


People's Democratic Republic of Algeria

Ministry of Higher Education and Scientific Research

AKLI MOHAND OULHADJ UNIVERSITY -BOUIRA-

Faculty of Sciences and Applied Sciences

Computer Science Department

Master Thesis

In Computer Science

Speciality:*Information Systems and Software Engineering -ISIL-*

Theme

Transformer-based Question Answering Model for the Biomedical Domain

---

Supervised by

- Dr. AID Aicha

Realized by

- HADDOUCHE Ahcene
- RABIA Ikram

2022/2023

Acknowledgments

First and foremost, we extend our heartfelt gratitude to our dedicated supervisor, Dr. Aid.A, whose unwavering efforts have played a pivotal role in the successful completion of this thesis. We are profoundly grateful for her invaluable guidance, unwavering encouragement, unfailing availability, and genuine interest in our work.

We would also like to express our sincere appreciation to all the esteemed professors in our academic career. Their profound knowledge, expertise, and passion for teaching have significantly contributed to our education and professional growth.

We cannot overstate the immeasurable support and involvement of our beloved families and dear ones throughout the entire process of bringing this work to life. Their constant encouragement, both direct and indirect, has been an invaluable source of motivation and inspiration for us.

We extend our sincerest thanks to all the individuals mentioned above, whose unwavering support made this thesis possible. Your encouragement was instrumental in our success, and we could not have done it without you.

-Rabia Ikram

-Haddouche Ahcene

Dedications

À mes parents, qui ont été mes piliers tout au long de ce parcours, je dédie ce mémoire. Votre amour inconditionnel et votre soutien indéfectible ont été ma source d'inspiration et ma force motrice. Ma chère MANII, qui nous a récemment quittée, disait toujours : "Le savoir est une lumière qui brille éternellement." Je sais qu'elle veille sur moi là-haut, et si elle était avec moi aujourd'hui, elle serait fière. Je lui dédie spécialement ce travail. Merci d'avoir cru en moi et de m'avoir boosté à réaliser mes rêves.

À ma frangine Fadou, mon p'tit frangin Midou et mon neveux chéri Mirou, j'ai vraiment pris la tête avec vous pendant tous ces mois, entre la pression et les galères. Merci quand même de m'avoir supporté, Je vous aime les gars !

À mes potes les plus proches, qui ont partagé les kifs et les galères de cette aventure, je vous suis grave reconnaissante. Votre présence, votre écoute et vos encouragements ont Ambiancé mon chemin et m'ont littéralement motivé.

À mon binôme Ahcene, qui a vraiment contribué dans la réalisation de ce projet et avec qui j'ai tout partagé pour mener à bien cette thèse. We did it finally!

Ce travail est le fruit de l'amour, de l'encouragement et du soutien de ces personnes hors normes. Leur présence a rendu cette expérience enrichissante et inoubliable. Je leur suis infiniment reconnaissante pour leur rôle essentiel dans cette réalisation.

-Rabia Ikram

Dedications

I dedicate this modest work to

My dear parents for all their sacrifices, unwavering love, support, and endless patience, I ask God to provide you with happiness and a long life.

My dear sisters and brothers always gave me hope and believed in my abilities.

and all my family for their encouragement throughout my academic career.

A special dedication to my dear partner Ikram Rabia for her patience and hard work despite difficult circumstances.

To all my colleagues and friends. May Allah bless you with success.

To all my teachers since my first years of studies.

To all those who feel dear to me and whom I have failed to mention.

-Ahcene Haddouche

Abstract

Motivation

Question Answering (QA) is a highly focused topic in the field of Natural Language Processing (NLP). Recent progress in neural network models and the availability of large datasets like SQuAD have played a significant role in improving performance in open domains. However, there remains a need to further effectively implement these systems in more specific domains, especially in the biomedical field, to help medical practitioners provide accurate solutions for inquiries related to medicine and healthcare, including specific subjects such as the COVID-19 disease. Fortunately, recent models, such as transformers, have opened up avenues and modern techniques for developing accurate systems.

Aims

In this work, we aim to leverage transformer models and Transfer Learning to effectively train models in the biomedical domain. By taking a pre-trained model for Question Answering tasks and further fine-tuning it on specific domains, we enhance the system's performance in the biomedical domain. Our ultimate goal is to develop a QA system specifically tailored for COVID-19 QA.

Results

We have trained BERT and RoBERTa models on the COVID-QA dataset and achieved competitive results on COVID-19 QA. Our RoBERTa model achieved an Exact Match (EM)/F1 score of 0.38 / 0.64, respectively, on COVID-QA, indicating successful performance in COVID-19 QA.

Keywords

COVID-QA, COVID-19, Question Answering systems, transformers, RoBERTa, BERT.

Résumé

Motivation

Le Question Answering (QA) est un sujet trés ciblé dans le domaine du Traitement Automatique du Langage Naturel (TALN). Les progrès récents dans les modèles de réseaux neuronaux et la disponibilité de vastes ensembles de données (dataset), tels que SQuAD, ont joué un rôle significatif dans l'amélioration des performances dans des domaines ouverts. Cependant, il est essentiel d'optimiser davantage la mise en œuvre de ces systèmes dans des domaines plus spécifiques et restreints, notamment dans le domaine biomédical, afin d'assister de manière précise les professionnels de la santé dans la fourniture de réponses exactes aux questions relatives à la médecine et aux soins de santé, y compris des sujets spécifiques tels que la COVID-19. Avantageusement, l'émergence de modèles récents tels que les transformers a ouvert de nouvelles perspectives prometteuses et a introduit des techniques modernes pour développer des systèmes d'une précision accrue.

Objectifs

Dans cette étude, notre objectif consiste à exploiter les modèles de transformers et l'apprentissage par transfert pour entraîner de manière efficace des modèles dans le domaine biomédical. En utilisant un modèle pré-entraîné pour les tâches de question-réponse et en l'ajustant davantage pour des domaines spécifiques, nous améliorons les performances du système dans le domaine biomédical. Notre objectif ultime est de développer un système de Question Answering spécifiquement adapté pour répondre aux interrogations relatives à la COVID-19.

Résultats

Nous avons procédé à l'entraînement de deux modèles, à savoir BERT et RoBERTa, en utilisant le dataset COVID-QA, et nous avons obtenu des résultats remarquable en matière de réponse aux questions relatives à la COVID-19. Notre modèle RoBERTa a obtenu un score d'Exact Match (EM) de 0,38 et un score F1 de 0,64 sur le dataset COVID-QA, ce qui témoigne d'une performance réussie dans la réponse aux questions spécifiques à la COVID-19.

Mots-clés

COVID-QA, COVID-19, Question-Answering systems, transformers, RoBERTa, BERT.

Contents

List of Figures

List of Tables

List of Acronyms

QA	Question-Answering.
NLP	Natural Language Processing.
IE	Information Extraction.
KB	Knowledge Base.
IR	Information Retrieval.
MRC	Machine Reading Comprehension.
AI	Artificial Intelligence.
ML	Machine Learning.
DL	Deep Learning.
MLM	Masked Language Model.
NSP	Next Sentence Prediction.
MSE	Mean Squared Error.

General introduction

Every year, biomedical academic communities publish a large number of scholarly articles. According to the American National Library of Medicine (NLM), as of December 22, 2022, over 1.5 million new citations were added to MEDLINE in that year [12]. With such a high volume of new information, it becomes increasingly challenging for medical professionals to stay up to date with recent developments. To address this issue, NLP systems that can identify and return relevant information in a human-readable format are crucial. QA systems are one type of tool that can be used to address this need.

QA systems can automatically extract information from text, such as scientific articles, and present it in a way that is easy for users to understand. This can help medical professionals stay up-to-date on the latest research and make better decisions about patient care. However, developing accurate QA systems can be difficult due to the limited diversity and complexity of the data, especially in specialized tasks and topics, such as COVID-19, where relevant datasets are small and often scarce.

One solution is to fine-tune pre-trained transformer-based QA systems. Transformers are a type of neural network that has demonstrated efficacy across various NLP tasks, including QA. They are widely preferred over other types of models due to their ease of use and efficiency in training. Additionally, they offer a broad scope for research and continuous advancements to enhance their performance.

In this study, we aim to demonstrate the application of transformer models in a biomedical domain QA scenario. Our focus will be on utilizing NLP and Deep Learning (DL) techniques to train BERT and RoBERTa models to answer questions related to biomedical specifically COVID-19.

The proposed models will be built upon the BERT and RoBERTa architectures, which

have been pre-trained for language comprehension. Fine-tuning these pre-trained models using the SQuAD dataset allows them to acquire knowledge of general QA patterns. As well, further fine-tuning on smaller datasets can enhance their performance in specific domains, such as COVID-19, through the utilization of Transfer Learning.

This thesis includes four chapters:

Chapter One provides a comprehensive overview of QA systems, techniques, architectures, and the classification of QA systems based on various criteria, including approach, application domain, question type, and target task. It also explores a list of some of the well-known datasets, as well as the evaluation metrics used to assess their performance.

Chapter Two introduces some preliminaries, delves into the advancements in Artificial Intelligence (AI), with a specific focus on DL and neural networks, and explores the application of neural networks in NLP, especially Transformers. The chapter provides a Thorough analysis of related works.

In Chapter Three, we will discuss the proposed approach for solving the problem. We will present the datasets we used, the models we worked on, and the architecture of our system.

The final chapter presents an analysis and discussion of the research findings from the preceding chapters. We provide an overview of the experimental setup, including the software and hardware employed. Furthermore, we discuss the results, showing the performance improvements achieved through the model training.

Chapter 1

Question Answering

1.1 Introduction

QA is a field of study that involves a combination of three interrelated yet distinctive areas: Information Retrieval (IR), Information Extraction (IE), and Natural Language Processing (NLP). The primary aim of QA systems is to retrieve accurate answers to natural language queries, as opposed to search engines that provide long documents relevant to a specific topic. QA focuses on providing extracted short and precise responses that are specifically tailored to the user's requests.

In the next sections, we will provide a general overview of QA systems by discussing their definition, techniques, architectures, and evaluation metrics. We will also examine their classification according to several criteria, such as the approach used, the application domain, the question type, and the target tasks.

1.2 General overview of Question Answering

1.2.1 General overview

Human beings are by nature curious and worthy of knowledge. They tend to ask limitless questions related to different topics. Nowadays, in order to satisfy this positive curiosity and bring informative answers to their knowledge needs, they rely on various search engines to interrogate them without really knowing the process that occurs behind the screen. One of the systems that handles this process is called the QA System.

QA systems refer to computer-based systems that have the ability to provide automatic

responses to questions in any language asked by a user. Their goals are to understand the meaning of the question, retrieve relevant information from a large corpus of documents, and present an accurate and concise answer to its requester. QA systems mostly make use of IR and NLP techniques to achieve these goals. Their performance is typically evaluated based on their accuracy and ability to provide relevant and useful information to users [13].

Here is a simple example that illustrates the QA process described above [14]:

Document	"... The novel virus was first identified in an outbreak in the Chinese city of Wuhan in December 2019. Attempts to contain it there failed, allowing the virus to spread to other areas of Asia....."
Question	Where was the first identified case of the coronavirus disease?
Answer	Wuhan.

1.2.2 Brief history of QA Systems

QA systems have a long history, with roots dating back to the 1960s.
The earliest QA system, BASEBALL (Green Jr et al., 1961), was developed specifically for QA relating to baseball games played in the American League over a season, including statistics and more. The system used a rule-based language model for "decoding" natural text generation and accessed a baseball relational database for authentic responses.

By 1973, significant enhancements in syntactical and semantic parsers had improved the capabilities of QA systems, allowing for greater freedom of expression in questioning. The best-known example of this was LUNAR, originally designed to aid lunar geologists in accessing and evaluating chemical composition data on lunar rocks and soil, as a result of the Apollo moon mission.

