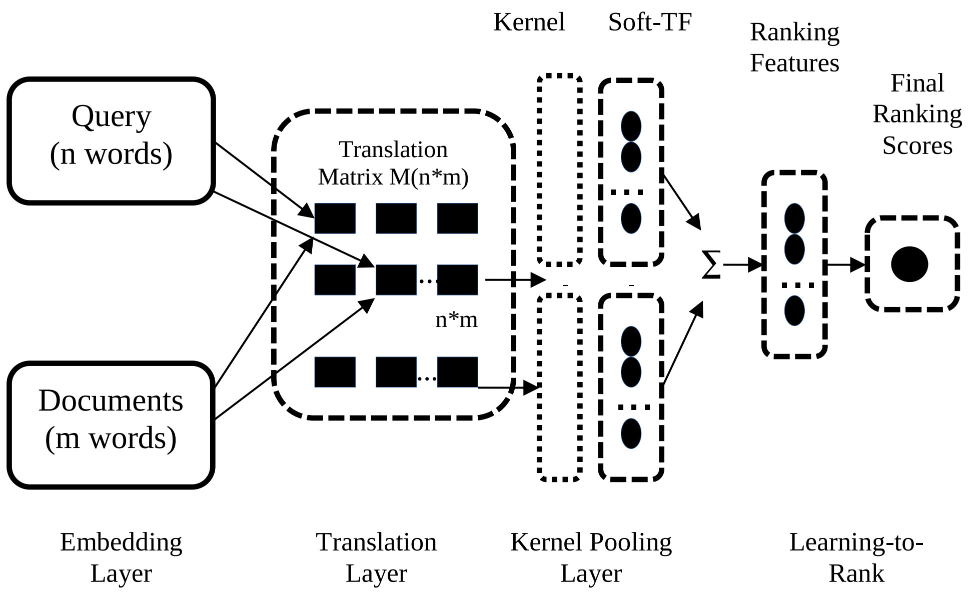

Figure 3.4: Architecture of the Kernelized Neural Ranking Model

As in Figure 3.4, input query *n words* (include sequence of words or tokens) and document *m words* (include collection of documents) in the *embedding layer* converting words or terms in the query and document into continuous vector representations. *Translation layer* calculates the word-word similarities between the terms in the query and documents and forms the translation matrix, the *kernel pooling layer* employs multiple kernel functions, each designed to capture specific matching patterns between terms in the translated query and document embeddings and generate soft-TF counts as ranking features. Soft-TF is a weighting mechanism that aims to model the importance of terms based on their frequencies. The process of ranking features in the *learning-to-rank* phase involves extracting informative features from the kernel pooling output and using term to train a model that can accurately predict the relevance of documents to queries and combines the soft-TF to the final ranking score.

3.3.5 Position-aware Convolutional Recurrent Relevance (PACRR)

With an emphasis on relevance matching between queries and documents, the Position-Aware Convolutional Recurrent Relevance Model (PACRR) is a neural network architecture created for IR applications.

In order to capture both long-range dependencies and local matching patterns, PACRR blends convolutional and recurrent neural networks. Additionally, it incorporates term positioning information to enhance relevance estimation. The main elements and characteristics of PACRR are interaction matrix, and convolutional layers. PACRR creates an interaction matrix that depicts the similarities between query terms and document terms, much like other relevance models like DRMM and MP. Usually, cosine similarity or dot product based on word embeddings is used to populate this matrix. Convolutional layers extract local matching patterns; PACRR uses convolutional layers to the interaction matrix. In order to find n-gram matches, the convolutional filters pass across the interaction matrix, catching both exact and approximatively word matches.

Position-aware pooling is carried out by PACRR following the convolutional layers. Positional information is preserved as the convolutional features are aggregated in this stage. Position-aware pooling helps the model grasp the positioning context of matching words by tracking the locations of maximum values, as opposed to regular max-pooling, which discards positional information. Layers of a RNN layers, such as LSTM or GRUs, are used in PACRR to capture long-range relationships and sequential patterns in the query-document interactions. In order to learn, these RNN layers process the position-aware pooled features. Dense, fully connected layers get input from the RNN layers.

A final relevance score for the query-document pair is generated by combining the features that have been extracted by these layers. The model can learn intricate relationships between the convolutional and recurrent features because to the deep layers. Ranking loss function to optimize the model and rank pertinent documents higher than irrelevant ones, PACRR is trained using a ranking loss function, such as pairwise hinge loss or cross-entropy loss. This loss function makes sure the model can discriminate between different levels of relevance. A complete framework for relevance matching in IR tasks is provided by PACRR, which combines position-aware pooling with convolutional and recurrent neural networks. Local matching patterns are captured by the convolutional layers, within the RNN layers.

PACRR [26] is a neural model that combines CNN and RNN to capture hierarchical matching patterns between queries and documents. It is known for its ability to capture positional information.

A hierarchical structure is visualizing where term pairs are compared at different levels. Convolutional and recurrent layers analyze these pairs while considering their positions in the text. PACRR combines CNN and RNN to capture hierarchical matching patterns. It considers both term similarity and term position in the document.

$$L(q, d^+, d^-; \theta) = max(0,1 - rel(q, d^+) + rel(q, d^-)) \tag{3.5}$$

where a query q, relevant document d^+, and non-relevant document d^-, minimizing a standard pairwise max margin loss by the Equation 3.5.

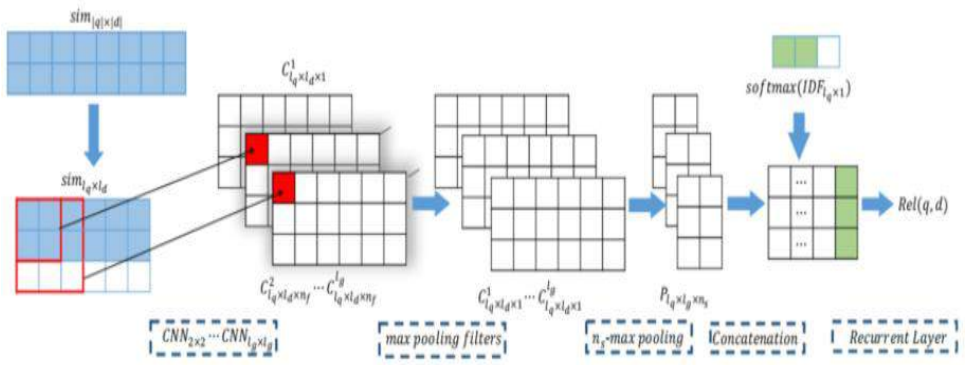
$sim_{|q|\times|d|}$
$sim_{l_q \times l_d}$
$C^1_{l_q \times l_d \times 1}$
$C^2_{l_q \times l_d \times n_f} \cdots C^{l_g}_{l_q \times l_d \times n_f}$
$C^1_{l_q \times l_d \times 1} \cdots C^{l_g}_{l_q \times l_d \times 1}$
$P_{l_q \times l_g \times n_s}$
$softmax(IDF_{l_q \times 1})$
$Rel(q,d)$
$CNN_{2\times2} \cdots CNN_{l_g \times l_g}$
max pooling filters
n_s-max pooling
Concatenation
Recurrent Layer

Figure 3.5: Architecture of the Position-aware Convolutional Recurrent Relevance

As in Figure 3.5, each query q and document d is first converted into a query-document similarity matrix $sim_{|q|\times|d|}$. Thereafter, a distillation method (first k is displayed) transforms the raw similarity matrix into unified dimensions, namely, $sim_{|q|\times|d|}$. Here, l_{q-1} *convolutional layers (CNN)* are applied to the distilled similarity matrices. As l_g =3 is shown, layers with kernel size 2 and 3 are applied. Next, *max pooling* is applied, leading to l_g matrices C_1 ... C_{lq}. Following this, $n_s - max$ pooling captures the strongest ns signals over each query term and n-gram size, and the case for ns = 2 is shown here. Finally, the similarity signals from different n-gram sizes are *concatenated*, the query terms normalized IDFs are added, and a *recurrent layer* combines these signals for each query term into a query-document relevance score *rel(q, d).*

3.3.6 Convolutional Kernelized Neural Ranking Model (CONV-KNRM)

For IR tasks, an enhanced neural network architecture called the CONV-KNRM was created. Convolutional layers are added to the KNRM in order to capture n-gram interactions between query and document terms. This method improves relevance estimation by strengthening the model's comprehension of intricate matching patterns and semantic linkages in text. The main elements and functionalities of CONV-KNRM are word embeddings, and covoluational layers. Word embeddings represent the terms in the query and the document and CONV-KNRM uses pre-trained word embeddings (such as word2vec, GloVe). These embeddings act as the input for the layers that follow, capturing the semantic meanings of the words. To be Convolutional layers the word embeddings of the query and document terms, the model adds convolutional layers. By swiping convolutional filters across the embeddings, these layers provide n-gram representations that capture local contextual data and term interaction patterns.

The n-gram representations produced by the convolutional layers are the foundation for building an interaction matrix. The similarity between a pair of n-grams from the query and the document is represented by each item in this matrix. This similarity is usually calculated using a similarity measure like the dot product or cosine similarity. CONV-KNRM converts the interaction matrix into a set of kernel scores by using many Gaussian kernels, just like KNRM does. Every kernel captures both precise and approximate similarities within a particular range. Throughout the query and document, the matching signals are aggregated by pooling the kernel scores. A fixed-length feature vector that condenses the relevant signals that each of the many kernels was able to collect is produced by this pooling operation.

Fully connected (dense) layers receive the pooled kernel scores. In order to determine the final relevance score for the query-document pair, these layers learn to aggregate the kernel scores. The model may learn intricate relationships between the kernel scores and determine a final relevance assessment to the deep layers. A ranking loss function like pairwise hinge loss or cross-entropy loss is used to train CONV-KNRM. By optimizing the model to rank pertinent documents higher than irrelevant ones, this loss function makes sure the model which can distinguish between documents with differing levels of relevance.

Convolutional layers and kernel-based methods are combined to provide CONV-KNRM, a strong framework for relevance ranking in IR problems. The Gaussian kernels provide flexible matching of both exact and semantic similarities while the convolutional layers improve the model's capacity to record n-gram interactions and local context. This combination improves performance across a range of retrieval circumstances, especially those that call for a sophisticated comprehension of textual significance.

CONV-KNRM [11] is an extension of KNRM that incorporates convolutional layers to better model term interactions. It uses CNN to capture multi-level matching patterns. An extension of KNRM with convolutional layers added. These convolutional layers capture more intricate matching patterns between terms. CONV-KNRM extends KNRM by incorporating convolutional layers. This allows it to capture multi-level matching patterns in text.

$$f(q,d) = \tanh(w_r^T\ \emptyset(M) + b_r) \tag{3.6}$$

The Learning-To-Rank (LeToR) layer combines the soft-TF ranking features $\emptyset(M)$ into a ranking score. w_r and b_r are the linear ranking parameters to learn. $|\mathrm{w_r}|$ = $|\emptyset(\mathrm{M})|$ and $|b_r|$ = 1. $tanh()$ is the activation function as in by the Equation 3.6.