In the 1970s and 1980s, researchers began to explore the use of NLP techniques for QA. NLP enabled the development of systems that could understand the meaning of questions and match them to relevant information stored in databases. These systems were more reliable than their rule-based predecessors but still had significant limitations.

The 1990s saw the rise of knowledge-based QA systems, which used semantic networks

and ontologies to represent knowledge. These systems were able to reason about the relationships between different pieces of information and provide more accurate answers. However, they struggled with questions that required complex reasoning or inference.

The early 2000s saw the development of web-based QA systems, which used the vast amounts of digital data available online to provide answers to user queries. These systems were able to provide much more detailed and accurate answers than their predecessors, but they struggled with understanding the context and meaning of questions.

2010s-present: In recent years, QA systems have continued to evolve rapidly, with advances in DL and neural networks leading to major breakthroughs in NLP and machine comprehension. Today, cutting-edge QA systems are able to understand complex questions and provide highly accurate, detailed answers across a wide range of domains and subject areas [13].

1.3 Generic architecture of QA systems

As shown in Figure 1.1, a typical QA system consists of three distinct modules namely, Question Processing, document processing, and Answer Processing [15].

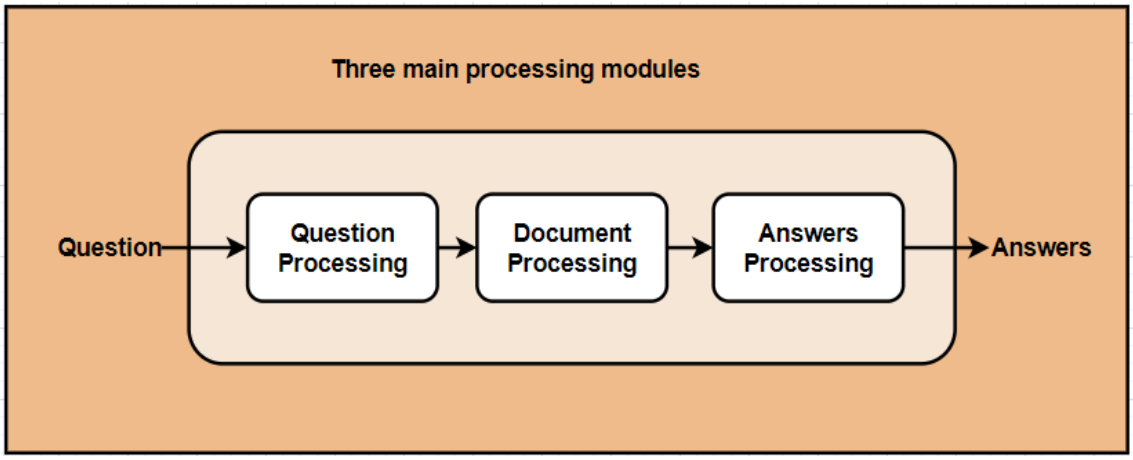


Figure 1.1: Three main Processing Modules of QA

1.3.1 Question Processing Module

Question Processing is the module that takes questions as input, then, identifies the **question analysis**, classifies the **question type**, derives the expected **answer type**, and finally reformulates the original question into many similar ones. The Question

Processing module is thus required to analyze, classify, and reformulate the question [15], as shown in the figure 1.2 below.

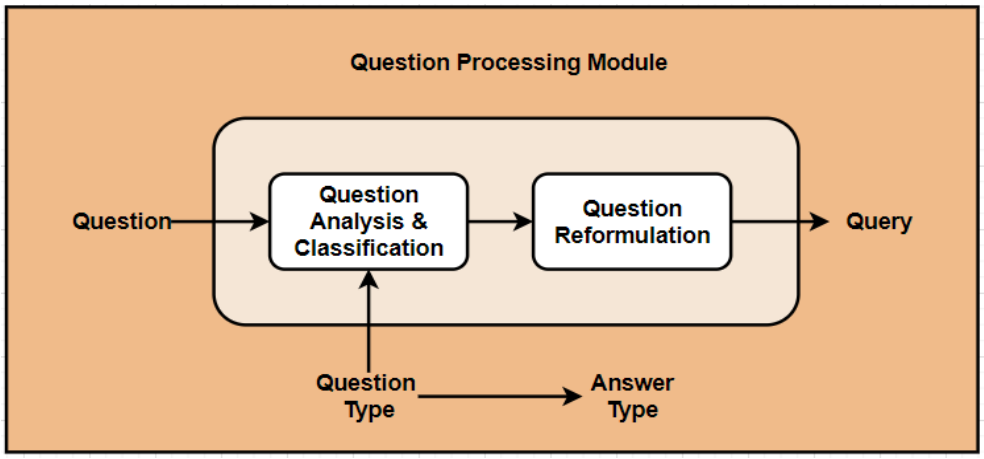


Figure 1.2: Question Processing Module

Question Type Classification

Seeking to answer a given question, different strategies can be considered based on the question type. For instance, one of these types is the so-called factoid questions (or wh-questions: "who," "what," "where," "when" and "how") [15].

In fact, the question type classification directly affects the answers, just as the study results made by Dan Moldovan et aL. in [16] show that 36.4% of errors happen due to misclassification. This is due to the fact that defining the question type can impose limitations on the characteristics of the expected response, which allows other modules accurately locate and verify a response [17].

Question analysis

Question analysis, also called "Question Focus". Knowing the **question type** alone is not enough to find answers to all questions. Indeed, factoid "What" questions in particular can be quite ambiguous in the information they ask for. The same applies to many other types of questions. To manage this ambiguity, an extra component that extracts the question's focus is critical [18].

A **Question Focus** is a word or a sequence of words that define the question and disambiguate it by indicating what the question is looking for. For example, in the

question, "What is the best programming language to use in deep learning projects?", the best programming language and Deep learning are the focus [19].

Question Reformulation

After identifying the "focus" and the "question type," the module passes a list of keywords (a query) to the IR component in the document processing module. The process of extracting keywords could be performed with the help of standard techniques like named-entity recognition, part-of-speech taggers, keyword and keyphrase extraction algorithms, etc [15].

Answer type detection

Answer type detection takes as a foundation the mapping of the question classification [18]. After a question has been categorized, a simple rule-based mapping will be used to determine the potential answer types [15].

1.3.2 Document Processing Module

The document Processing module in QA systems is also referenced by the Passage Retrieval module, which applies several techniques to find the relevant documents for the query generated in the previous module based on the indexed set of documents [18]. The IR system may return a large number of documents. Therefore, the main goal of the Document Processing module is to create a set of **candidate passages** that contain the answer(s), and in pursuit of this objective, the Document Processing module is required to:

- Retrieve a collection of ranked documents that are relevant to the query.

- Filter the returned documents from the retrieval system to minimize the number of candidate documents and the size of candidate text contained in each document.

- Out of the retrieved and selected documents, **candidate answers** is extracted, which constitutes the input for the Answer Processing module, as shown in the figure 1.3 below.

The reason for dividing the documents into shorter passages is to increase system speed. The response time of a QA system is extremely necessary due to the interactive nature of QA. [15].

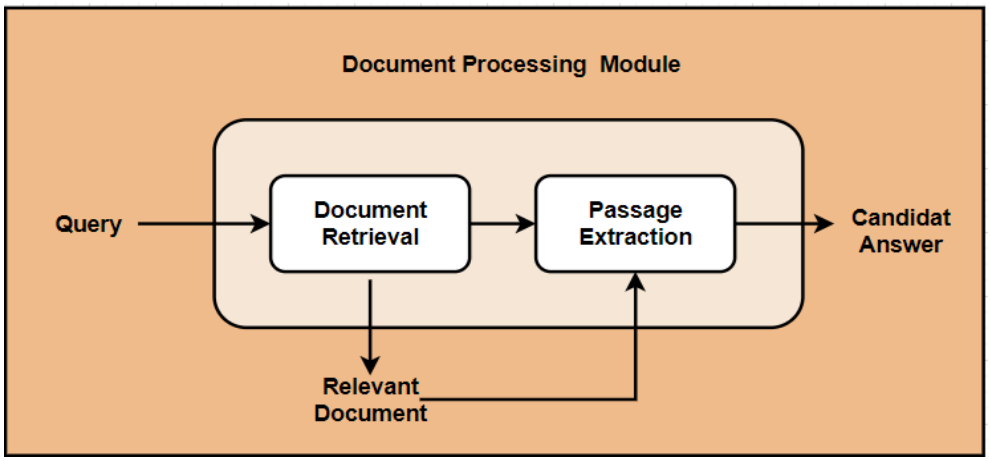


Figure 1.3: Document Processing module

1.3.3 Answer Processing Module

The Answer Processing module is the final stage of the QA system and is responsible for identifying, extracting, and validating the candidate answers passed from the Document Processing module. Thus, the Answer Processing module needs to: identify the answer candidates; Extract and validate the final answer [15].

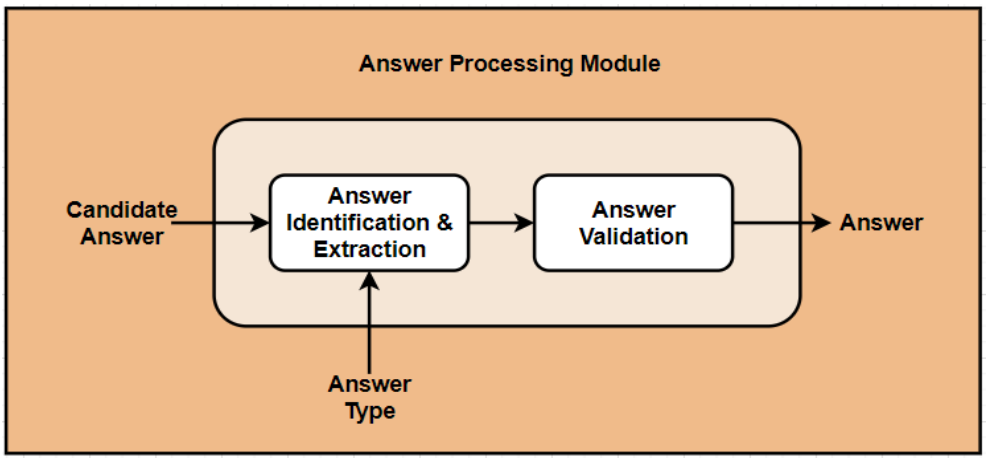


Figure 1.4: Answer Processing Module

Answer Identification

The answer type determined during question processing is critical in determining the answer. Due to the fact that the answer type is usually not explicit in the question or the answer, a parser is required to recognize named entities (e.g., names of persons and organizations, monetary units, dates, etc.). Using a part-of-speech tagger (for example, Brill

tagger) can also aid in the recognition of answer candidates within identified paragraphs. A candidate answer is created by recognizing the answer type returned by the parser. The extraction and validation of the answer are based on a set of heuristics [15].

Answer Extraction

After identifying the correct answer, the shallow parsing technique is leveraged to extract only the relevant word or phrase that corresponds to the question [2].

Answer Selection

Confidence in the correctness of an answer can be increased in a number of ways. One way is to use a lexical resource to validate that a candidate's answer was of the correct answer type. Specific knowledge sources can also be used as a second opinion to validate answers to questions within specific domains. This allows candidate answers to be sanity checked before being presented to a user [15].

1.4 Classification of QA Systems

Several works have classified QA systems, based on different points of view and criteria. For example, Jurafsky et al. [1] have classified them based on the used data source. Yogish, Manjunath et al. have classified them based on the knowledge source and the techniques used [20], whereas Pundge et al[21] have classified them based on the application domain and the techniques used in the system.