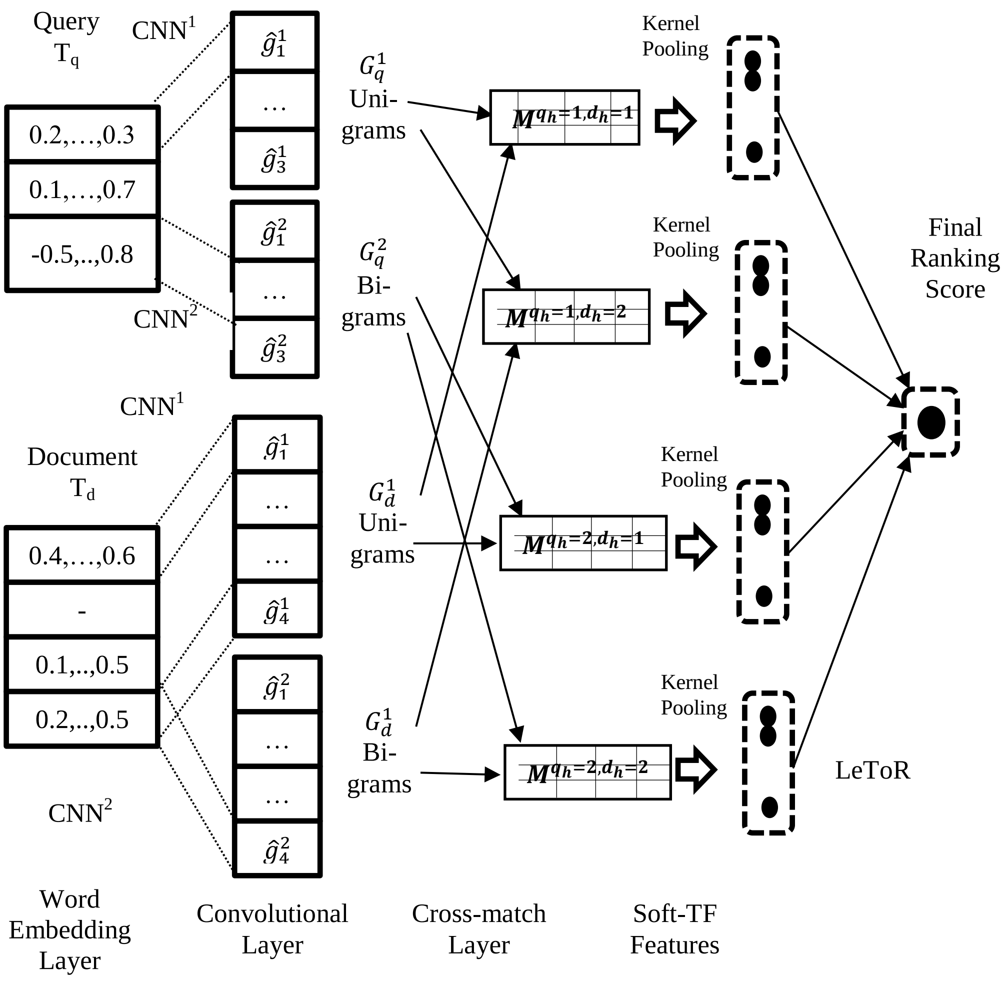

Figure 3.6: Architecture of the Convolutional Kernelized Neural Ranking Model

As in Figure 3.6, given input query T_q and document T_d, the *word embedding layer* maps their words into distributed representations, the *convolutional layer* generates n-gram embeddings; the *cross-match layer* matches the query n-grams and document n-grams of different lengths, and forms the translation matrices; the kernel pooling layer generates *soft-TF features* and the *learning-to-rank (LeToR) layer* combines them to the ranking score. The case with Unigrams and Bigrams (hmax = 2) is shown.

3.3.7 MatchZoo-CONV-KNRM (MZ-CONV-KNRM)

A toolkit called MatchZoo is intended for text matching in NLP and IR. For a variety of matching tasks, such as document retrieval and question-answer matching, it offers a variety of neural network models. CONV-KNRM is one of the models available in MatchZoo. Convolutional Kernel-based Neural Ranking Model, or CONV-KNRM, is a neural ranking model intended for use in ad hoc retrieval applications. CNNs extract local contextual features from text, CONV-KNRM use convolutional layers. The n-gram characteristics, which are essential for comprehending the semantic context of words within a sentence, are captured by the convolutional layers. Kernel Pooling: The model employs a kernel pooling approach following convolution.

The concept is to quantify the degree of similarity between the query and the document at different granularities using a set of kernel functions. This aids in capturing varying levels of query and document term similarity. A neural network is then fed the combined similarity features to provide a relevance score as neural ranking. Documents are ranked by the neural network according to how relevant they are to the query. Benefits of the CONV-KNRM are contextual understanding, and end-to-end learning. CONV-KNRM can better grasp context by capturing intricate, local patterns in text through the use of CNNs. Accurate ranking depends on a flexible and comprehensive measurement of similarity, which is made possible by the kernel pooling layer. End-to-end learning by training the model from beginning to end, all of its components may be optimized at the same time to boost performance.

CONV-KNRM is the process of using a query to find pertinent documents within a sizable corpus for document retrieval. QA is the process of selecting the right response from a list of possible answers for a given question. Any job that involves matching text inputs to pertinent textual responses or resources is known as IR. CONV-KNRM, which combines the benefits of kernel-based similarity measurement and convolutional networks, is an effective tool for text matching and ranking overall. MatchZoo is a framework for text-matching tasks, including IR. Models like MZ-KNRM and MZ-CONV-KNRM [19] are specific implementations of KNRM and CONV-KNRM within the MatchZoo framework, making them easy to use. It can easily adapt and experiment with these models for various IR tasks.

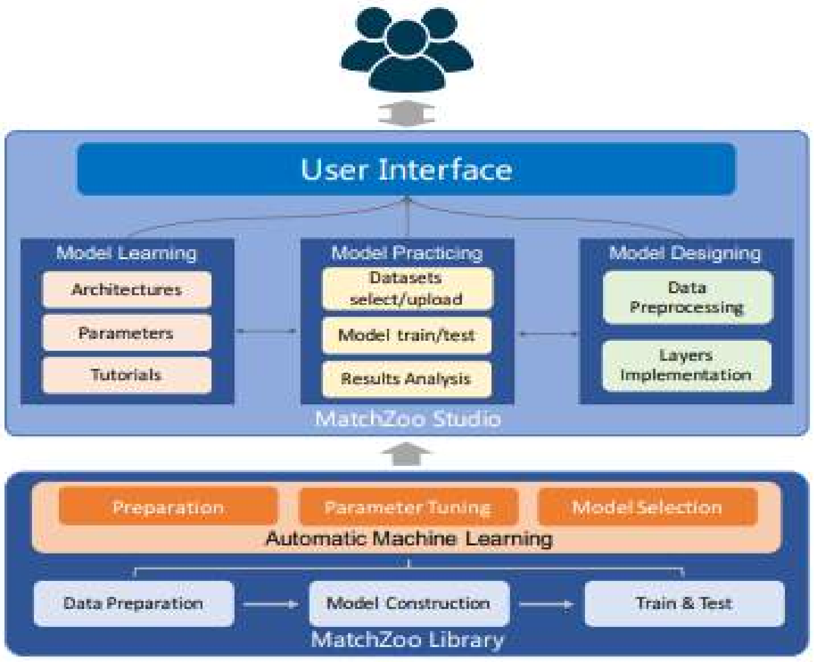

Figure 3.7: An overview of the Match-Zoo Architecture

In Figure 3.7, data preparation involves organizing and pre-processing the dataset to make it suitable for training. Model construction involves selecting a text matching model architecture, configuring its parameters, and building the model. Training and testing a model provide batches of data during training and evaluating the model's performance on a separate test set. In MatchZoo, the process of preparation, parameter tuning, and model selection in automatic machine learning involve the AutoModel class.

3.4 Fine-tuned Model

A pre-trained machine learning model that has been further trained on a particular task or dataset to enhance its performance on that specific task is known as a fine-tuned model. Usually using a smaller, task-specific dataset, fine-tuning applies the skills and information gained during the first pre-training phase to the current task. In pre-training, a model is trained on a sizable, varied dataset for a broad purpose, like image classification on a sizable dataset like ImageNet or language modeling (guessing the word that will appear in a phrase).

The model learns many different traits and patterns that are generally helpful, such as identifying edges, forms, and textures in picture models or comprehending grammar, syntax, and semantics in language models. Examples are pre-training on large datasets is common for models such as BERT, GPT, and ResNet. The model is adjusted using a smaller, task-specific dataset following pre-training in order to modify its general knowledge to meet the demands of the novel task. The process of fine-tuning entails using task-specific data to retrain the previously learned model. The architecture (if necessary) will alter. Lowering the learning pace in order to prevent significant updates that can wipe out previously learned information. Sometimes, only training the final few layers or adding new layers for a given job while freezing some layers preserves the features that were learned during the pre-training phase. Examples are included fine-tuning a pre-trained ResNet for a particular kind of image classification (e.g., medical image analysis) or BERT for sentiment analysis, Named Entity Recognition (NER), or QA.

Since fine-tuning begins with a model that has already acquired valuable representations, it is both computationally faster and less expensive than training a model from scratch. Compared to models trained from scratch using the same quantity of task-specific data, fine-tuned models frequently perform better on certain tasks. Fine-tuning might be helpful when there is a shortage of task-specific data because it takes less data than training from scratch. A model has already been trained and is suitable for the basic task. For example, models like BERT or GPT might be useful if users are working on a linguistic task. Users need to prepare the task-specific Information Collect and then prepare the data that are necessary for the particular task at hand. If required to modify, modify the model architecture. Adding task-specific layers may be necessary for this. Use the task-specific dataset to train the previously trained model, may be with some layers frozen and at a lower learning rate. The optimized model is analyzed using a validation set, and it can be change as needed to enhance performance. Use the task-specific dataset to retrain the previously trained model, usually at a slower learning rate and perhaps with certain layers frozen. Analyze the optimized model using a validation set, and change as needed to enhance performance. In contemporary machine learning, fine-tuning is a potent approach that enables practitioners to take advantage of the advantages of large, pre-trained models and effectively modify them for particular applications.

Language model pre-training has been shown to be effective in improving many NLP tasks [23], [10], [57], [61]. These include sentence-level tasks such as natural language inference [78], [60] and paraphrasing [13], which aim to predict the relationships between sentences by analyzing them holistically, as well as token-level tasks such as NER and QA, where models are required to produce fine-grained output at the token level [64], [68]. There are two existing strategies for applying pre-trained language representations to downstream tasks: feature-based and fine-tuning methods. The feature-based approach, such as ELMo [77], uses task-specific architectures that include the pre-trained representations as additional features. The fine-tuning approach, such as the Generative Pre-trained Transformer (OpenAI GPT) [61], introduces minimal task-specific parameters, and is trained on the downstream tasks by simply fine-tuning all pre-trained parameters.

In this research, it improves the fine-tuning-based approaches by applying BERT: Bidirectional Encoder Representations from Transformers. BERT is conceptually simple and empirically powerful. For this research, it used a fine-tuned model, Vanilla-BERT [12], aiming to improve performance scores. Fine-tuning method is straightforward since the self-attention mechanism in the Transformer allows BERT to model many downstream tasks whether they involve single text or text pairs—by swapping out the appropriate inputs and outputs. For applications involving text pairs, a common pattern is to independently encode text pairs before applying bidirectional cross attention [70], [55].

$$BERT(X) = Transformer\left(Embeddings(X)\right) \tag{3.7}$$

where $\mathrm{BERT(X)}$ represent the output embedding for the input sequence X, X can be a concatenation of a query and a document, $\mathrm{Embeddings(X)}$*:* This involves converting the input tokens into embedding. $Transformer()$*:* This refers to the transformer architecture, which consists of multiple layers of self-attention and feedforward mechanisms.