In [17], the authors Mishra and Jain have listed eight criteria in their classification, as follows: application domain, question type, query analysis, techniques used for answer retrieval, features of databases, types of matching functions, databases, and forms of answer generation.

In the following sections, we will describe and detail the four most important criteria that we have identified in the literature, which are: the application domain, the question type, the form of output answer, and the approaches used.

1.4.1 Classification based on Approaches

Besides the main architecture, each QA system can be classified by an implementation approach to one of the following four types [22]:

Information Retrieval

This approach uses a search engine to locate the answer to the query. The QA system retrieves documents that contain relevant information and extracts the answer from them. The system may use techniques such as keyword matching, document ranking, and text similarity to find the most relevant documents, as shown in figure 1.5 [22].

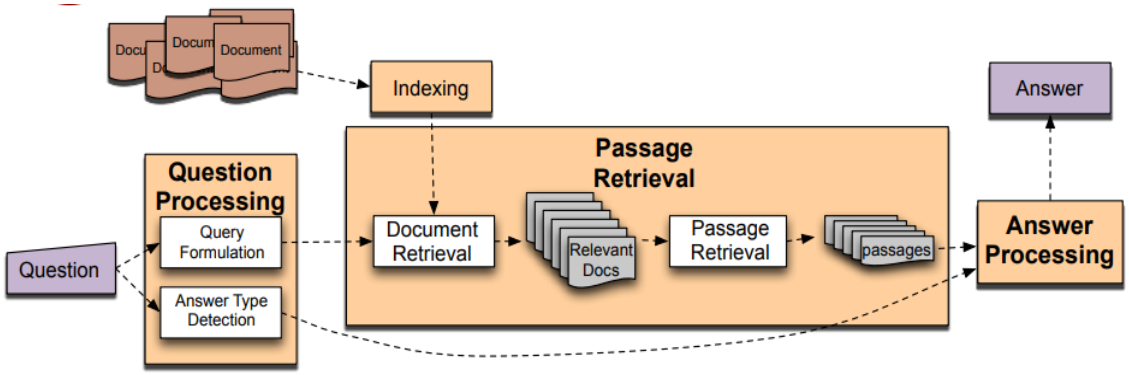


Figure 1.5: IR QA architecture [1]

Knowledge Base (KB)

Instead of relying on unstructured text, this approach involves extracting answers from structured data sources, specifically a knowledge base. Instead of using word-based searches, standard database queries are employed. This paradigm leverages structured data, such as ontologies, which provide a conceptual representation of concepts and their relationships within a specific domain. Ontologies serve as a more advanced form of knowledge base compared to relational databases. To retrieve information from the ontology, queries are executed using structured languages, with one such language being SPARQL (SPARQL Protocol and RDF Query Language), as illustrated in Figure 1.6 [22].

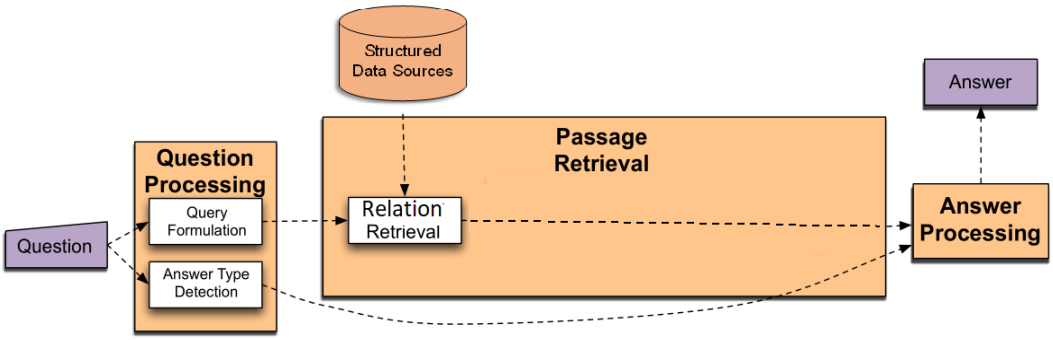


Figure 1.6: KB QA architecture [2]

Natural Language Processing (NLP)

In Question Answering systems based on NLP, a combination of linguistic intuitions and machine learning (ML) algorithms can be employed to extract answers from paragraphs of text that are retrieved [23].

Linguistic intuitions involve knowledge about the structure and meaning of natural languages, such as knowledge of grammar, syntax, and semantics. These intuitions aid in identifying relevant information in the retrieved text and determining the meaning of the question and answer. Examples of linguistic intuitions include part-of-speech tagging, named entity recognition, and dependency parsing, etc [24].

To complement these linguistic intuitions, machine learning methods can be employed to learn patterns and associations from large volumes of training data. By training on a significant corpus of text and associated questions and answers, a machine learning algorithm can learn to identify relevant information in the text using features derived from linguistic intuitions. This approach enables the system to recognize patterns and make accurate predictions, thereby improving the effectiveness of the QA system [25].

The NLP architecture employed for QA solely concentrates on the elements highlighted in red within Figure 1.7.

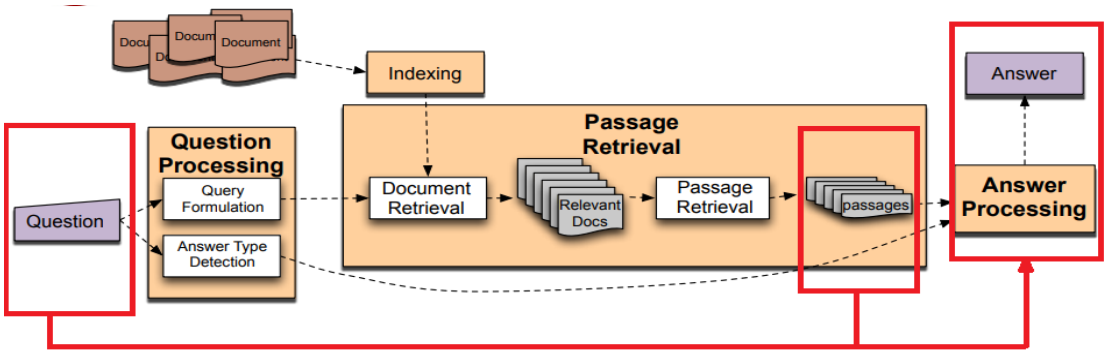


Figure 1.7: NLP QA architecture (only the modules highlighted in red) [2]

Hybrid

A powerful approach for building effective QA systems is to combine various techniques and resources such as IR, NLP QA, and KB QA. IBM Watson serves as an excellent example of this hybrid model, as shown in figure 1.8, which uses machine learning algorithms, NLP techniques, and a vast corpus of knowledge to understand natural language queries and provide accurate answers. By continuously learning from past experiences, Watson improves its performance over time. The success of IBM Watson highlights the importance of integrating multiple techniques and resources to create high-performance QA systems [22].

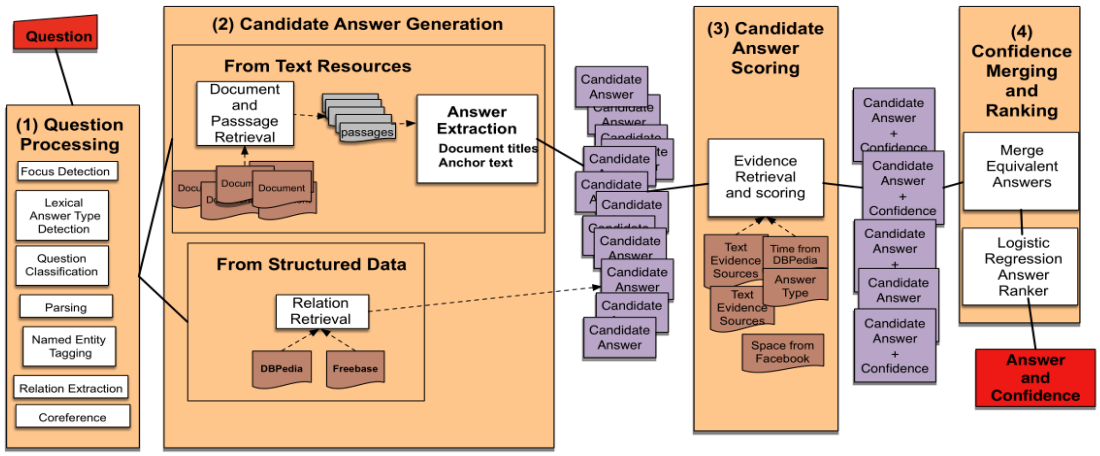


Figure 1.8: IBM Watson QA architecture [1]

1.4.2 Classification based on application domain

In addition to classifying QA systems from an implementation approach, their foundational features can be presented, like domain, which defines QA systems through closed-domain or open-domain models. We distinguish two types of QA systems classified based on their application domain: Open-domain and Closed/Restricted domain.

Open Domain

An open domain QA system is designed to handle a wide range of general questions that come from various domains, and it depends solely on general text and knowledge bases. Additionally, these systems are typically expected to find answers from substantial Open Domain knowledge sources, such as the web or Wikipedia, instead of a specific document [26].

Closed Domain

A closed domain QA system is designed to handle questions related to a particular domain. Typically, such systems are applied in a limited context where a specific type of question is asked, and a small amount of context is given [26]. Examples of closed domain QA systems include customer service chatbots, legal advice systems, and biomedical QA systems. These systems are developed to provide accurate answers within their specific domain, and they have access to a resource of domain-specific knowledge.

QA systems in the Biomedical Domain

The biomedical domain is a specialized form of closed domain QA that is designed to provide answers to questions within the biomedical domain. They are designed to help medical practitioners by offering solutions to queries in the fields of medicine and healthcare such as coronavirus disease. Professionals, researchers, and patients can easily and precisely locate, and extract pertinent information that answers their needs [27].

1.4.3 Classification based on Types of Questions

Generating answers to a user's query is directly related to the type of its questions; therefore, the classification of queries in QA systems directly affects the responses. The

classification is based on all the possible types of questions identified in the literature. These types are as follows: list questions, factoid questions, definition questions, causal questions, confirmation questions, and hypothetical questions [17].

Factoid Questions

Factoid-type questions are simple and fact-based, requiring a short phrase or sentence as an answer [17], such as "How long is the incubation period for COVID-19?" These questions typically start with wh-words like "what," "who," "when," "where," and "how.".

List questions

List questions require providing a list of items in the answer [17]. An example of a list question could be "What are the symptoms of COVID-19?" where the answer would be a list of symptoms such as fever, cough, fatigue, and shortness of breath.

Causal questions [how or why]

Causal questions seek to understand the reasons behind a particular phenomenon by retrieving information that explains the events or factors that led to it. The answer to a causal question provides an explanation of the cause-and-effect relationship being inquired about [17].

Summary Questions

Summary questions can only be answered by a phrase extracted from a relevant document or by writing a short text summarizing the most important relevant information [27]. For example, "What are the most effective measures to prevent the spread of COVID-19?".

Definition questions

Definition questions require intricate processing of retrieved documents, and the absolute answer either consists of a text piece or is acquired after summarizing more documents. Usually, they start with "What is." Answers to definition questions can be any event or entity [28].

Confirmation questions

Confirmation questions are designed to elicit a binary answer, either "yes" or "no," and often require common sense reasoning to produce the answer [28]. For example, "Is COVID-19 caused by a virus?" is a confirmation question that can be answered with a simple "yes" or "no" and requires a basic understanding of the nature of COVID-19.

1.4.4 Classification based on Target tasks

QA comprises multiple different tasks varying in their nature with some being more related to semantics while others are more related to IR.

Document Retrieval

Also known as "Document Search," this is a task in IR systems that involves finding relevant documents from a large collection of text data in response to a user's query after having performed a deeper analysis of these documents.

The principal challenge in document retrieval is to effectively represent the queries and documents in a way that allows for efficient and accurate matching as well as greatly minimizing the amount of time it takes to get the right answer [29].

In a QA system, if a retrieval system cannot locate any pertinent documents for a particular question, the answer selection module will not be able to return a correct answer [29].

Answer Selection

Answer Selection involves the task of discerning the accurate response to a question from a set of candidate answers. The conventional approach typically favors answers that exhibit semantic similarity to the question [30].

In other words, the process of Answer Selection (or Extraction) involves evaluating relevant paragraphs or sentences to determine the most appropriate answer sentence (long) or answer span (short) as the correct response to a given question. [31].

Answer extraction

An answer extraction task is the parser that enables the recognition of the answer candidates in the paragraphs.
So, once an answer candidate has been identified, a set of techniques is applied such as pattern matching, named entity recognition, and machine learning algorithms which depend on the question type, the text format, and the desired level of accuracy, to extract only the relevant word or phrase that answers the question [15].

Machine Reading Comprehension (MRC)

MRC or Reading Comprehension (RC) is a subfield that focuses on developing algorithms and models that can understand and answer questions about the text.
In other words, MRC is the ability to understand, analyze, and interpret a written text. It involves making connections between words, sentences, and paragraphs, as well as using background knowledge and context to understand the meaning of the text. It requires more than just understanding what is clearly stated in the text, but also reading between the lines, the understanding of what has not yet been said, is evident True [13].

Open QA

Open Domain QA, also known as OpenQA, refers to a QA task in which the provided input consists of a question and a concise answer (typically a factoid answer), without any accompanying paragraphs or documents. The objective of the systems is to locate relevant documents from various sources, identify pertinent paragraphs within those documents, and then employ answer processing techniques to extract a succinct answer.
In other words, Open QA is a union of both Answer Sentence Selection and RC tasks done sequentially but with certain changes [31].

1.5 Evaluation Techniques

Several parameters are used to analyze the performance of different QA systems. The evaluation metrics used in QA are accuracy and F1. To understand these measures, we have to use the confusion matrix.

1.5.1 Confusion Matrix

A confusion matrix summarizes the performance of a model with respect to test data. It is a two-dimensional matrix, indexed in one dimension by the true class (actual) and in the other by the class that the model assigns (predicted). The four cells of the matrix are designated as true positive, false positive, true negative, and false negative [32], as indicated in table 1.1.

Various measures, such as accuracy, recall, and precision, are derived and computed from the confusion matrix.

		Predicted	
		Positive	Negative
Actual	Positive	True positive	False negative
	Negative	False positive	True negative

Table 1.1: Confusion Matrix

1.5.2 Accuracy

Accuracy means how many data points are predicted correctly, the number of correct predictions is given as the number of correct predictions divided by the total number of predictions. The formula for it is:

$$accuracy = \frac{truepositive + truenegative}{truepositive + falsepositive + truenegative + falsenegative}$$

The issue observed is the high rate of true negatives that a system can find, e.g., when a factual question is made, there is only one correct answer; anything else would be incorrect and not selected. In this case, a system could have a high calculated accuracy that is unmeaningful. To fix this issue, the F1 measure must be addressed.

1.5.3 F1 Score

The F1 measure is based on the same confusion matrix and has two measures: precision and recall. Precision is the percentage of selected answers that are correct, and recall is

the opposite measure; it is the percentage of correct answers selected. The fact that there is a high rate of true negative answers is no longer relevant when precision and recall are used.

$$Precision = \frac{truepositive}{truepositive + falsepositive}$$

$$Recall = \frac{truepositive}{truepositive + falsenegative}$$

The key concept here is the trade-off researchers must make for each measure while looking for the best metrics to evaluate their systems. Most QA systems should use recall as a metric because it doesn't matter how high the false positive rates are if the true positive rates are high. However, for a list or definition of QA systems, precision would be better. The F1 measure is presented to balance this trade-off [22]. It is calculated as follows:

$$F1 = \frac{2(Precision * Recall)}{Precision + Recall}$$

Some other metrics can be used to evaluate QA systems, such as Mean Reciprocal Rank (MRR) and Exact Match (EM).

1.5.4 Mean Reciprocal Rank (MRR)

The MRR is used to calculate the answer relevance and measure the proportion of questions for which the model provides the correct answer.

$$MRR = \sum_{i=1}^{n} \frac{1}{r_i}$$

where n is the number of test questions and r_i is the rank of the first correct answer for the i-th test question[22].

1.5.5 Exact Match (EM)

EM is a binary metric, which means that it only indicates whether an answer is completely correct or not. In other words, there is no partial credit given for answers that are partially correct.

1.6 Datasets used in QA for Biomedical Domain

The evaluation of a QA model's performance can be conducted by testing it on benchmark datasets commonly employed within the research community. In the following section, we present a compilation of reputable datasets frequently used to report outcomes pertaining to our QA model.

Stanford Question Answering Dataset (SQuAD)

SQuAD is a dataset specifically designed for question answering tasks and reading comprehension. It involves crowd workers posing questions based on a set of Wikipedia articles. SQuAD 1.1 comprises approximately 100,000 question-answer pairs, while SQuAD 2.0 expands on this by including unanswerable questions within the corresponding reading passage. In both versions, the answer to each question is a specific segment of text extracted from the given reading passage [33]. SQuAD itself is not a classifier, but it can be used as a benchmark dataset to train and evaluate the performance of question-answering classifiers.

BIOASQ

The BIOASQ challenge is a notable and accomplished event in the field of biomedical research that has been running successfully since 2013. It encompasses various tasks centered around biomedical data analysis and QA. Task B Phase B specifically concentrates on biomedical question answering, aiming to extract accurate answers from relevant snippets for a given question. This challenge offers the largest and extensively utilized manually-annotated dataset known as Machine Reading Comprehension for Biomedical QA (MRC BQA), serving as a valuable resource in the field [34].

COVID-QA

COVID-QA is a QA dataset that encompasses a collection of 2019 QA pairs. These pairs have been meticulously annotated by volunteer biomedical experts and are specifically focused on scientific articles related to COVID-19.

Its answers are taken from longer texts (6118.5 tokens), and answers are generally longer (13.9 words) and it does not contain n-way annotated development nor test sets

[14].

PubMedQA

PubMedQA is a dataset specifically designed for biomedical question-answering, comprising of questions extracted from PubMed abstracts. The primary objective of this dataset is to provide answers to research inquiries utilizing the relevant information contained within the abstracts, with response options of yes/no/maybe.

As well, this dataset marks the first instance where successful execution of reasoning over biomedical research especially their quantitative contents, is required to answer the asked questions [35].

We provide the comparison table 1.2 below, which highlights the variations between the datasets listed.

	Size	**Evaluation**	**Question Type**	**Domain**
SQuAD	100K	EM, F1	Confirmation, Extraction	Open domain
COVID-QA	2k	EM, F1	Extraction	Closed domain
BioASQ (Task B Phase B)	3.7k	F1, MRR	Extraction, Summary, Confirmation	Closed domain
PubMedQA	212k	Acc, F1	Confiramtion	Closed domain

Table 1.2: Benchmark Datasets

1.7 Summary table

We summarize in the table below 1.3 all the points that encompass our work.

A summary table	
Techniques	Neural NLP, Transformers-Based
Data Format	Text data
Domain	Closed domain
Task	Machine Reading Comprehension
Evaluation	F1 and EM
Question Type	Factoid, List, and Summary questions
Models	BERT and RoBERTa
Datasets	SQUAD and Covid-QA

Table 1.3: Summary table

1.8 Conclusion

In this chapter, we have explored the field of QA and its systems, which are computer systems designed to automatically answer natural language questions asked by users. We began by discussing its history, how it functions, and so on.

Next, we examined some of the key components of a QA system, including the Question Processing module, the document processing module, and the Answer Processing module and their classification.

Finally, we discussed evaluation metrics used to evaluate the performance of QA systems. We also reviewed some of the common datasets used for training QA models, including the SQuAD and the COVID-QA dataset.

In the next chapter, we will explore one of the most powerful techniques for building QA systems: neural networks and their several architectures used in NLP.

Chapter 2

Deep Learning and Transformers Architecture

2.1 Introduction

During the past few years, the field of AI has made significant progress, especially in the domain of DL and neural networks. DL, a subset of machine learning, has enabled machines to learn from large amounts of data and make decisions based on that learning. Neural networks, on the other hand, are a set of algorithms inspired by the structure and function of the human brain. They have become an essential tool for many applications in ML, Computer Vision (CV), NLP, and more.

This chapter focuses on the main architectures of artificial neural networks (ANN), starting with the formal neuron and basic models of ANN. We will discuss neural network learning and the backpropagation algorithm, which is one of the most popular algorithms used in training DL networks. Additionally, we will explore neural networks for NLP, including Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM), Bidirectional Long Short-Term Memory (BLSTM), Transformers, their model architecture, and attention mechanisms.

Furthermore, we will delve into TL, which is a technique that enables the use of pre-trained models to improve the performance of new tasks. We will focus on transfer learning in NLP and transformer-based models in QA systems, including BERT, ELECTRA, GPT, and T5. Finally, we will review related works in this field and provide an overview of the current state-of-the-art techniques. By the end of this chapter, readers will have a strong understanding of the fundamentals of DL, NN, and their applications in NLP and TL.

2.2 Deep Learning

AI is a set of algorithms and methods that try to mimic human intelligence. ML and DL represent one of these widely used techniques [36], as illustrated in Figure 2.1 below.

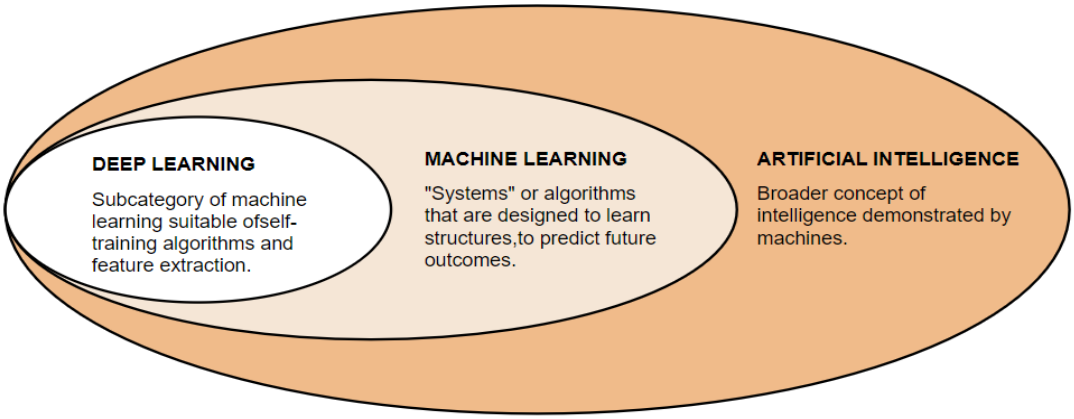


Figure 2.1: AI, ML, and DL [3].

DL focuses on creating large NN models that are capable of making accurate data-driven decisions. It is also particularly suited to contexts where the data is complex and where there are large datasets available [37].

There have been three distinct phases in the development of DL. The first phase, known as **cybernetics**, took place during the 1940s–1960s and involved the use of perceptrons to train a single neuron. The second phase, referred to as **connectionism**, occurred between the 1980s and 1990s and utilized back-propagation to train NN with one or two hidden layers. Finally, the third and current phase, **Deep Learning** as we know it since 2006, which allows us to train very large and deep networks [38].