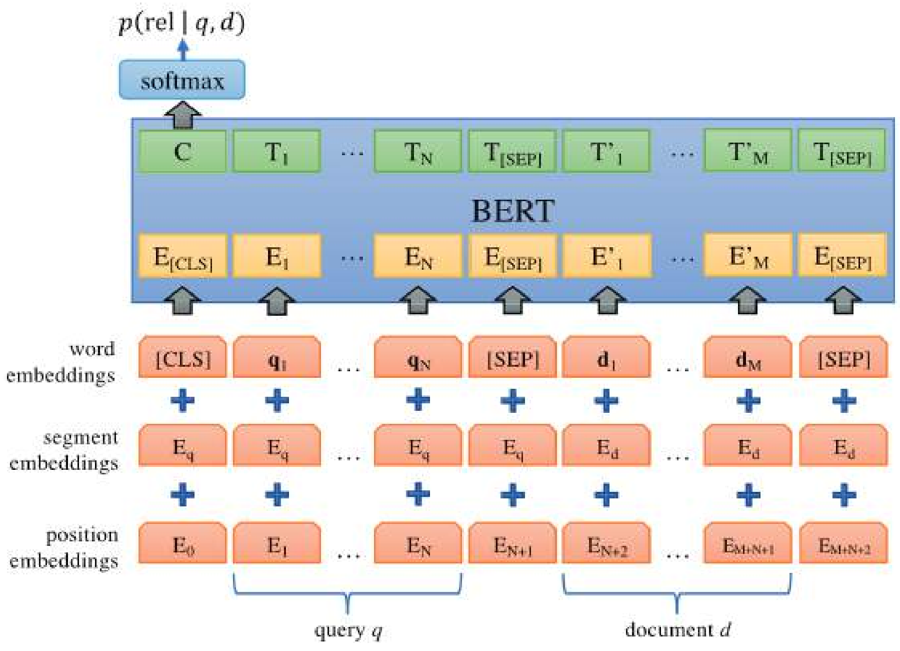

Figure 3.8 BERT for query-document matching

In Figure 3.8 shows a query-document pair (q, d), the input to the BERT model includes query words, document words: *"[CLS],*q_1*, . . . ,* q_N *, [SEP],* d_1*, . . . ,* d_M *, [SEP]*", where "*[CLS]* " is the token to indicate whether the query-document pair is relevant or not, "*[SEP]*" is the token to indicate the separation of query and document, and q_i and d_j are the i^{th} query word and the j^{th} document word, respectively. The query (and document) is padded or truncated to have *N (and M)* words. Each word is represented with its embedding. The input embedding of a word is the sum of the corresponding position embeddings, the segment embeddings, and the word embeddings. Position embeddings adds information about the position of each token in the sequence. Segment embeddings adds information about the segment to which each token belongs. Word embeddings captures the semantic meaning of each token. Outputs a sequence of high level semantic representations for the special input tokens and query and document words: "C, T_1 , . . . , T_N , *T[SEP]* , T_{10} , . . . , T_M, *T[SEP]* ", where *C* is the representation of the token *[CLS]*, *T1 , . . . , TN* of the query words, T_{10} , . . . , T_M of the document words, *T[SEP]* and *T[SEP]* of the two separators. The representation of the *[CLS]* token *C* is fed into an output layer (e.g., single layer neural network) to obtain *p(rel | q, d)* , which represents the probability of document's being relevant to query.

3.4.1 Bidirectional Encoder Representations from Transformers (BERT)

The Bidirectional Encoder Representations from Transformers (BERT) architecture is utilized by a BERT-based model in an IR system to enhance the process of locating pertinent documents or information fragments in response to a user query. This is a thorough rundown of BERT's applications in infrared systems: Google created the transformer-based BERT paradigm. It is pre-trained using two training tasks on a big corpus of text; BERT can comprehend a sentence's context in both directions by learning to anticipate words that are lacking in MLM. Next, Sentence Prediction (NSP) analyzes the connections between two sentences; BERT enhances its comprehension of sentence relationships and context. BERT can be used in an IR system in a number of ways to increase the relevancy and precision of search results. Understanding a query by identifying the subtleties and context in the query wording, BERT can be utilized to comprehend the user's inquiry more fully. This makes it easier to understand the user's intent.

DL models that have been pre-trained on copious quantities of textual data in order to uncover the underlying patterns and connections between words and phrases are known as pre-trained text models. These models are intended to capture the context and meaning of text data, and they are taught using methods like neural networks. They are the ideal instrument for determining the text embeddings of a document. BERT is one example of such a model. Based on the training data used to generate the vectors, static word embeddings map words with multiple senses into an average or most common-sense representation. A word's vector remains constant regardless of the other words used in the phrase around it. The employment of a unique neural layer dubbed self-attention in combination with feed forward and linear layers, the transformer neural network is able to explicitly consider the context of any length of text. Sequences of input length are mapped to sequences of output length by the self-attention layer. The layer can access all n input elements (bidirectional self-attention) or just the first i input elements (causal self-attention) while calculating the i^{th} output element. The network is able to consider the relationships between several elements in the same input to a self-attention layer. A self-attention layer computes token representations that consider the surrounding words when the input elements are tokens of a specific text. In doing so, the transformer computes contextualized word embeddings, in which the input text as whole conditions each input token's representation.

Transformers have shown effective in a variety of natural language processing applications, including QA, summarization, machine translation, and more. These jobs are all specific examples of a broader objective, which is to convert an input text sequence into an output text sequence. This general task has been addressed by the invention of the sequence-to-sequence model. The two components of a sequence-to-sequence neural network are an encoder model that produces a contextualized representation of each input element given an input sequence, and a decoder model that uses these contextualized representations to produce an output sequence tailored to a job. The components of both types are many stacked transformers. Bidirectional self-attention layers are used by the encoder's transformers on either the input or the output sequence from the preceding transformer. The decoder's transformers use bidirectional cross-attention on the output of the final encoder transformer and causal self-attention on the output of the preceding decoder transformer.

Two particular applications of sequence-to-sequence models have been investigated in Neural IR: encoder-only models and encoder-decoder models. All of the tokens in a given input sentence are received as input by encoder-only models, which then compute an output contextualized word embedding for each token in the phrase. The models BERT [12], Robustly Optimized BERT Pre-training Approach (RoBERTa) [42], and Distilled BERT (DistilBERT) [69] are examples of this family of models. Depending on the input sentence provided, encoder-decoder models produce new output sentences. One token at a time, the decoder model sequentially accesses these embeddings to produce new output tokens, while the encoder model takes all of the tokens of a given sequence as input and creates a contextualized representation. The input values are normalized into a probability distribution using the soft-max procedure. Within the domain of deep learning, the inputs of a soft-max operation are commonly referred to as logits. These inputs are the unprocessed predictions produced by a multi-class classification model, which the soft-max operation transforms into a probability distribution across the classes. A sequence-to-sequence model can be trained as a Casual Language Model (CLM), like T5, or as a Masked Language Model (MLM), like BERT, depending on the training goal. While CLM training focuses on predicting the next token in an output sequence given the previous tokens in the input sequence, MLM training teaches learners to predict missing tokens in a sequence given the surrounding tokens.

BERT can be used to determine how similar the texts in the corpus are to the query as document ordering and semantic matching. BERT has the ability to ascertain the degree of semantic similarity between the query and documents by encoding them into high-dimensional vectors. Re-ranking based on a more thorough contextual understanding; BERT can improve the order of the original list of documents that were obtained by a conventional IR model (such as BM25). BERT can assist in locating and prioritizing particular text portions within documents that are most pertinent to the query, as opposed to retrieving entire documents as retrieve passage. This is especially helpful for lengthy publications where only a few sections might be pertinent. BERT can be used in QA systems to extract the document's most pertinent response as extract the answer. BERT can identify the section of text that most effectively responds to a given question given the inquiry and its context (such as a paragraph). An example of the query preprocessing workflow for a BERT-based IR system transforms the query into input tokens that BERT can process, tokenize it using the tokenizer built into BERT. To get a starting collection of potentially pertinent documents, use a conventional IR approach (such as BM25). BERT is used to encode the query as well as any documents or passages that are obtained. To obtain contextual embeddings, the text must be fed through BERT. The degree of similarity is determine between each document/passage embedding and the query.

Cosine similarity or other distance metrics can be used for this. Reordering: Based on how closely the papers or passages match the query, order them. The things with the highest scores are deemed to be the most pertinent. BERT is used to determine the most pertinent text segment among the top-ranked papers if the task requires locating a specific answer inside a document. Unlike previous models, BERT is bidirectional, which helps it better understand the text's semantics and context.

BERT's utilization of deep learning enables it to assimilate minute details and enhance the significance of search outcomes. BERT is adaptable to a range of IR applications, such as document rating, response extraction, and query interpretation. Obstacles and Real-time IR systems may face difficulties due to the computationally demanding nature of BERT models. The memory and processing power limitations of BERT make it difficult to handle large-scale corpora. Labeled data and computing resources are needed for BERT to perform better when fine-tuned on domain-specific data.

BERT-based models have significantly advanced the field of IR by providing more accurate and contextually aware search results. By integrating BERT into IR systems, developers can leverage its powerful language understanding capabilities to enhance the user experience and improve the relevance of retrieved information.

3.5 Summary

This chapter discussed the methodologies and theoretical background of Deep Neural Networks. It explained what about DNNs, architectures of DNNs, type of the Deep Neural Ranking Models and Fine-tuned Model are described how they work.

CHAPTER 4
BUILDING MYANMAR NEWS DATASET

This chapter covers the development of text datasets used in this research. Text dataset building is an imperative and a very first task for implementing any Myanmar News retrieval system. Although there are freely and widely available resources in well-resourced language like English, the text dataset is needed to build first for Myanmar which has no available text dataset easily to use. Moreover, four evaluation methods such as MAP, MRR, P@1, and P@3 metrics are described in this chapter

4.1 Building Myanmar News Dataset

Text data collection is the very first step in any IR tasks especially for under resourced language which is to gather the text data. The main problem in IR research for Myanmar language is the lack of proper data. Therefore, the text dataset is necessary to develop first. The text dataset is an abundant collection of Myanmar languages and is important for IR. For Myanmar News retrieval, as the first contribution of this work, the text dataset needs to build first systematically for Myanmar language. The text data were obtained from main sources: [1] online source and collected in this research. The next steps will be exact to features for retrieval process if the data have been prepared properly. Many IR systems are constructed on the neural models based on the text data. Therefore, text dataset building is essentially needed to develop the retrieval related systems. The text dataset used in this work is constructed by collecting from the main sources: Web based collected news and for the purpose of training the IR system.

Due to the lack of a large dataset for the retrieval task on Search Engine, it decided to develop a large Myanmar News dataset containing 118,486 documents composed in Myanmar Unicode font collected from the Myanmar News webpages. Types of news are Health, Sport, Entertainment, Political and Economic. Each document consists of two parts: title and contents. Table 4.1 shows collections of Myanmar News datasets.

[1] Eleven Broadcasting media, Mizzima News Myanmar, Irrawaddy Burmese News, 7days news, Thit Htoo Lwin Burmese News

Table 4.1 Statistics of Myanmar News Datasets

Number of documents	Number of sentences	Number of words
118,486	54,634,415	1,283,260,155

Text collection is the most important effort in every retrieval related system. Nowadays, Myanmar News is available on many Web sites. From the sites of [2]Eleven Myanmar News, [3]7days Myanmar News, [4]Irrawaddy Myanmar News, [5]Mizzima Myanmar News, and [6]Thit Htoo Lwin Myanmar News.

4.1.1 Data Pre-processing

Textual data from search engine, especially Google, is unstructured and contains noisy tokens or stop words, which need to be cleaned to improve its quality and usefulness for training deep learning models. Data pre-processing methods prepare data for further processing, verify its integrity and consistency, reduce data noise, fill in missing values, and structure it in databases. To facilitate data cleaning, preparation techniques of Myanmar dataset are word segmentation and stop word removal.

[2] https://news.eleven.com/
[3] https://www.7Daynews.com/
[4] https://www.irrawaddy.com/
[5] https://www.mizzimaburmese.com/
[6] http://www.thithtoolwin.com/

4.1.1.1 Word Segmentation

The foundational task in NLP is word segmentation. This procedure involves breaking down text into individual words and sentences, where the objective is to identify word tokens and sentence boundaries. While English word boundaries are easily defined, the same does not hold true for Myanmar. Myanmar word boundaries often lack spacing within sentences, making it challenging to discern individual words. Hence, in the context of IR, effective word segmentation proves invaluable for navigating sentences and word tokens. To fulfill this objective, the Myanmar word segmentation tool is employed which is developed by UCSY [52]. Examples of word segmentation are additionally illustrated in Table 4.2. During the pre-processing stage, the significance of word segmentation is heightened, particularly in the context of IR evaluation.