2.3 Neural Networks (NN)

The term "neural network" was traditionally used to describe a circuit or network of biological neurons.[1] However, in modern usage, "Artificial Neural Network" (ANN) more commonly refers to a mathematical model for processing information, inspired by the information processing mechanisms in biological nervous systems. ANNs are specifically designed to address problems in AI without necessarily relying on the modeling of real

[1]Biological neuron is a cell of the nervous system specialized in the communication and processing of information between the environment and the body, or within the body [39].

biological systems. They have various applications, including speech recognition, image analysis, and adaptive control. To perform these tasks, ANNs undergo a learning process similar to that of biological systems, which involves the tuning of synaptic connections between neurons. This learning process is also a fundamental aspect of ANNs [40].

Main Architectures of ANNs

The basic architecture of an ANN consists of three basic parts (named layers) [41]:

- Input Layer: It is the first layer responsible for receiving information (data) from the external environment.
- Hidden, intermediate, or invisible layers: These layers perform most of the basic work in a network. The layers are made up of neurons responsible for extracting features.
- Output Layer: It is the last layer of neurons that produces and delivers the final network outputs.

We distinguish two main ANN architectures, which are as follows [4]:

Feedforward Architecture: This architecture is called feedforward because information flows always in one direction, which is from the input layer to the output layer (no feedback connections), as shown in figure 2.2.

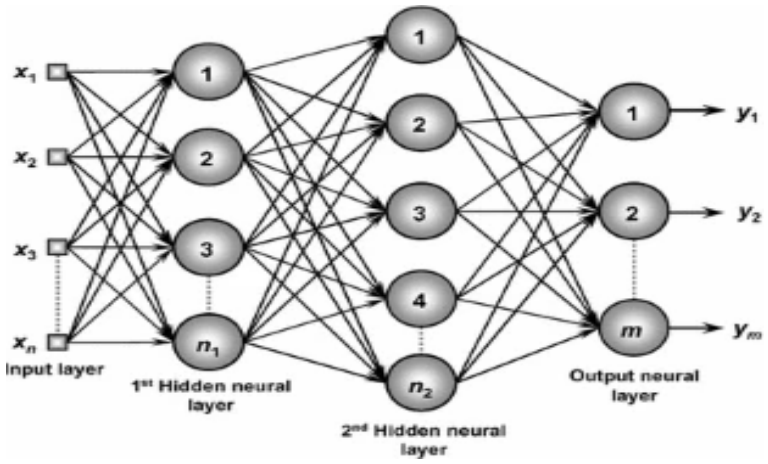


Figure 2.2: Feedforward [4]

Feedback (Recurrent) Architectures: In this architecture, the outputs of neurons are used as feedback inputs for other neurons. This feature enables the network to dynamically process information and maintain relationships, allowing it to store information over time. An example of this architecture is shown in figure 2.3.

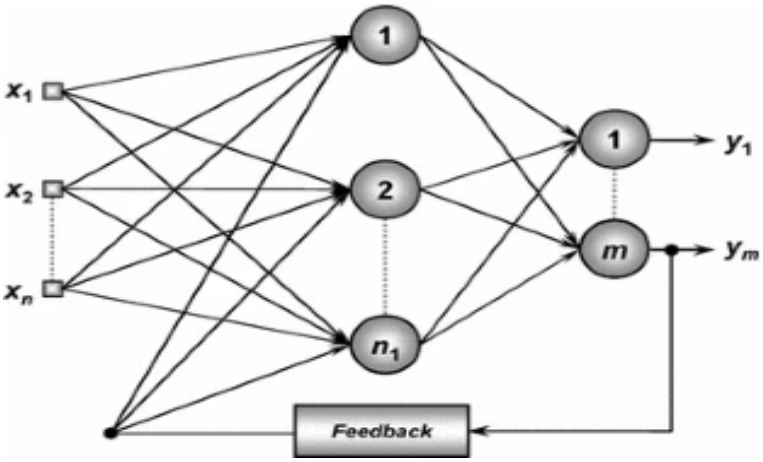


Figure 2.3: Feedback [4]

Formal neuron

Proposed in 1943 by Warren McCulloch and Walter Pitts [42], a formal neuron (artificial neuron) is the building block of the ANN. It is a mathematical function that takes in one or more inputs, applies a weight to each input, sums them up, and passes this sum through an activation function to produce an output [5]. The following figure 2.4 represents the general model of an artificial neuron. Its parameters are as follows:

1. Input connections (inputs) are a vector $(x_1, x_2, \ldots, x_n)$ with weights $(w_1, w_2, \ldots, w_n)$, where each input is multiplied by its weight.

2. A summation function sums weights after multiplying each input by its own associated weight, with the addition of the bias b (used to adjust the output along with the weighted sum of the inputs to the neuron);

$$a = \left(\sum_{i=1}^{n} x_i * w_i + b \right). \tag{2.1}$$

3. An activation function f normalizes the input and produces an output, which is then passed forward into the subsequent layer. It adds non-linearity to the output,

which enables neural networks to solve non-linear problems. In other words, a NN without an activation function is essentially just a linear regression model.

The original activation function, the Heaviside step function, produces an output of either 1 or 0, depending on whether the input is positive or negative, but it is rarely used in modern neural networks due to its non-differentiability. Defined as follows:

$$H(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases}$$

Instead, there are other more well-known and commonly used activation functions, such as the sigmoid function, which normalizes the output of each neuron to a value in the range [0,1] [43]. The sigmoid function can be defined as follows:

$$f(x) = \frac{1}{1 + e^{-z}}$$

4. Output: Output the final activation y.

$$y = f\left(\sum_{i=1}^{n} x_i * w_i + b\right) \tag{2.2}$$

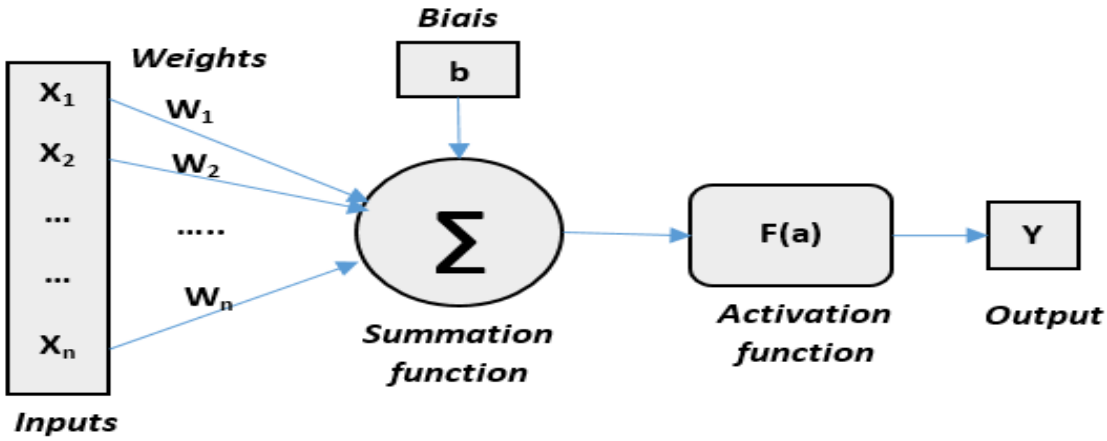


Figure 2.4: Mathematical model of the formal neuron [5]

2.4 Basic models of Artificial Neural

2.4.1 Perceptron

Frank Rosenblatt invented the perceptron in 1957. He proposed to provide neural networks with a supervised learning rule inspired by Hebbian learning. A perceptron is

a supervised learning algorithm for binary classifiers that allows neurons to learn and process elements in the training set one at a time [42].

2.4.2 Multi-Layer Perceptron

A multilayer perceptron is created by chaining several perceptrons. Each neuron in each layer still behaves as a linear classifier, but the use of intermediate layers allows for the creation of complex partitions of the space. This makes it possible to project the data provided as input into new spaces, in which an initially nonlinear task can become linear. However, the use of intermediate layers makes it impossible to train these networks using the perceptron learning rule. This is why it was not until the publication of gradient backpropagation techniques that these networks became more widely used [42].

2.5 Neural Network Learning

One of the most important feature of an artificial neural network is its ability to learn [44]. Neural network learning is divided into two types:

- **Supervised Learning:** This type of learning is based on matching inputs with their correct outputs. The neural network is trained by improving the values of the weights in order to reach the appropriate weights, to be able to produce the output with the required accuracy corresponding to the input, and that makes it able to produce the correct outputs for any new input. This type of learning is based on a training algorithm called the Backpropagation Algorithm.
- **Unsupervised Learning:** This type of learning depends only on the inputs without their correct outputs. The network works to find relationships that link these inputs and try to classify them into similar categories by extracting distinct patterns for each category. This enables the network to give output for the new input, depending on its patterns [45].

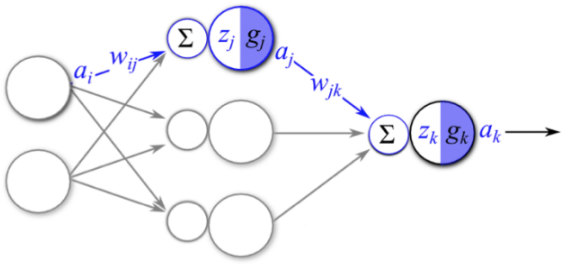
In supervised learning tasks, the objective of training ANNs is to minimize the errors between the desired output signal and the actual output signal; this error is typically defined as a **cost function**. In order to minimize errors, we need to update the weights related to each neuron. One common method for performing this weight update is to use

a **gradient descent algorithm**. However, it only focuses on the updating of a single neuron, whereas in ANN, we must design a rule to train all neurons, and **Backpropagation Algorithm** is the most widely used algorithm to compute the gradient of each neuron [45].

2.5.1 Backpropagation Algorithm

This Algorithm is combined with the gradient descent algorithm to calculate the gradient of the loss function for all weights in the network and use the gradient value to update the weights to minimize the loss function. This is done through these successive steps:

1. Forward-propagate: The first step of the backpropagation algorithm is to propagate the inputs forward through the network layers towards the outputs, As shown in Figure 2.5, the output of the network is denoted by a_k, and to obtain the output of each layer, the summation function z_l and activation function a_l are applied to the inputs of each layer l. The indices i, j, and k are used to denote the input layer, hidden layers, and output layer, respectively. This process is calculated using equation (2.2).



$$a_k = g_k(b_k + \sum_j g_j(b_j + \sum_i a_i w_{ij}) w_{ik})) \quad (2.3)$$

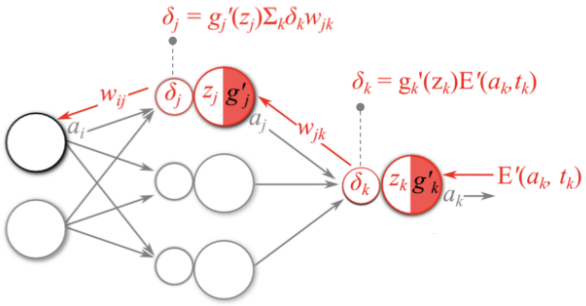
Figure 2.5: Forward-propagate [6].

2. Back-propagate: The second step of the algorithm is to calculate the error E between the network output a_k and the real output t_k. This error is calculated through a cost function by the following equations (2.4)(This function can be as simple as MSE (Mean Squared Error) or more complex like cross-entropy.)

$$E = \frac{1}{2} \sum_{k \in K} (a_k - t_k)^2 \quad (2.4)$$

The error signal δ' is calculated (δ_k for the output layer and δ_j for the hidden layer) by the following equation (2.5) to backpropagate it toward the input as shown in

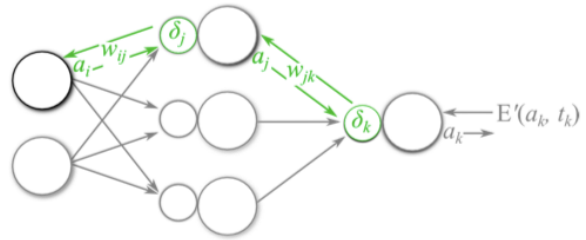
Figure 2.6, where $g'_k(z_k)$ and $g'_j(z_j)$ are the derivatives of the activation functions of the output and hidden layers, respectively. The function $E'(a_k, t_k)$ represents the derivative of the error function.



$$\begin{cases} \delta_k = g'_k(z_k)E'(a_k, t_k) \\ \delta_j = g'_j(z_j)\sum_k \delta_k w_{jk} \end{cases} \tag{2.5}$$

Figure 2.6: Back-propagate [6].

3. Calculate parameter gradient: The third step of the algorithm is to calculate the gradients of the error function for weights in each layer l, to do that we use the forward signals $a_{l\text{-}1}$ from the previous layer and the backward error signals δ_l from the current layer, as shown in Equation (2.6). Similarly, to calculate the gradient for each bias, we use the same gradient rule as for weights, but with one important difference: the "feed-forward activations" for biases are always l+1, as shown in Equation (2.7). This reflects the fact that biases are not connected to the previous layer in the same way that weights are.



$$\frac{\partial E}{\partial w_{l-1,l}} = a_{l-1}\delta_l \tag{2.6}$$

$$\frac{\partial E}{\partial b_l} = b_l\delta_l \tag{2.7}$$

Figure 2.7: Calculate parameter gradient [6]

4. Update parameters: The last step is to update all weights and biases using the gradients calculated in the third step. With the learning rate parameter η by the following equations:

$$w_{l-1,l} = w_{l-1,l} - \eta\frac{\partial E}{\partial w_{l-1,l}}$$

$$b_l = b_l - \eta \frac{\partial E}{\partial b_l}$$

These four steps are repeated until the network error reaches an acceptable low value. At this point, we can say that the artificial neural network has been trained [46].

2.6 Neural Networks for NLP

Deep neural networks have recently contributed significantly to natural language processing tasks, including language modeling, sentiment analysis, syntactic parsing, and machine translation. These models exhibit significant architectural differences, ranging from RNNs and LSTM networks to Transformers. Of course, these distinctions are not clear-cut, as different architectures can be combined within a single model and integrated with other operational blocks, such as neural attention. We summarize these architectures in the following paragraphs [47].

2.6.1 The recurrent neural networks (RNN)

The concept of RNNs was first proposed in the 1980s by a number of researchers, including Paul Werbos, David Rumelhart, and James Elman [48]. RNNs are a type of neural network designed to process sequential data, such as time-series data or natural language sentences. This type uses recurrent connections that allow information to persist over time.

The RNN generates a prediction y_t for the current element of the sequence at time t based on the input x_t, and it takes into account the predictions it has made for the previous elements of the sequence. Practically, this requires introducing a memorization mechanism, in such a way that the prediction at time t uses both the memory at time t-1 and the input sequence element at time t for that RNN define a hidden state $h_{t\text{-}1}$ as the memory of the network at time t-1 and Xt the input of the RNN at time t [7]. The currently hidden state h_t is used to generate the current state's output o_t as shown in figure 2.8, where: x_t is input vector, h_t is hidden layer vector, $o_t$is output vector and tanh: Activation functions.

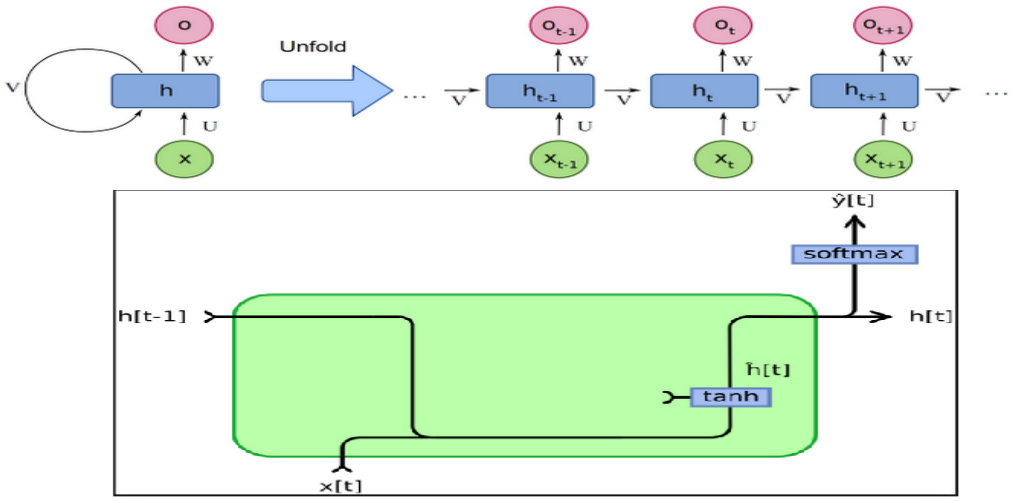


Figure 2.8: The recurrent neural network [7]

The simple RNN has two versions to calculate h and y, with $t = 1$ to t as follows:

- Elman network: Uses the output from the hidden layers as an input with the normal input [49].

$$h_t = g(W_x x_t + U_h h_{t1} + b_h)$$

$$y_t = W_y h_t + b_y$$

- Jordan network: Uses the outputs from the output layers as an input with the normal input[50].

$$h_t = g(W_x x_t + U_h y_{t1} + b_h)$$

$$y_t = W_y h_t + b_y$$

  where at each time step t, ht represents the hidden state, xt is the input, $ht1$ is the hidden state from the previous time step, Wx is the weight matrix for the input, Uh is the weight matrix for the hidden state, bh, and by are the bias term for the hidden state and the output layer respectively,yt represents the output, Wy is the weight matrix for the output layer and g is the activation function.

The architecture of RNNs makes them challenging to train, especially when dealing with lengthy input sequences. Despite the fact that RNNs are excellent at short-term memory, as more data is added, they trend to lose track of previously viewed information. More specifically, there are two major issues that can arise when updating the network weights using the error gradient [51]:

Exploding Gradient: Occurs when gradients accumulate during an update.

Vanishing Gradient: Occurs when the gradients become very small or zero, causing them to vanish during backpropagation. In order to address these problems, a set of solutions has been proposed, the most important of these: Long Short-Term Memory.

2.6.2 Long Short Term Memory (LSTM)

Proposed by Hochreiter and Schmidhuber in 1997, LSTM is a variant of RNN architecture designed to handle the vanishing gradient and exploding Gradient problems [52]. They have replaced the recurrent hidden layer with a more complex cellular structure as shown in Figure 2.9. The cell consists of several components that work together to allow the network to selectively store or discard information over time.

The main components of this cell are as follows:

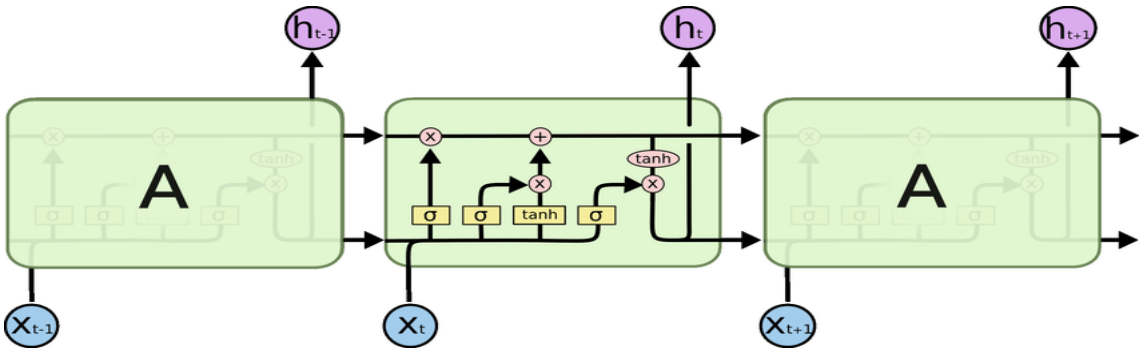


Figure 2.9: LSTM cell [7]

1. Memory cellC_t: This stores the long-term memory and is updated over time based on the input and the gates.

2. Input gate i_t: This determines which information from the input should be stored in the memory cell.

3. Forget gate f_t: This determines which information from the previous hidden state should be forgotten.

4. Output gate o_t: This determines which information from the memory cell should be output to the next layer in the network.

5. Cell state activation functionσ tanh: This is the function that applies to the memory cell's internal state vector to transform it before outputting it to the next layer.

By selectively storing or discarding information over time, this cellular structure can learn to recognize complex patterns in sequential data, making it a powerful tool for tasks such as natural language processing, speech recognition, and image captioning.

2.6.3 Bidirectional Long Short-Term Memory (BLSTM)

The fundamental concept behind BLSTM networks is to use two separate LSTM layers to process each training sequence in both the forward and backward directions. After processing in both directions, the outputs of the two networks are concatenated and fed into the same output layer. This means that for every point in a given sequence, the model can exploit both past and future information [53], as shown in the figure 2.10 .

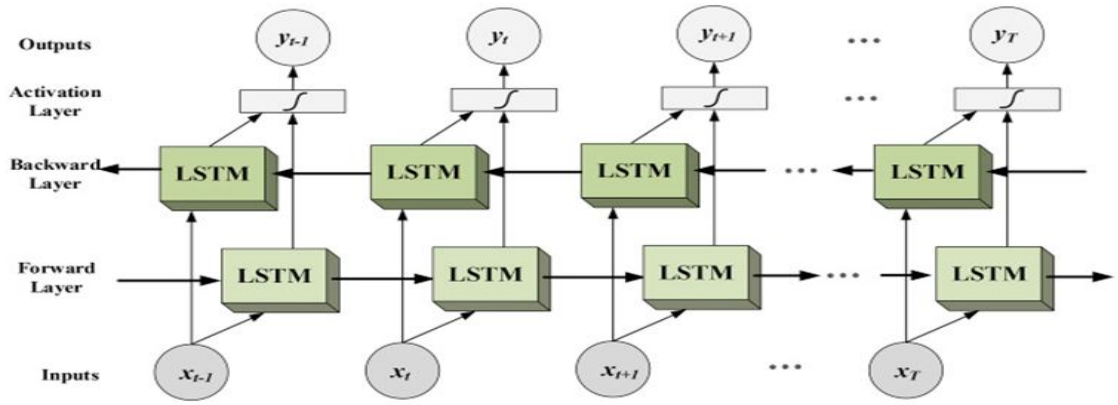


Figure 2.10: BLSTM structure [8]

BLSTM is commonly used in NLP tasks, such as speech recognition, machine translation, sentiment analysis, and text classification.

2.6.4 Sequence-to-sequence models (Seq2Seq)

Sequence-to-sequence models are a straightforward application of the long short-term memory architecture. As shown in figure 2.11, the Seq2Seq model consists of two LSTM models: An encoder and a decoder, where the encoder takes an input sequence and produces a fixed-length vector called the "context vector" or "thought vector", which represents the input sequence's semantic meaning. The decoder then uses this context vector as an initial hidden state and generates an output sequence [54].