Table 4.2 Example of Word Segmentation

Original sentences	After word segmentation
မိုးသည်းထန်စွာ ရွာသည်နှင့်တစ်ပြိုင်နက် ရေလျှံမှုများ ဖြစ်ပေါ်နေသည့်ရန်ကုန်	"မိုးသည်းထန်စွာ”, “ ရွာသည်နှင့်", "တစ်ပြိုင်နက်", "ရေလျှံမှုများ", "ဖြစ်ပေါ်", "နေသည့်", "ရန်ကုန်"
ကပ္ပလီပင်လယ်ပြင်နှင့် ဘင်္ဂလားပင်လယ်အော် တောင်ပိုင်းတို့တွင် တိမ်အသင့်အတင့်မှ တိမ်ထူထပ်နေပြီး ကျန်ဘင်္ဂလားပင်လယ်အော်တွင် တိမ်အနည်းငယ်ဖြစ်ထွန်းနေကြောင်းသိရသည်	"ကပ္ပလီ", "ပင်လယ်ပြင်", "နှင့်", "ဘင်္ဂလား", "ပင်လယ်အော်", "တောင်ပိုင်း”, “တို့တွင်”, “တိမ်”, “အသင့်အတင့်”, “မှ”, “တိမ်”, “ထူထပ်”, “နေ”, “ပြီး”, “ကျန်”, “ ဘင်္ဂလားအော်”, “တွင်”, “တိမ်”, “အနည်းငယ်”, “ဖြစ်ထွန်း”, “နေ”, “ကြောင်း”, “သိရသည်”
ပုံမှန်သွေးလှူနိုင်အောင် ကိုယ်ကကျန်းမာနေဖို့ လိုအပ်တဲ့အတွက် ပုံမှန်အိပ်၊ ပုံမှန်စား အနေအထိုင်သင့်တင့်မျှတအောင် နေသင့်တယ်	“ပုံမှန်”, “သွေး”, “လှူ”, “နိုင်”, “အောင်”, “ကိုယ်က”, “ကျန်းမာ”, “နေ”, “ဖို့”, “လိုအပ်”, “တဲ့အတွက်”, “ပုံမှန်” “အိပ်”, “၊”, “ပုံမှန်”, “စား”, “အနေအထိုင်”, “သင့်တင့်”, “မျှတ”, “အောင်”, “နေ”, “သင့်”, “တယ်”

4.1.1.2 Stop-word Removal

The aim of stop word removal is to filter out words that are prevalent in the majority of documents. In Myanmar, stop words encompass ရ, သည်, မ, မှ, နှင့်, ခဲ့, တွေ, မည်, မယ်, ရန်, ထံ, ပါ, က, များ, ကို, တွင်, etc. Illustrations of Myanmar stop words removal are also presented in Table 4.3, removed the Myanmar stop words are "စွာ, သည်, နှင့်, များ, သည့်, တို့, တွင်, ပြီး, ကြောင်း, သည်, က", etc. This stage holds significant importance in the pre-processing techniques applied in NLP methods.

Table 4.3 Example of Stop-word Removal

Original sentences	After stop-word removal
မိုးသည်းထန်စွာ ရွာသည်နှင့်တစ်ပြိုင်နက် ရေလျှံမှုများ ဖြစ်ပေါ်နေသည့်ရန်ကုန်	"မိုးသည်းထန်", " ရွာ", "တစ်ပြိုင်နက်", "ရေလျှံမှု", "ဖြစ်ပေါ်", "နေ", "ရန်ကုန်"
ကပ္ပလီပင်လယ်ပြင်နှင့် ဘင်္ဂလားပင်လယ်အော် တောင်ပိုင်းတို့တွင် တိမ်အသင့်အတင့်မှ တိမ်ထူထပ်နေပြီး ကျန်ဘင်္ဂလားပင်လယ်အော်တွင် တိမ်အနည်းငယ် ဖြစ်ထွန်းနေကြောင်းသိရသည်	"ကပ္ပလီ", "ပင်လယ်ပြင်", "ဘင်္ဂလား", "ပင်လယ်အော်", "တောင်ပိုင်း", "တိမ်", "အသင့်အတင့်", "တိမ်", "ထူထပ်", "နေ", "ကျန်", "ဘင်္ဂလားအော်", "တိမ်", "အနည်းငယ်", "ဖြစ်ထွန်း", "နေ", "သိရ"
ပုံမှန်သွေးလျှူနိုင်အောင် ကိုယ်ကကျန်းမာနေဖို့ လိုအပ်တဲ့အတွက် ပုံမှန်အိပ်၊ ပုံမှန်စား အနေအထိုင်သင့်တင့်မျှတအောင် နေသင့်တယ်	"ပုံမှန်", "သွေးလှူ", "နိုင်အောင်", "ကိုယ်", "ကျန်းမာ", "နေ", "လိုအပ်", "ပုံမှန်" "အိပ်", "ပုံမှန်", "စား", "အနေအထိုင်", "သင့်တင့်", "မျှတ", "အောင်", "နေသင့်"

4.2 Evaluation Metrics for Myanmar News Retrieval

The performance of the system is measured using MAP, MRR, P@1, and P@3 metric to assess the performance of IR systems by comparing their retrieved results to the ground truth relevance assessments. These metrics are commonly used in IR evaluation to assess the quality of ranking systems. Higher values for these metrics indicate better-performing systems. These performance metrics commonly used for evaluating neural networks in IR and recommendation tasks: MAP (Mean Average Precision), MRR (Mean Reciprocal Rank), P@1 (Precision at 1), and P@3 (Precision at 3).

These equations provide a quantitative measure of the performance of a ranking system based on different aspects such as precision, average precision and reciprocal rank. They are used in IR to assess the quality of ranked lists of documents.

4.2.1 MAP (Mean Average Precision)

MAP is a metric used to evaluate the performance of a ranking system. It calculates the average precision across multiple queries. For each query, calculate precision at each position where a relevant document is found in the ranked list. These precision values are averaged, compute the mean (average) precision across all queries. A higher MAP indicates a better ranking system. MAP ranges from 0 to 1.

$$\mathrm{MAP} = \frac{1}{\mathrm{Q}} \sum_{\mathrm{i}=0}^{\mathrm{Q}} \mathrm{AveragePrecision}_{\mathrm{i}} \tag{4.1}$$

where Q is the number of queries and $AveragePrecision_i$ is the average precision for query i as in Equation (4.1).

4.2.2 MRR (Mean Reciprocal Rank)

MRR is a metric that measures the effectiveness of a ranking system based on the reciprocal of the rank of the first relevant item. For each query, the rank of the first relevant document is identified in the ranked list. Taking the reciprocal of this rank and computing the mean (average) of these reciprocal ranks are across all queries. A higher MRR indicates a better ranking system. MRR ranges from 0 to 1.

$$\mathrm{MRR} = \frac{1}{\mathrm{Q}} \sum_{\mathrm{i}=1}^{\mathrm{Q}} \frac{1}{\mathrm{Rank}_{\mathrm{i}}} \tag{4.2}$$

where Q is the number of queries and $Rank_i$ is the rank of the first relevant document for query i as in Equation (4.2).

4.2.3 P@1 (Precision at 1)

P@1 is a metric that evaluates the precision of a ranking system at the top position. For each query, check if the first document in the ranked list is relevant.

If relevant, P@1 is 1; otherwise, it is 0. The mean (average) precision at the top position is computed across all queries. P@1 measures the precision of the top-ranked document for each query. P@1 ranges from 0 to 1.

$$\mathrm{P@1} = \frac{1}{\mathrm{Q}}\sum_{\mathrm{i}=1}^{\mathrm{Q}} \mathrm{Precision_i@1} \tag{4.3}$$

where Q is the number of queries and $Precision_i@1$ is the precision at the top position for query i as in Equation (4.3).

4.2.4 P@3 (Precision at 3)

P@3 is similar to P@1 but considers precision at the top 3 positions. For each query, check if any of the top 3 documents in the ranked list are relevant. Calculate precision as the number of relevant documents among the top 3 is divided by 3. The mean (average) precision at the top 3 positions is computed across all queries. P@3 measures the precision of the top 3 documents for each query. P@3 ranges from 0 to 1.

$$\mathrm{P@3} = \frac{1}{\mathrm{Q}}\sum_{\mathrm{i}=1}^{\mathrm{Q}} \mathrm{Precision_i@3} \tag{4.4}$$

where Q is the number of queries and $Precision_i@3$ is the precision at the top 3 positions for query i as in Equation (4.4).

4.3 Summary

This chapter presents building the Myanmar News dataset for using in IR. It describes collecting, preparing and segmenting the Myanmar data obtained from Web. The Myanmar News datasets are also created. The number of news and different type of the Myanmar News contained in Myanmar dataset are represented and finally express the information of Myanmar datasets used in this work. Moreover, four evaluation methods such as MAP, MRR, P@1, and P@3 metrics are described in this chapter.

CHAPTER 5
THE PROPOSED SYSTEM ARCHITECTURE

Myanmar News retrieval is to retrieve query based on a Myanmar phrase. It is the way to make relevance documents on the basis of collected Myanmar datasets. There are many perspectives approaching to IR system from the aspects of data point of view, and state-of-the-art technologies to enhance the retrieval performance. This work emphasizes from the data point of view to boost the performance of the ranker models. The basic structure of the Myanmar News retrieval system, including retrieving and relevance of the documents as well as similarity score, is shown in this chapter. It also presents the design and implementation processes of proposed system architecture together with clear understanding of pictorial representation.

5.1 Basic Structure of Information Retrieval System

An IR system is developed in order to help users to discovery relevant information from a storehouse containing collection of documents. The idea of IR assumes that there exist several documents or records comprising data have been arranged in a suitable order for easy retrieval is depicted in Figure 5.1.

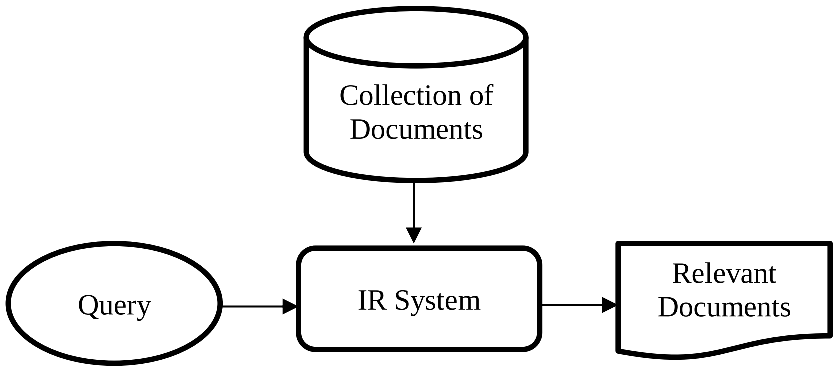

Figure 5.1 Basic Structure of Information Retrieval System

5.2 Basic Structure of Neural Information Retrieval System

Neural IR is a subfield of IR that leverages neural networks to improve the search and retrieval of relevant information from large datasets. Traditional IR systems often rely on techniques such as keyword matching and statistical methods, but Neural IR aims to enhance these processes by incorporating advanced machine learning models, particularly deep learning is depicted in Figure 5.2.

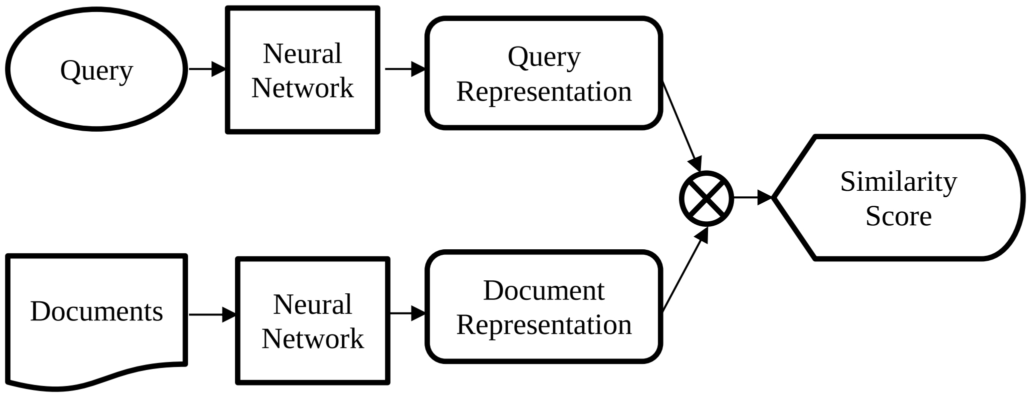

Figure 5.2 Basic Structure of Neural Information Retrieval System

5.3 Proposed System Architecture

The process of precisely retrieving documents through analysis of their relevance query is known as IR. This section describes the design and implementation of proposed Myanmar News retrieval system architecture with pictorial representation as shown in Figure 5.3.

As shown in Figure, there are two phases in Myanmar News retrieval system: training and testing phase.

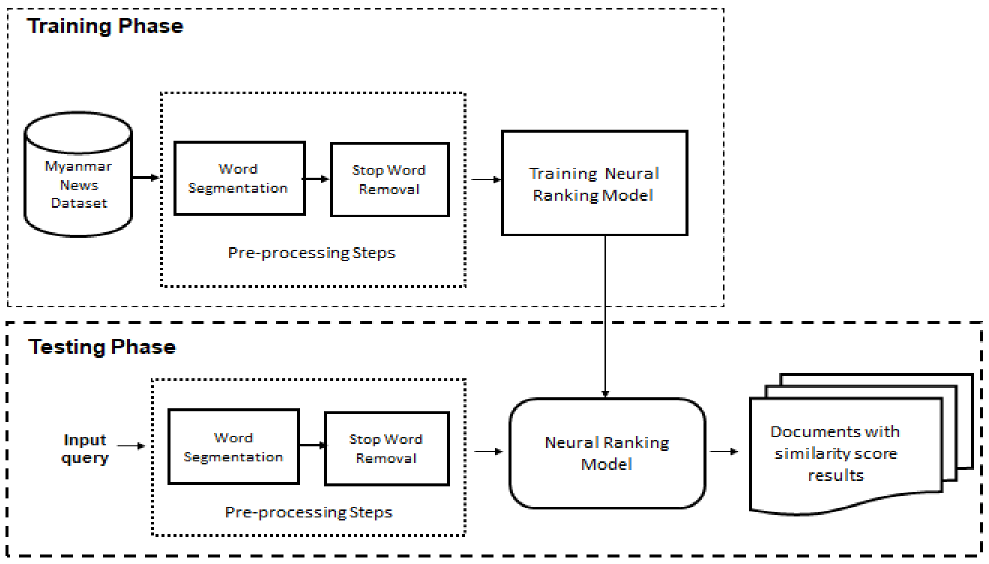

Figure 5.3 Proposed Architecture of Myanmar News Retrieval System

In pre-processing steps, the original clean data are obtained from the raw data which come from online source data. As part of online source data, the raw data are mainly taken in the Zawgyi font. These fonts are converted to Unicode font.

In the process of training neural ranking model, the training data for updating model weights given a training dataset and designed to continue training from a previous checkpoint if restarted, by using saved versions of both the ranker weights and optimizer state and by fast-forwarding a dataset back to its previous state. The neural ranking model based on DRMM, MP, Duetl, KNRM, PACRR, CONV-KNRM, MZ-CONV-KNRM is constructed in the training phase. The model corresponding to each of the individual token is obtained by adapting the parameters with the use of evaluation metrics such as MAP, MRR, P@1, and P@3.

After the training phase, the neural ranking model is assessed in the testing phase. At first, the input text query is preprocessed and then retrieved the relevance of similarity score documents.

5.4 Fine-tuned Model

The process of fine-tuning a machine learning model involves pre-trained it and then retraining it on a more focused, smaller collection of data. The goal of fine-tuning is to preserve a pre-trained model's initial capabilities while modifying it to fit more specific use cases.

On the tasks for which it was fine-tuned, a fine-tuned model's performance can outperform the initial pre-trained model. An existing model that has already been trained on a sizable, varied data set to acquire a broad variety of features and patterns are the starting point for fine-tuning. Through the process of identifying underlying patterns and characteristics in its training data, the pre-trained model gains the ability to generalize during this initial training. The model learns to accurately understand fresh data over time. The goal of this procedure is to strike a compromise between optimizing the model's performance on the fine-tuning use case and preserving its important basic information. In order to do this, model creators frequently choose a lower learning rate, a hyper-parameter that specifies the amount by which a model's weights are changed during training. By preventing significant modifications to the already learned weights, a reduced learning rate can be set during fine-tuning to help ensure the model retains its current knowledge.

Fine-tuning involves taking a pre-trained model and continuing its training on a smaller, task-specific dataset. This process adapts the general knowledge embedded in the pre-trained model to the specific requirements and nuances of the target task. Fine-tuning typically involves dataset preparation collecting and preparing a dataset that reflects the specific IR task or domain, transfer learning leveraging the pre-trained model and its learned representations, and task-specific training trains the model further using the prepared dataset, often with a lower learning rate to adjust the pre-trained weights slightly. Fine-tuning enhances the model's ability to perform the specific IR task by adjusting it to the specific characteristics of the data. Since the model starts from a pre-trained state, it requires less time and computational resources compared to training from scratch. The model retains the broad language understanding from the pre-training phase while specializing in the specific IR task.

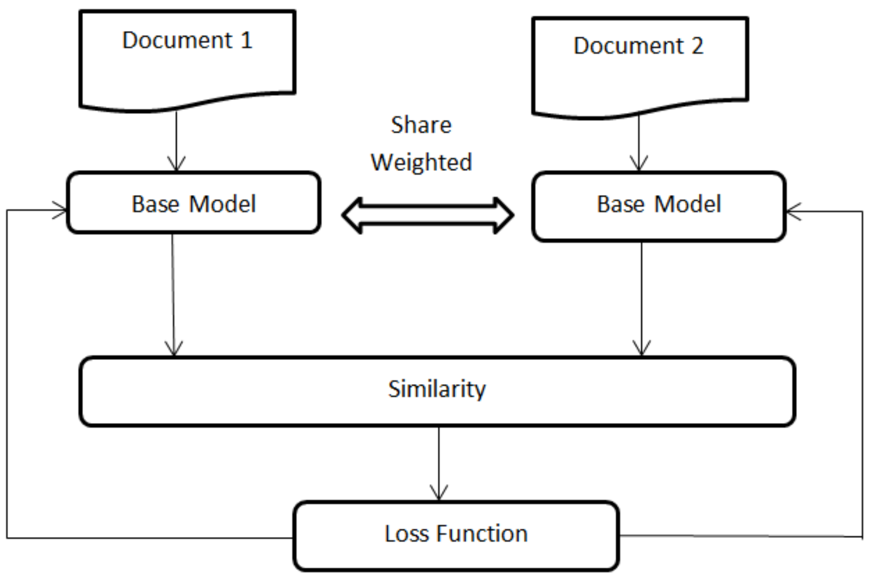

Figure 5.4 Example of Fine-tuned Model

5.4.1 Popular Models and Techniques

BERT (Bidirectional Encoder Representations from Transformers) is a widely used pre-trained language model that can be fine-tuned for various IR tasks, T5 (Text-to-Text Transfer Transformer) is a model that converts all tasks into a text-to-text format and can be fine-tuned for specific IR applications, and GPT (Generative Pre-trained Transformer) though primarily used for text generation, GPT models can be adapted for IR tasks through fine-tuning.

A fine-tuned model in Neural IR is a powerful tool that combines the general language understanding of pre-trained models with the specific requirements of IR tasks, leading to more effective and efficient IR systems.

5.5 Summary

This chapter presents what Myanmar News Retrieval system is, the basic structures of IR system, and Neural IR system with pictorial representations. The design and implementation of proposed architecture of this work are explained in detail together with clear understanding of pictorial representation.

CHAPTER 6

PERFORMANCE EVALUATIONS FOR MYANMAR NEWS RETRIEVAL

This chapter presents the experimental setup regarding with building ranking models, the promising results derived from assessing the performance of the neural ranking models and showing the improvement of recognizing quality by Myanmar News dataset. Building or training the neural ranking models is one of the important phases of Myanmar News retrieval system employed. This is the process of retrieving the documents relevant to query and deals with collecting data from the online source. Only the neural ranking models are constructed, the performance of Myanmar News retrieval system can be assessed by comparing the different neural ranking models. The evaluation of Myanmar News retrieval performance is done with MAP, MRR, P@1, and P@3 with Myanmar News datasets. Moreover, the next experiment is fine-tuned methods on different datasets: Myanmar News and Antique dataset.

6.1 Building Deep Neural Ranking Models

Retrieval models take as input a user query, and then present a set of documents that are relevant to the query. In order to return a useful set of documents to the user, the retrieval model should be able to rank documents based on the given query. This means that the model ranks the documents using features from both the query and documents. Machine learning algorithms can learn ranking models, and the inputs to these models are a set of often hand-crafted features. This setting is known as learning to rank (LTR) using hand-crafted features. These features are domain specific and time-consuming in terms of defining, extracting, and validating a set of specific features for a given task. In order to overcome the limitations of using handcrafted features, deep ranking models that accept raw text data as an input and learn suitable representations for inputs and ranking functions. A key feature in IR models is the relevance judgement.

A ranking model with sufficient capacity is needed to capture the matching signals, and map document-query pairs to accurate prediction of a real-valued relevance score. Deep neural networks are known for their ability to capture complex patterns in both feature extraction and model building phases.

Due to the advantages of deep neural networks, designing neural ranking models to learn both features and model simultaneously. Neural ranking models have many challenges to address in IR tasks. First, the queries and documents have different lengths: the query is usually a short text that typically consists of a few keywords, and the document is long with both relevant and irrelevant parts to the query. Second, in many cases, the query and documents have different terms, so exact matching models cannot be used to accurately rank documents; a neural matching model should be designed to capture semantic matching signals to predict the relevance score. The semantic similarity is context dependent, and another challenge for the neural ranking model is to understand the context of both query and documents in order to generalize across multiple domains.

6.2 Building Fine-tuned Model

Pre-train a neural network model on a source dataset (such as the ImageNet dataset), also referred to as the source model. As the target model, a new neural network model is created. With the exception of the output layer, this duplicates all model designs and parameters on the source model. It expects that the knowledge gained from the source dataset is contained in these model parameters, and that this knowledge will also be relevant to the target dataset. Additionally, it is assumes that the source model's output layer is not employed in the target model because it is strongly correlated with the labels in the source dataset. To the target model, an output layer is added with as many outputs as there are categories in the target dataset. Next, the model parameters are initialized at random. Fine-tuning aids in enhancing the capacity of models to generalize when target datasets are significantly smaller than source datasets as shown in Figure 6.1.

The knowledge acquired from the source dataset is transferred to the target dataset through transfer learning. One typical method for transfer learning is fine-tuning. With the exception of the output layer, the target model replicates the source model's entire model architecture and parameter set, fine-tuning them in light of the target dataset. On the other hand, the target model's output layer requires initial training. While training the output layer from scratch can require a higher learning rate, fine-tuning parameters often requires a lower learning rate.

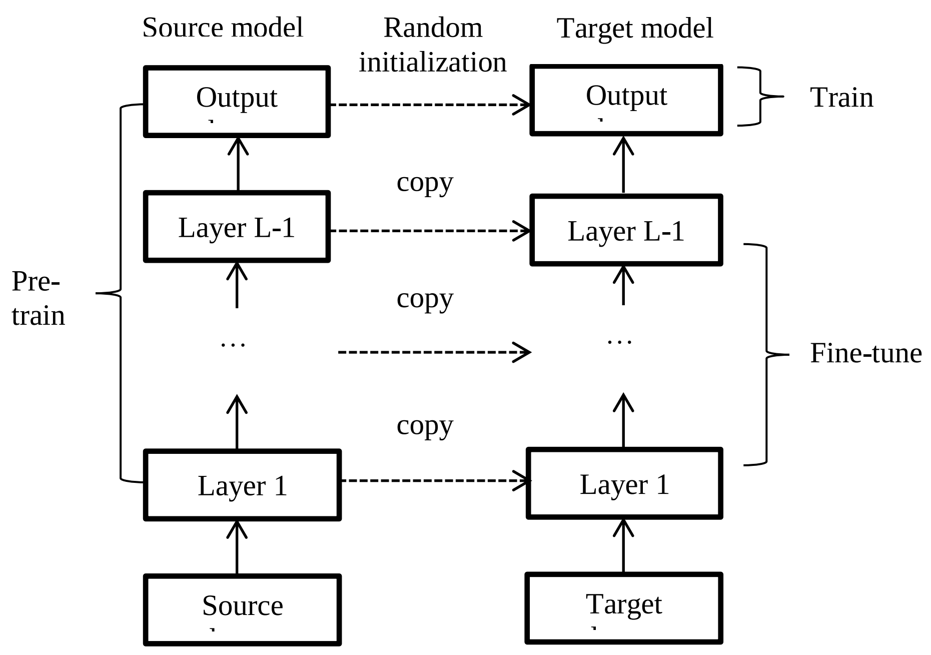

Figure 6.1 Architecture of Fine-tuned Model

6.3 Experiments

Experiments are done on Myanmar News dataset based on deep neural ranking models.