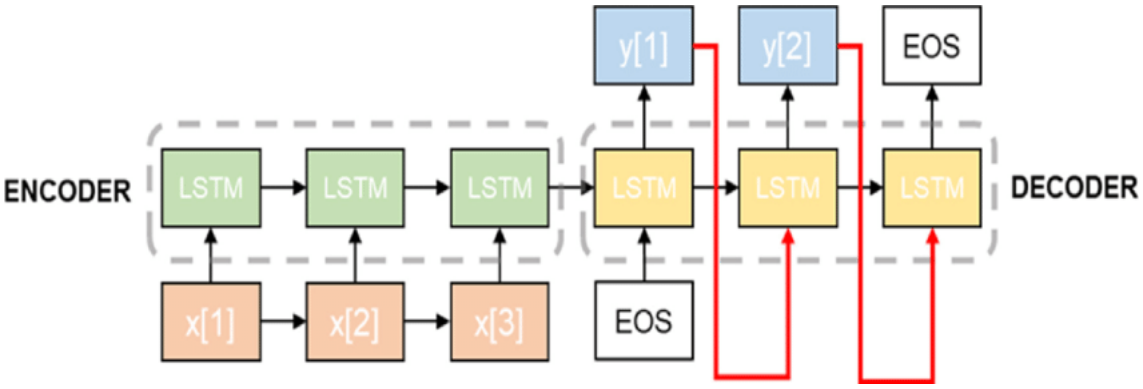


Figure 2.11: Seq2seq LSTM structure [8]

Seq2Seq models have achieved state-of-the-art results on several NLP tasks, including machine translation, text summarization, and speech recognition [7].

2.6.5 Transformers

In 2017, the Google Brain team introduced Transformers in a paper titled "Attention Is All You Need" by Ashish Vaswani et AL. [9]. They proposed a new network architecture based only on an attention mechanism, eliminating the need for recurrence and convolutions. Today, Transformers have become the preferred model for NLP and computer vision. This model architecture consists of a **multi-head self-attention mechanism** combined with an **encoder-decoder structure**. This mechanism can achieve results that outperform various other models in both evaluation score and training time [9].

The Encoder-Decoder Model:

The encoder-decoder models used in Transformers are similar to those used in sequence-to-sequence models, but they are not exactly the same. The main difference is that Transformers use self-attention mechanisms instead of RNNs to process input sequences.

The Attention Mechanism:

The attention mechanism is part of a neural architecture that enables it to dynamically highlight relevant features of the input data, which, in NLP, is typically a sequence of textual elements. It can be applied directly to the raw input or to its higher-level representation. The core idea behind attention is to compute a weight distribution on the input sequence, assigning higher values to more relevant elements [55].

Attention-based neural encoder-decoder models use context vectors to capture source sentence information. These vectors are generated using the previous hidden state of the

decoder and all encoder hidden states. The decoder can attend to all encoder hidden states and use its previous hidden state to generate the target word. The attention mechanism allows the decoder to access relevant information from the context vector without the encoder compressing the source sentence into a single fixed-length vector. The attention mechanism is effective for processing long and any length text sequences [56].

2.6.6 Model Architecture

The structure of most competitive neural sequence models that convert one sequence into another is an encoder-decoder (as in the seq2seq model), in which the encoder maps an input sequence of symbol representations $(x_1, \cdot, x_n)$ to a sequence of continuous representations $(z_1, \cdot, z_n)$. then, the decoder generates an output sequence $(y_1, \cdot, y_m)$, with each symbol generated one at a time. At each step, the model considers the symbols generated in previous steps and uses them as additional input to generate the next symbol. This process is called auto-regression. The Transformer model uses this overall architecture, with stacked self-attention and point-wise fully connected layers for both the encoder and decoder [9], as illustrated in Figure 2.12.

- **Encoder:** The encoder shown in the left half of Figure 2.12 is composed of N = 6 identical layers, where each layer contains two sub-layers: multi-head self-attention and a fully connected feed-forward network. To enable residual connections, all sub-layers, including the embedding layers, produce outputs of dimension $d_model = 512$. The output of each sub-layer is computed as $LayerNorm(x + Sublayer(x))$, where x is the input to the sub-layer, and $Sublayer(x)$ represents the function implemented by the sub-layer itself [9].

- **Decoder:** The decoder shown in the right half of Figure 2.12 is also composed of a stack of $N = 6$ identical layers, each layer contains three sub-layers. The first sub-layer is a masked multi-head self-attention layer that allows the model to attend to different parts of the input sequence. The second sub-layer is a multi-head attention layer that performs attention over the output of the encoder stack. The third sub-layer is a fully connected feed-forward network that applies non-linear transformations to the outputs of the second sub-layer.

  All sub-layers in the decoder, as well as the embedding layers, produce outputs of

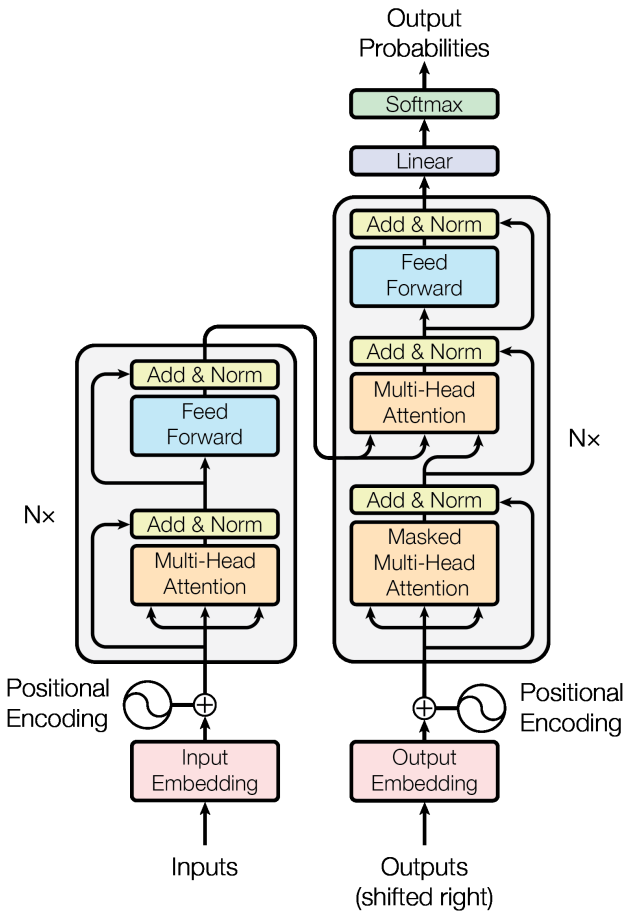


Figure 2.12: Transformer model architecture [9]

dimension $d_model = 512$. Residual connections and layer normalization are applied around each of the three sub-layers in each decoder layer [9].

- **Attention**

  An attention function uses vectors to map a query and a set of key-value pairs to an output. The output is a weighted sum of the values, with each weight determined by a compatibility function between the query and its corresponding key [9].

  Figure 2.13 shows two types of attention mechanisms used in the Transformer: the left panel depicts Multi-Head Attention and the right panel shows Scaled Dot-Product Attention.

  Scaled Dot-Product Attention: is an attention mechanism that operates on queries Q, keys K, and values V, where the dot products query with all keys are scaled down by $\sqrt{d_k}$, and apply a softmax function to obtain the weights on the values. practically, we have a query Q, a key k, and a value V, and we calculate the

attention as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

Scaling the dot products by $\frac{1}{\sqrt{d_k}}$ ensures that the dot products of the query and key vectors are not too large, which helps prevent the $softmax$ function from saturating.

Additionally, by scaling down the dot products, the resulting values will have a mean of 0 and variance of 1, which makes it easier for the model to learn and update the weights during training [9].

Multi-Head Attention: allows the model to capture various types of information and learn complex relationships between different parts of the input, by attending to different subspaces of the input sequence, and is defined as:

$$\text{MultiHead}\left(\mathbf{Q}, \mathbf{K}, \mathbf{V}\right) = \left[\text{head}_1, \ldots, \text{head}_h\right] \mathbf{W}_0$$

$$\text{where head}_i = \text{Attention}\left(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V\right)$$

where $\mathbf{W}_i^Q, \mathbf{W}_i^K, \mathbf{W}_i^V$ and $\mathbf{W}_0$ are parameter projection matrices

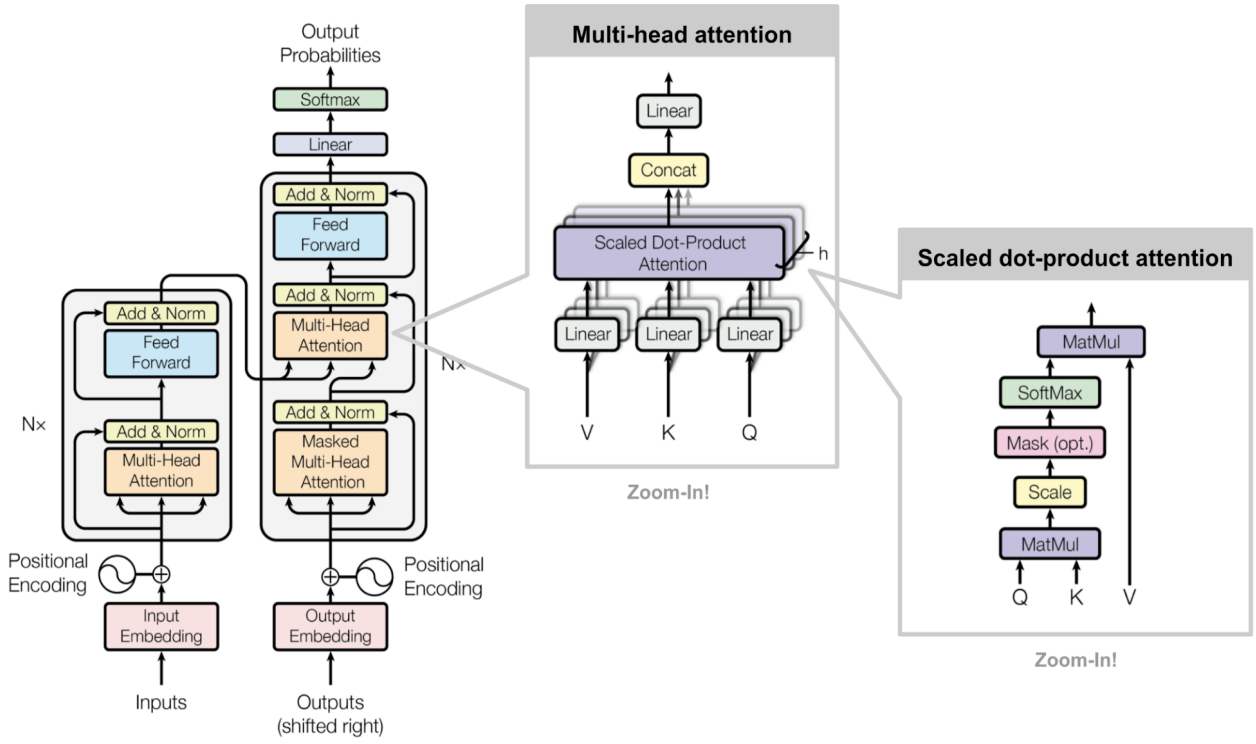


Figure 2.13: (left)Multi-Head Attention consists of several attention layers running in parallel (right)Scaled Dot-Product Attention [10]

2.6.7 Transfer Learning

Transfer learning (TL) was defined by Pan and Yang in 2010 [57] as a machine learning technique that aims to improve the learning of a target predictive function $f_T()$ in a target domain D_T and learning task T_T using knowledge gained from a related but different source domain D_S and learning task T_S, where $D_S \neq D_T$ or $T_S \neq T_T$ as shown in figure 2.14.

The objective of transfer learning is to enable learning in D_T using the information gained from D_S and T_S, where $D_S \neq D_T$ or $T_S \neq T_T$. This involves transferring knowledge from the source domain to the target domain, which may involve adapting a pre-trained model, transferring features, or reusing parameters from the source domain to improve the learning of the target function [57].