6.3.1 Experimental Setups

The different deep neural ranking models for creating and training deep neural networks are conducted on gpu_determ, Ubuntu Linux machine. Python library is used to implement with Python programming language. When the given amount of training data is trained, all of the models are set at the 20 epochs.

6.3.2 Data Setups

This section trained different deep neural rankings models on the Myanmar News dataset as mentioned in Chapter 3. The detailed information of the Myanmar News dataset is presented in Table 6.1. It applied the DRMM [33], MP [40], Duetl [72], KNRM [34], PACRR [36], CONV-KNRM [35], MZ-CONV-KNRM [73] models in advance datasets.

6.3.3 Experimental Results

The evaluation results on different deep neural ranking models of Myanmar News retrieval are represented in this section to prove that the CONV-KNRM methods can also enhance the system performance. As this result, it can be studied that the score results are significantly different in MAP and MRR because MAP measures the average precision at different recall levels, providing an overall assessment of a ranking model's ability to retrieve relevant items across the entire list and MRR calculates the average of the reciprocal ranks of the first relevant items in the ranked lists, emphasizing the model's effectiveness in placing relevant items high in the list.

Table 6.1 Statistics of Training, Testing and Validation the Myanmar News Dataset

	Number of documents	Number of sentences	Number of words
Training Set	90,607	47,964,418	1,122,242,776
Testing Set	13,940	3,784,204	93,569,044
Validation Set	13,939	2,885,793	67,448,335

The results obtained from the experiments can be seen in Figure 6.2 to 6.5. The comparisons of neural ranking performance on the Myanmar News dataset are illustrated in the following Figures: Figure 6.2 shows the performance measured by MAP, Figure 6.3 shows the performance measured by MRR, Figure 6.4 shows the performance measured by P@1, and Figure 6.5 shows the performance measured by P@3. It can be observed that CONV-KNRM performs better than other neural ranking models. This demonstrates the versatility and adaptability of CONV-KNRM in addressing the retrieving task across various contexts and the similarity scores of different deep neural ranking models using the Myanmar News dataset.

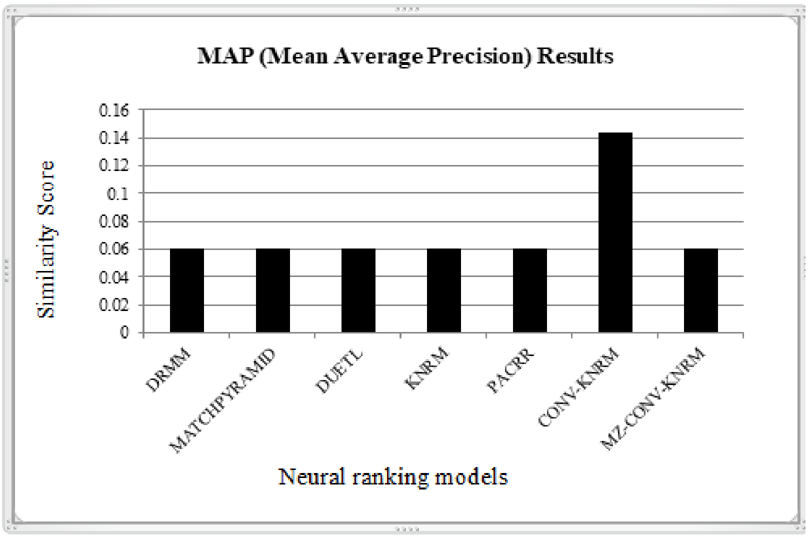

Figure 6.2 Comparison of neural ranking performance on the Myanmar News dataset measured by MAP

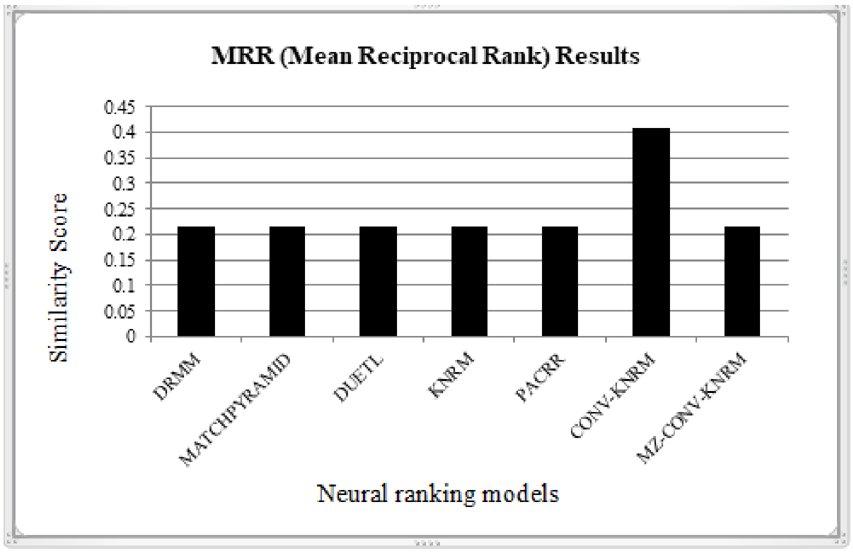

Figure 6.3 Comparison of neural ranking performance on the Myanmar News dataset measured by MRR

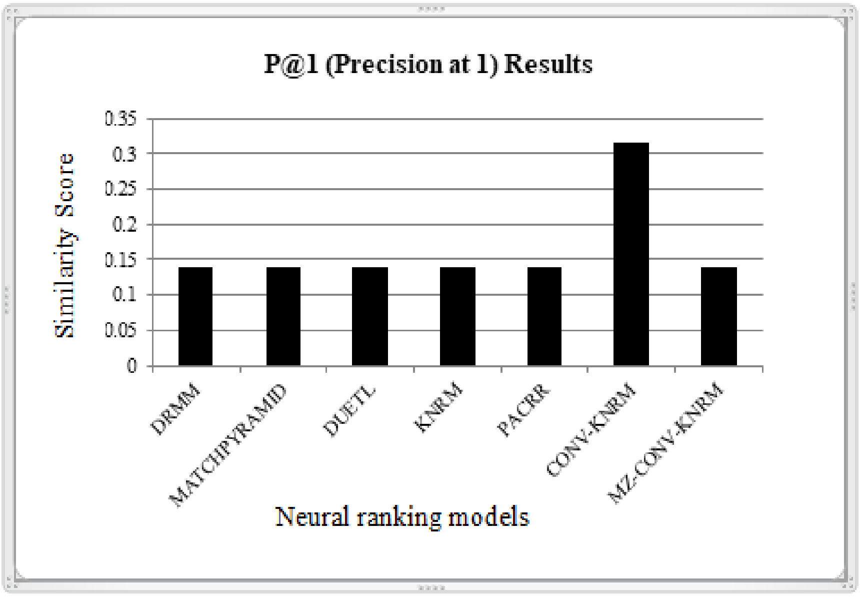

Figure 6.4 Comparison of neural ranking performance on the Myanmar News dataset measured by P@1

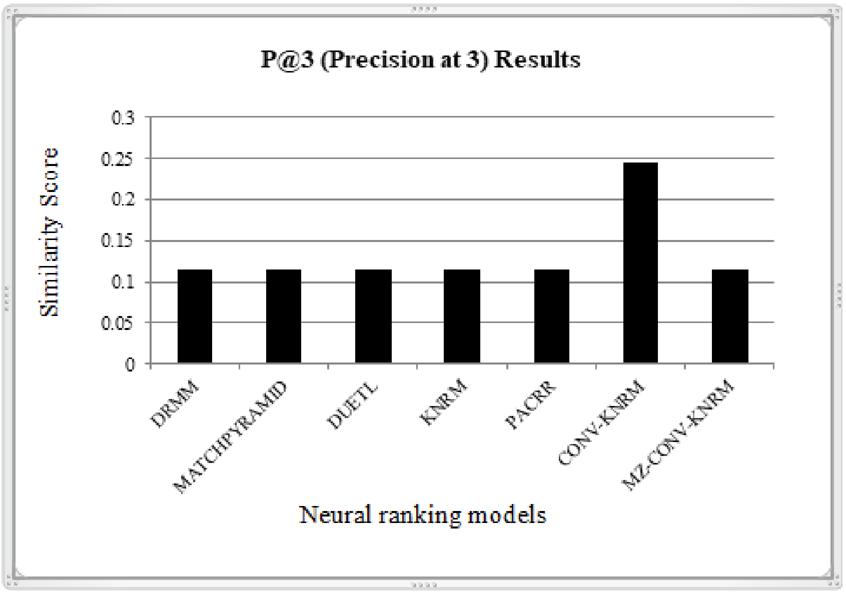

Figure 6.5 Comparison of neural ranking performance on the Myanmar News dataset measured by P@3

The best neural ranking model “CONV-KNRM” has been used as a baseline model for in this research work. The comparison was done between the fine-tuned ranking model and CONV-KNRM. During fine-tuning, it applied Vanilla-BERT fine-tuned model to improve the performance of ranking.

Table 6.2 Comparison of Performance on CONV-KNRM and Fine-tuned Models on the Myanmar News Dataset measured by Evaluation Metrics

Ranking and fine-tuned models	MAP	MRR	P@1	P@3
CONV-KNRM	0.1439	0.4066	0.3150	0.2450
Fine-tuned Model	0.1472	0.4415	0.3700	0.2433

Table 6.3 Comparison of Performance on CONV-KNRM and Fine-tuned Models on the Antique Dataset measured by Evaluation Metrics

Ranking and fine-tuned models	MAP	MRR	P@1	P@3
CONV-KNRM	0.2031	0.5902	0.4800	0.3717
Fine-tuned Model	0.2801	0.7101	0.5950	0.4967

As in Table 6.2, fine-tuning using Vanilla-BERT is found to be better than CONV-KNRM in all evaluation metrics except P@3 on the Myanmar News dataset. Specifically, the MRR results were 0.4066 and 0.4415, which is the best statistically significant different score results on other evaluation metrics (MAP, P@1 and P@3), whereas for CONV-KNRM and Vanilla-BERT. As this result, it studied that the score results are significantly different in MAP and MRR because MAP measures the average precision at different recall levels, providing an overall assessment of a ranking model's ability to retrieve relevant items across the entire list and MRR calculates the average of the reciprocal ranks of the first relevant items in the ranked lists, emphasizing the model's effectiveness in placing relevant items high in the list.

As in Table 6.3, the Antique datasets [22] is also used to see the clear performance of this ranking model in the experiments. The Antique datasets consists of 89M questions and answers–pair datasets collection. According to this experiments and results, observed that the fine-tuned model outperforms CONV-KNRM with the best score of 0.4415 in the Myanmar News dataset and 0.7101 in the Antique dataset in terms of MRR.

It can be clearly seen in Tables 6.2 and 6.3 that the fine-tuned model achieved better performance than the CONV-KNRM on the Myanmar News dataset, specifically, 0.0349 MRR value higher than the CONV-KNRM, and the fine-tuned model achieved better performance than the CONV-KNRM on the Antique dataset, specifically, 0.1199 MRR value higher than the CONV-KNRM. The experimental results provide interesting results while comparing the performance of different deep neural rankings on the Myanmar News dataset. The results suggest that the choice of fine-tuned technique can significantly impact the performance of the deep neural ranking models.

6.4 Results and Discussion

The comparisons of seven different deep neural ranking model architectures for Myanmar News retrieval are discussed in this chapter. The following is a list of the models that it experimented on Myanmar News dataset which contains 118,486 documents that are taken from online source. In this chapter, word segmentation and stop-word removal is considered in text preprocessing step. The models are trained on GPU and implemented with Python by using Python library. Performance scores of the models have been evaluated with evaluation metrics. The best result is attained from comparing different deep neural ranking model as a CONV-KNRM.

6.5 Summary

This chapter presents the experimental results regarding with building deep neural ranking models, fine-tuned model, the discussion and analysis of promising results derived from assessing the performance of the different deep neural ranking models, fine-tuned model and showing the improvement of recognizing quality by Myanmar News retrieval on Myanmar News dataset.

CHAPTER 7
CONCLUSION AND FUTURE WORKS

Summarization of the dissertation, its advantages and limitations of proposed system are described and future works will be discussed in this chapter.

7.1 Dissertation Summary

This research focused on IR for the Myanmar News dataset which contained title and contents. Different experiments have been conducted, with a wide variety of deep neural ranking models, and a fine-tuned model. It was observed that the best-performing model is fine-tuned model using Vanilla-BERT in this research. The statistical significance of the superior performance has been confirmed by comparing the results of the baseline CONV-KNRM and the fine-tuned model on the Myanmar News and Antique datasets. The experiments also indicate that the use of fine-tuning techniques can result in significant improvements in the performance of deep neural ranking models for different datasets. The experiment results suggest that fine-tuning approach can potentially be extended to other retrieval applications. Concerning further research as future work, it would be interesting to investigate the effect of adding more features to the textual data. This study adds valuable insights to the ongoing discussions within the field, paving the way for future research endeavors aimed at optimizing models to address a spectrum of challenges in IR.

In this research as objective is to build the Myanmar News dataset first for applying to IR, Myanmar News dataset for sufficient amount of training data and to investigate the text quality by using fine-tuned techniques for the retrieving the similarity score. Myanmar News dataset are created as one of main contributions. The Myanmar News dataset is built for better performance by using these seven deep neural ranking model and especially fine-tuned model is the best performance on different datasets.

This dissertation summarizes effectiveness of the Myanmar News dataset for retrieving the similarity score results on various deep neural ranking models and a fine-tuned model built with different training datasets.

7.2 Advantages

Neural models, especially those based on deep learning, can capture complex semantic relationships between queries and documents, leading to more relevant search results and improved relevance and accuracy semantic understanding. Models like BERT understand context within queries and documents, improving the retrieval of contextually appropriate information.

Neural IR systems can process natural language queries more effectively, handling nuances like synonyms, homonyms, and polysemy, enhanced query understanding in NLP. Neural networks can suggest relevant query expansions.

Neural IR systems can handle and retrieve information across text data, providing a more comprehensive search experience.

The ability to build end-to-end models that integrates various retrieval components (embedding, ranking, relevance feedback) into a single neural network, optimizing the entire process cohesively as end-to-end learning in unified models.

7.3 Limitations

Training and deploying deep neural networks require significant computational resources, including powerful GPUs or Tensor Processing Units (TPUs) and large memory capacities and high computational costs in resource intensive. The energy consumption of running large-scale neural models can be substantial, raising concerns about sustainability.

Effective training of neural retrieval models often requires vast amounts of labeled data, which can be difficult and expensive to obtain and data requirements in large datasets. The performance of Neural IR systems is heavily dependent on the quality of the training data, and biases in the data can lead to biased retrieval results and quality of data.

7.4 Future Works

The results show that, with corresponding relative improvements, the performance of models using the Myanmar News dataset beats that of models using different deep neural ranking modeling. By further optimization, these examining techniques will be used for other research.

The retrieval system of different deep neural ranking model and fine-tuned model will be implemented as extensions of the Myanmar language. End to end learning approach will be pursued in IR as future work for more improving the performance of Myanmar News Retrieval.

LIST OF ACRONYMS

IR	Information Retrieval
IT	Information Technology
DRMM	Deep Relevance Matching Model
MP	Match-Pyramid
Duetl	Duet local
KNRM	Kernelized Neural Ranking Model
PACRR	Position-Aware Convolutional Recurrent Relevance
CONV-KNRM	Convolutional Kernelized Neural Ranking Model
MZ-CONV-KNRM	MatchZoo-CONV-KNRM
Vanilla-BERT	Vanilla-Bidirectional Encoder Representations from Transformers
MAP	Mean Average Precision
MRR	Mean Reciprocal Rank
P@1	Precision at 1
P@3	Precision at 3
BM25	Best Matching 25
LTR	Learning-To-Rank
NLP	Natural Language Processing
Neural IR	Neural Information Retrieval
AI	Artificial Intelligent
QA	Question Answer
DL	Deep Learning
Word2Vec	Word-to-Vector
GloVe	Global Vectors

BERT	Bidirectional Encoder Representations from Transformers
GPT	Generative Pre-trained Transformer
RNNs	Recurrent Neural Networks
LSTM	Long Short-Term Memory
CBOW	Continuous Bag of Words
CNN	Convolutional Neural Network
ELMo	Embeddings from Language Models
Doc2Vec	Document-to-Vector
DSSM	Deep Structured Semantic Models
BOW	Bag-of-Words
VSM	Vector Space Model
LambdaMART	combines LambdaRank and Multiple Additive Regression Trees (MART)
URL	Uniform Resource Locator
RoBERTa	Robustly Optimized BERT Pre-training Approach
DNNs	Deep Neural Networks
T5	Text-to-Text Transfer Transformer
CNN	Convolutional Neural Network
IDF-based	Inverse Document Frequency-based
DistilBERT	Distilled BERT
CLM	Casual Language Model
MLM	Masked Language Model
CPU	Central Processing Unit
DPR	Dense Passage Retrieval
COIL	Contextualized Inverted List

ME-BERT	Multi-Vector Encoding from BERT
ColBERT	Contextualized Late Interaction over BERT
ANCE	Approximate nearest neighbor Negative Contrastive Estimation
STAR	Structured Transformer-based Autoencoder for Representation learning
DNN	Deep Neural Network
ANN	Artificial Neural Network
LLM	Large Language Model
LLMs	Large Language Models
GPT	Generative Pre-trained Transformer
ViTs	Vision Transformers
CNNs	Convolutional Neural Networks
ML	Machine Learning
PEFT	Parameter-Efficient Fine-Tuning
GPU	Graphics Processing Unit
LoRA	Low Rank Adaptation
QLoRA	Quantized Low Rank Adaptation
LoRA	Low Rank Adaptation
MSE	Mean Squared Error
SGD	Stochastic Gradient Descent
Adam	Adaptive Moment Estimation
RMSprop	Root Mean Square Propagation
FNNs	Feedforward Neural Networks
GRUs	Gated Recurrent Units
VAEs	Variational Auto-encoders

GANs	Generative Adversarial Networks
MLP	Multilayer Perceptron
LeToR	Learning-To-Rank
soft-TF	soft-Term Frequency
ResNet	Residual Network
NER	Named Entity Recognition
UCSY	University of Computer Studies, Yangon
NSP	Next Sentence Prediction
TPUs	Tensor Processing Units

AUTHOR'S PUBLICATIONS

[P1] Hay Man Oo, Win Pa Pa “Myanmar News Retrieval in Vector Space Model using Cosine Similarity Measure”. The 18th International Conference on Computer Applications (IEEE-ICCA2020), 27-28 Feb, 2020. **Page [214-218]**

[P2] Hay Man Oo, Win Pa Pa “Myanmar News Retrieval using Kernelized Neural Ranking Model”, Proceedings of The IEEE 21st International Conference on Computer Applications 2024, 16th March, 2024. **Page [22-27]**

[P3] Hay Man Oo, Win Pa Pa, Aye Mya Hlaing “DEEP NEURAL RANKING MODEL FOR MYANMAR NEWS RETRIEVAL”, Indian Journal of Computer Science and Engineering (**IJCSE, 2024**), e-ISSN : 0976-5166 p-ISSN : 2231-3850, Vol. 15 No. 3 May-Jun 2024. DOI: 10.21817/indjcse/2024/v15i3/241503054. **Page [301-308]**

Bibliography

[1] M. Bendersky, W. B. Croft, and Y. Diao. “Quality-biased ranking of web document”. 2011. In Proc.WSDM, pp. 95–104.

[2] L. S. Blackford, A. Petitet, R. Pozo, K. Remington, R. C. Whaley, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, et al. “An Updated Set of Basic Linear Algebra Subprograms (BLAS)”. 2002. ACM Transanctions on Mathematical Software, 28(2): 135–151.

[3] J. Bromley, I. Guyon, Y. LeCun, E. Sackinger, and R. Shah. Signature Verification ¨ using a ”Siamese” Time Delay Neural Network. 1993. In Proc. NIPS.

[4] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. “Learning to Rank Using Gradient Descent”. 2005. In Proc. ICML, p. 89–96.

[5] C. J. C Burges, “From RankNet to LambdaRank to LambdaMART: An overview”. 2010. Microsoft Research Technical Report.

[6] S. Buttcher, C. Clarke, and G. V. Cormack. “Information Retrieval: Implementing and Evaluating Search Engines”. 2010. The MIT Press.

[7] B. B. Cambazoglu, R. A. Baeza-Yates, “Scalability Challenges in Web Search Engines”. 2015. Morgan & Claypool Publishers.

[8] Wei. Cheng Chang, Felix X. Yu, Yin-Wen Chang, Yiming Yang, Sanjiv Kumar. “Pre-training Tasks for Embedding-based Large-scale Retrieval”. Feb-20. https://www.researchgate.net/publication.

[9] Ruey-Cheng Chen, Luke Gallagher, Roi Blanco, J. Shane Culpepper. “Efficient Cost-Aware Cascade Ranking in Multi-Stage Retrieval”. Aug-17. DOI: 10.1145/3077136.3080819, the 40th International ACM SIGIR Conference

[10] Andrew M Dai, Quoc V Le. “Semi-supervised sequence learning”. 2015. In Advances in neural information processing systems, pages 3079–3087.

[11] Z. Dai, C. Xiong, J. Callan, and Z. Liu, "Convolutional Neural Networks for Soft-Matching N-Grams in Ad-Hoc Search", 2018. In Proc. WSDM, p. 126–134.

[12] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding", 2019. In Proc. NAACL, pp. 4171– 4186.

[13] William B. Dolan, Chris Brockett, "Automatically constructing a corpus of sentential paraphrases", 2005. In Proceedings of the Third International Workshop on Paraphrasing (IWP2005).

[14] Y. Fan, J. Guo, Y. Lan, J. Xu, C. Zhai, and X. Cheng, "Modeling Diverse Relevance Patterns in Ad-Hoc Retrieval", 2018b. In Proc. SIGIR, p. 375–384.

[15] L. Gao, Z. Dai, and J. Callan, "COIL: Revisit Exact Lexical Match in Information Retrieval with Contextualized Inverted List", 2021. In Proc. NAACL-HLT, pp. 3030–3042.

[16] Luyu Gao, Zhuyun Dai, Tongfei Chen, Zhen Fan Benjamin Van Durme, Jamie Callan, "Complement Lexical Retrieval Model with Semantic Residual Embeddings", 2021. arXiv:2004.13969v3 [cs.IR]

[17] F. C. Gey, "Inferring Probability of Relevance Using the Method of Logistic Regression", 1994. In Proc. SIGIR, p. 222–231

[18] D. Gillick, S. Kulkarni, L. Lansing, A. Presta, J. Baldridge, E. Ie, and D. Garcia-Olano, "Learning Dense Representations for Entity Retrieval", 2019. In Proc. CoNLL, pp. 528– 537

[19] Jiafeng Guo, Yixing Fan, Xiang Ji and Xueqi Cheng, "Match-Zoo: A Learning, Practicing, and Developing System for Neural Text Matching", In Proceedings of the 42nd Int'l ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'19), July 21–25, 2019, Paris, France. ACM, NY, NY, USA.

[20] J. Guo, Y. Fan, L. Pang, L. Yang, Q. Ai, H. Zamani, C. Wu, W. B. Croft,

and X. Cheng. “A deep look into neural ranking models for information retrieval”, 2020. Information Processing & Management, 57(6): 1–20.

[21] J. Guo, Y. Fan, Q. Ai, and W. B. Croft, “A Deep Relevance Matching Model for Ad-Hoc Retrieval”, 2016. In Proc. CIKM, p. 55–64

[22] Helia Hashemi, Mohammad Aliannejadi, Hamed Zamani, and W. Bruce Croft, “ANTIQUE: A Non-Factoid Question Answering Benchmark”, 19-Aug-19. ArXiv:1905.08957v2 [cs.IR]

[23] Jeremy Howard, Sebastian Ruder, “Universal language model fine-tuning for text classification”, 2018. In ACL. Associations for Computational Linguistics.

[24] B. Hu, Z. Lu, H. Li, and Q. Chen, “Convolutional Neural Network Architectures for Matching Natural Language Sentences”, 2014. In Proc. NIPS, p. 2042–2050.

[25] P.-S Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck, “Learning Deep Structured Semantic Models for Web Search Using Clickthrough Data”, 2013. In Proc. CIKM, p. 2333–2338

[26] K. Hui, A. Yates, K. Berberich, and G. de Melo, “PACRR: A Position-Aware Neural IR Model for Relevance Matching”, 2017. In Proc. EMNLP, pp. 1049–1058

[27] K. Hui, A. Yates, K. Berberich, and G. de Melo, “Co-PACRR: A Context-Aware Neural IR Model for Ad-Hoc Retrieval”, 2018. In Proc. WSDM, p. 279–287

[28] S. Humeau, K. Shuster, M.-A. Lachaux, and J. Weston, “Real-time inference in multi-sentence tasks with deep pretrained transformers”, 2019. DeepAI Technical Report.

[29] T. B. Johnson, C. Guestrin, “Training Deep Models Faster with Robust, Approximate Importance Sampling”, 2018. In Proc. NeurIPS.

[30] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, “Bag of Tricks for

Efficient Text Classification”, 2017. In Proc. EACL, pp. 427–431.

[31] Reza Karimpour, Amineh Ghorbani, Azadeh Pishdad, Mitra Mohtarami, Abolfazl Aleahmad, Hadi Amiri, Farhad Oroumchian, “Using part of speech tagging in Persian information retrieval”, Jan-08, https://www.researchgate.net/.

[32] V. Karpukhin, B. Oguz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W.-t. Yih, “Dense Passage Retrieval for Open-Domain Question Answering”, 2020. In Proc. EMNLP, pp. 6769–6781.

[33] A. Katharopoulos, F. Fleuret, “Not All Samples Are Created Equal: Deep Learning with Importance Sampling”, 2018. In Proc. ICML.

[34] Tom Kenter, Alexey Borisov, Christophe Van Gysel, Mostafa Dehghani, Maarten de Rijke, Bhaskar Mitra, “Neural Networks for Information Retrieval”, 2017. conference: SIGIR ’17; August 07-11, 2017; Shinjuku, Tokyo, Japan.

[35] Muhammad Hammad Khan, “Neural IR Models”, 18-Jul-22. https://medium.com/.

[36] O. Khattab, M. Zaharia, “ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT”, 2020. In Proc. SIGIR, p. 39–48.

[37] PM. Lavanya, E. Sasikala, “Deep Learning Techniques on Text Classification Using Natural Language Processing (NLP) In Social Healthcare Network: A Comprehensive Survey”, May-21, DOI: 10.1109/ICSPC51351.2021.9451752. Conference: 2021 3rd International Conference on Signal Processing and Communication (ICPSC).

[38] Y. LeCun, Y. Bengio, “Convolutional Networks for Images, Speech, and Time Series”, 1998. In The Handbook of Brain Theory and Neural Networks, pp. 255–258. MIT Press.

[39] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, “BART: Denoising Sequence-to-Sequence

Pre-training for Natural Language Generation, Translation, and Comprehension", 2020. In Proc. ACL, pp. 7871–7880.

[40] Vladislav Lialin, "Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning", 2024. https://ar5iv.labs.arxiv.org/html/2303.15647

[41] T.-Y. Liu, "Learning to Rank for Information Retrieval", 2009. Foundations and Trends in Information Retrieval, 3(3): 225–331.

[42] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A Robustly Optimized BERT Pretraining Approach", 2019. ArXiv, abs/1907.11692.

[43] Y. Luan, J. Eisenstein, K. Toutanova, and M. Collins, "Sparse, Dense, and Attentional Representations for Text Retrieval", 2021. Transactions of the Association for Computational Linguistics, 9: 329–34.

[44] S. MacAvaney, A. Yates, A. Cohan, and N. Goharian, "CEDR: Contextualized Embeddings for Document Ranking", 2019.In Proc. SIGIR, p. 1101–1104.

[45] S. MacAvaney, F. M. Nardini, R. Perego, N. Tonellotto, N. Goharian, and O. Frieder, "Efficient Document Re-Ranking for Transformers by Precomputing Term Representations", 2020b. In Proc. SIGIR, pp. 49–58.

[46] C. Macdonald, R. L. T. Santos, and I. Ounis, "The whens and hows of learning to rank for web search", 2012. Information Retrieval, 16(5): 584–628.

[47] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed Representations of Words and Phrases and their Compositionality", 2013. In Proc. NIPS.

[48] Bhaskar Mitra, Fernando Diaz, and Nick Craswell, "Learning to Match using Local and Distributed Representations of Text for Web Search", 2016, In WWW.

[49] R. Nogueira, K. Cho, "Passage Re-ranking with BERT", 2019. arXiv

1901.04085.

[50] R. Nogueira, W. Yang, K. Cho, and J. Lin, “Multi-stage document ranking with BERT”, 2019a. arXiv 1910.14424.

[51] R. Nogueira, Z. Jiang, R. Pradeep, and J. Lin, “Document Ranking with a Pretrained Sequence-to-Sequence Model”, 2020. In Proc. EMNLP, pp. 708–718.

[52] Win Pa Pa, Ye Kyaw Thu, Andrew Finch, and Eiichiro Sumita, “Word boundary identification for Myanmar text using conditional random fields”, 2008. In International Conference on Genetic and Evolutionary Computing.

[53] L. Pang, Y. Lan, J. Guo, J. Xu, and X. Cheng, “A Study of MatchPyramid Models on Ad-hoc Retrieval”, 2016. arXiv 1606.04648.

[54] L. Pang, Y. Lan, J. Guo, J. Xu, and X. Cheng, “DeepRank: A New Deep Architecture for Relevance Ranking in Information Retrieval”, 2017. In Proc. CIKM, pp. 257–266.

[55] Ankur P Parikh, Oscar Tackstrom, Dipanjan Das, and Jakob Uszkoreit, “A decomposable attention model for natural language inference”, 2016. In EMNLP.

[56] J. Pennington, R. Socher, and C. D. Manning, “GloVe: Global Vectors for Word Representation”, 2014. In Proc. EMNLP, pp. 1532–1543.

[57] Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer, “Deep contextualized word representations”, 2018a. In NAACL.

[58] F. Petroni, T. Rocktaschel, S. Riedel, P. Lewis, A. Bakhtin, Y. Wu, and A. Miller, “Language Models as Knowledge Bases?”, 2019. In Proc. EMNLP-IJCNLP, pp. 2463–2473.

[59] J. M. Ponte, W. B. Croft, “A Language Modeling Approach to Information Retrieval”, 1998. In Proc. SIGIR, pp. 275–281.

[60] Samuel R, Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning, “A large annotated corpus for learning natural language inference”, 2015. In EMNLP. Association for Computational Linguistics.

[61] A. Radford, K. Narasimhan, “Improving Language Understanding by Generative Pre-training”, 2018. OpenAI Techical Report.

[62] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, “Language Models are Unsupervised Multitask Learners”, 2019. OpenAI Technical report.

[63] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer”, 2020. Journal of Machine Learning Research, 21(140): 1–67.

[64] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang, “Squad: 100,000+ questions for machine comprehension of text”, 2016. In Proceedings of the Conference on Empirical Methods in Natural Language Processing, pages 2383–2392.

[65] S. E. Robertson, “The Probability Ranking Principle in IR”, 1977. Journal of Documentation, 33(4): 294–304.

[66] S. Robertson, H. Zaragoza, “The Probabilistic Relevance Framework: BM25 and Beyond”, 2009. Foundations and Trends in Information Retrieval, 3(4): 333–389.

[67] G. Salton, A. Wong, and C. S. Yang, A Vector Space Model for Automatic indexing”, 1975. Communications of the ACM, 18(11): 613–620.

[68] Erik F Tjong Kim, Sang, Fien De Meulder, “Language-independent named entity recognition”, 2003. Introduction to the conll - 2003 shared task. In CoNLL.

[69] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter”, 2019. In Proc. 5th Workshop on Energy Efficient Machine Learning and Cognitive

Computing @ NeurIPS 2019.

[70] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi, "Bidirectional attention flow for machine comprehension", 2017. In ICLR.

[71] Prashant Sharma, "Dependency Parsing in Natural Language Processing with Examples", Aug, 2023. https://www.analyticsvidhya.com/

[72] Simple Sharma, Supriya P. Panda, "Efficient information retrieval model: overcoming challenges in search engines-an overview", November 2023. Indonesian Journal of Electrical Engineering and Computer Science Vol. 32, No.2, pp.925~932 ISSN:2502-4752, DOI: 10.11591/ijeecs.v32.i2.pp925-932.

[73] N. Tonellotto, C. Macdonald, and I. Ounis, "Efficient query processing for scalable web search", 2018. Foundations and Trends in Information Retrieval, 12(4–5): 319–492.

[74] Nicola Tonellotto, "Lecture Notes on Neural Information Retrieval", Sep-22. arXiv:2207.13443v2 [cs.IR]

[75] Mohamed Trabels, Zhiyu Chen, Brian D. Davison, Jef Hefin, "Neural ranking models for document retrieval", Oct-21. Information Retrieval Journal https://doi.org/10.1007/s10791-021-09398-0.

[76] J. Urbanek, A. Fan, S. Karamcheti, S. Jain, S. Humeau, E. Dinan, T. Rocktaschel, ¨ D. Kiela, A. Szlam, and J. Weston, "Learning to speak and act in a fantasy text adventure game", 2019. In Proc. EMNLP-IJCNLP, pp. 673–683.

[77] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman, "Glue: A multi-task benchmark and analysis platform for natural language understanding", 2018a, In Proceedings of the EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP, pages 353–355.

[78] Adina Williams, Nikita Nangia, and Samuel R Bowman, "A broad-

coverage challenge corpus for sentence understanding through inference", 2018. In NAACL.

[79] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, et al, "Google's neural machine translation system: Bridging the gap between human and machine translation", 2016. arXiv preprint arXiv:1609.08144.

[80] E. P. Xing, A. Y. Ng, M. I. Jordan, and S. Russell, "Distance Metric Learning, with Application to Clustering with Side-Information", 2002. In Proc. NIPS, pp. 521–528.

[81] L. Xiong, C. Xiong, Y. Li, K.-F. Tang, J. Liu, P. Bennett, J. Ahmed, and A. Overwijk, "Approximate Nearest Neighbor Negative Contrastive Learning for Dense Text Retrieval", 2021. In Proc. ICLR.

[82] C. Xiong, Z. Dai, J. Callan, Z. Liu, and R. Power, "End-to-End Neural Ad-Hoc Ranking with Kernel Pooling", 2017. In Proc. SIGIR, p. 55–64.

[83] J. Zhan, J. Mao, Y. Liu, J. Guo, M. Zhang, and S. Ma, "Optimizing Dense Retrieval Model Training with Hard Negatives", 2021b. In Proc. SIGIR, pp. 1503–1512.