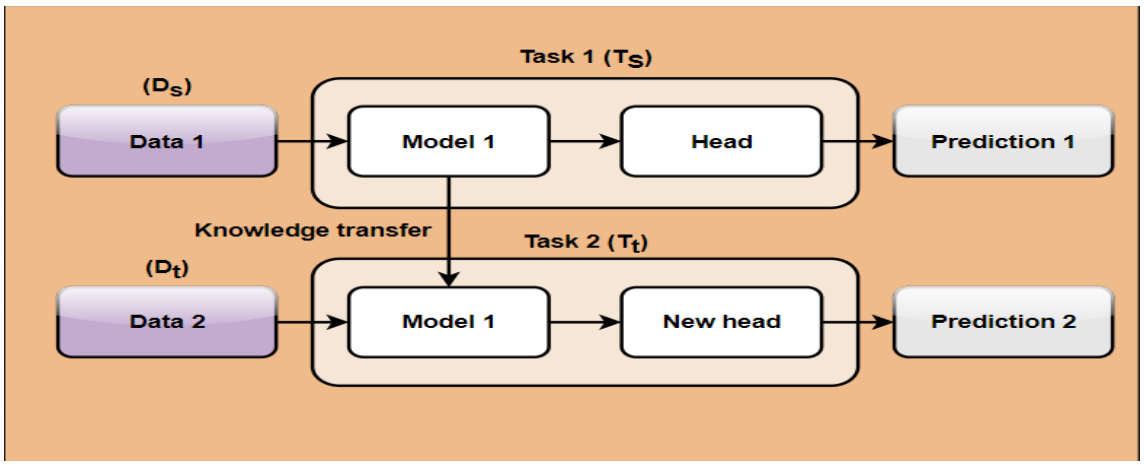


Figure 2.14: Transfer Learning

2.7 Transfer Learning in NLP

Transfer learning has proven to be highly effective in improving the performance of low-resource NLP tasks that have limited training data, such as question answering. The two most widely used transfer learning methods in NLP are feature-based transfer and fine-tuning [58].

2.7.1 feature-based transfer

Feature-based transfer learning for NLP involves using a pre-trained language model to extract features from text data and then using these features as input to a downstream

NLP task. The pre-trained language model can be a general-purpose model that has been trained on a large amount of text data, such as BERT, GPT-2, or RoBERTa, and the downstream NLP task can be a specific task such as sentiment analysis, text classification, or QA [58].

2.7.2 Fine-tuning

Fine-tuning a pre-trained model for downstream tasks in NLP has become a popular approach, particularly in recent years with the emergence of large pre-trained models such as BERT and GPT-2. Fine-tuning involves training the pre-trained model on a task-specific dataset, such as QA or text classification, by copying the weights from a pre-trained network and tuning them on the downstream task. The upstream model is usually a neural language model that is pre-trained on a large dataset. One advantage of fine-tuning is that it does not require task-specific model design, unlike feature-based TL, where a separate classifier is trained on top of the pre-trained model. Instead, fine-tuning replaces the final layer(s) of the pre-trained model with a new layer(s) that is specific to the downstream task and then trains the entire model on the task-specific labeled data [58].

2.8 Transformer-based models in QA systems

Transformer-based models have been widely used in QA systems due to their ability to capture the contextual information of a sentence. The most popular transformer-based models used in QA systems are BERT, ELECTRA, GPT, and T5.

2.8.1 BERT

Bidirectional Encoder Representations from Transformers (BERT): is an Encoder-based only model that is based on the Transformer architecture, introduced by Google in 2018. The BERT architectures originally presented by Devlin et al [59] include $BERT_BASE$ and $BERT_LARGE$, the "basic" version has 12 encoders, while the "large" version has 24 encoders. BERT's uniqueness lies in its ability to read words from both directions simultaneously, enabling it to utilize various strategies to learn contextualized word representations.

BERT comprises two phases: Pre-Training and Fine-Tuning.

Pre-Training: During the pre-training phase, BERT is trained on large amounts of unlabeled text to learn contextual representations of words. This is done by utilizing two tasks: Masked Language Model (MLM) and Next Sentence Prediction (NSP). In MLM, a certain percentage of words in a sentence are masked, and the model is trained to predict the masked words based on the surrounding context. In NSP, the model is trained to predict whether two sentences are consecutive in the original text or not. By pre-training on these tasks, BERT can learn to understand the relationships between words in a sentence and how sentences relate to each other.

Fine-Tuning: BERT can be fine-tuned for several downstream NLP tasks such as classification, Question Answering, and text summarization. Since BERT has language comprehension abilities, fine-tuning the pre-trained weights becomes easier. In this process, the pre-trained weights are adjusted to fit the specific task at hand, using labeled data specific to that task. By fine-tuning these tasks, BERT can be customized to perform specific NLP tasks with high accuracy [59].

2.8.2 ELECTRA

Efficiently Learning an Encoder that Classifies Token Replacements Accurately (ELECTRA) is a variation of BERT that pre-trains the model using a different task called replaced token detection, rather than the MLM task used in BERT. In this task, instead of masking tokens with [MASK], tokens are replaced with different tokens, and the model is trained to determine whether the tokens are actual or replaced. This approach allows ELECTRA to use a larger proportion of the input text for pre-training and has shown to be more computationally efficient than BERT while achieving similar or better performance on downstream tasks [7]. ELECTRA is Encoder-based only, which means that it only uses the encoder part of the transformer architecture for pre-training and inference. This makes the model computationally efficient and allows it to handle longer input sequences compared to Decoder-based models like GPT. ELECTRA has been used in several NLP applications, Including Sentiment Analysis (ISA), QA, and Language Modeling (LM), among others [60].

2.8.3 GPT

Generative Pre-trained Transformer (GPT) models have taken the NLP world by storm. GPT is a Decoder-based model that uses a transformer-based architecture for language modeling tasks, such as text generation, machine translation, and summarization. Unlike other pre-training models, GPT models are trained only with a left-to-right language modeling objective and do not require any supervised training data. They are fine-tuned on downstream NLP tasks with only a small amount of task-specific labeled data. To achieve state-of-the-art performance on these tasks, these language models require very little to no examples.

Radford et al. introduced GPT for downstream tasks through fine-tuning. In 2019, they developed GPT-2, and in 2020, Brown et al. introduced the GPT-3 zero-shot transformer technique, which eliminates the need for fine-tuning. The number of parameters in the original Transformer model increased from 65M to 175B in the GPT-3 model between 2017 and 2020 [7].

2.8.4 T5

Text-to-Text Transfer Transformer (T5) is a type of neural network architecture for NLP tasks, introduced by Raffel Colin and al. in the research paper "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer" [61]. T5 is an Encoder-Decoder-based model that is based on the Transformer architecture and is designed to perform a wide range of NLP tasks, including question-answering, summarization, and translation, among others. The T5 model is trained on a large corpus of text and fine-tuned on specific tasks, and it is capable of performing well even on tasks that it has never been explicitly trained on (zero-shot learning). Additionally, the T5 model can perform multiple tasks at once through a technique called "prefix tuning [61].

The T5 model is a Pre-Norm model that uses LayerNorm without bias, and it uses ReLU activation instead of the more common GELU activation function. The output projection weights are tied to the input embedding matrix, and it uses 128 relative positional embeddings that are added at each layer. The T5 model comes in several sizes, with the smallest having 233 million parameters and the largest having 2.8 billion parameters. T5v1.1 is an updated version of T5 that is trained on different data, uses GEGLU activations, and does not include dropout. It also slightly modifies the model shapes and does not tie the

logit layer with input embeddings [62].

2.9 Related Works

This section provides an overview of related works that had already been done in QA for biomedical domain and COVID-19 using DL techniques and Transformers architectures.

In [14], Möller et al. presented the first dataset for COVID-related QA in SQuAD-style, named COVID-QA. The authors used the CORD-19 scientific articles and provided 2019 annotated QA pairs related to COVID-19, based on 147 scientific articles. To evaluate the proposed dataset, they chose the pre-trained RoBERTa-base architecture was then fine-tuned on the SQuAD dataset, as well as the proposed COVID-QA dataset. This study shows that finetuning the model on the COVID-QA dataset results in significant improvement across both F1 and EM scores.

Tang et al. [63] developed CovidQA, a dataset consisting of 124 question-article pairs related to COVID-19, which were extracted from the CORD-19 dataset [64]. Although the dataset is small, it can still be useful for evaluating the performance of models in the COVID-19 domain, particularly those using zero-shot or TL methods. The authors tested the effectiveness of several baseline techniques, including term-based methods and various transformer-based models. In unsupervised models, BioBERT was found to be the most effective compared to BERT and T5. For fine-tuning models in MS MARCO and SQuAD, T5 achieved the highest overall effectiveness. The authors also described their methodology for constructing the dataset and presented their findings on the effectiveness of these baselines. However, due to its small size, the dataset may not be suitable for training DL Network models.

Lee et al.[65] proposed a system consisting of three components: COVIDASK, a QA system for COVID-19 that combines biomedical text mining and QA techniques to provide answers to questions in real-time. the system uses DenSPI (Document Encoder with Selective Paragraph Interactions) model which is purely trained on SQuaD and Natural Questions; and the evaluation of COVIDASK on a COVID-19 dataset created by the authors. Although COVIDASK is not an open-domain question-answering system, it employs techniques from such systems as it needs to handle a large amount of text. The primary focus of COVIDASK is reducing the latency between querying and receiving an

answer. This is achieved by either decreasing the number of documents retrieved or pre-indexing all appropriate answer phrases into dense and sparse vectors. Then, Maximum Inner Product Search is performed between query vectors and phrase vectors to reduce the number of times a model passes through a document to only one.

Su et al. [66] proposed the CAiRE-COVID QA system, which consists of three different modules. The first module, Document Retriever, preprocesses the user's query and retrieves the n most relevant publications. It paraphrases long, complex queries into simpler ones that are easier for the system to comprehend. These queries are run through the IR module, which returns paragraphs with the highest matching scores. The second module, Relevant Snippet Selector, highlights and re-ranks the most relevant parts of the retrieved paragraphs based on their scores,to enhance both generalization and domain-expertise capability, they combine the HLTC-MRQA model that is an XLNet-based QA model which is trained on six different QA datasets via multi-task learning with the BioBERT QA model, which is fine-tuned on the SQuAD dataset and evaluate their QA module performance on the CovidQA dataset The third module, Multi-Document Summarizer, returns an abstractive summary by summarizing the top relevant paragraphs. This QA module is an example of an extractor and re-ranker, as it focuses on a small-sized dataset such as sentences or articles.

Alzubi et al. [67] proposed and developed COBERT, a QA system that utilizes BERT to answer COVID-19 related queries. The authors pre-trained BERT on a COVID-19 specific dataset to enhance the model's ability to understand domain-specific language. The system was evaluated on a dataset consisting of questions related to COVID-19. The results show that COBERT achieves 81.3 EM and 88.7 F1-score on SQuAD 1.1 dev, and after fitting the pipeline on the CORD-19 corpus, the model achieves 81.5 EM and 88.3 F1-score. The authors suggest that COBERT could assist healthcare professionals in finding accurate information quickly during the COVID-19 pandemic.

In the paper proposed by Reddy et al. [68], the authors present an End-to-end QA system that requires both IR over a large document collection and MRC on the retrieved passages. The baseline MRC system fine-tunes a pre-trained RoBERTa-large model for 3 epochs on SQuAD2.0 and then for 1 epoch on Natural Questions (NQ) training examples. It achieves an EM 34.0 and F1 59.4 on the NQ dev set. The proposed method includes a novel example generation model that can produce synthetic training examples for both

IR and MRC. These generated examples are used to fine-tune a pre-trained language model on the target domain, which allows the model to adapt effectively to the COVID-19 domain without the need for large amounts of annotated data. Recent work has shown that neural IR systems can be trained successfully using only supervised QA examples from open-domain datasets.

In [69], Ngai et al. created three QA systems for the Kaggle COVID-19 Open Research Dataset (CORD-19) after reviewing various QA systems submitted to the Kaggle competition. They evaluated the performance of three transformer models (BERT, ALBERT, and T5) on two new QA datasets (CovidQA and CovidGQA) and presented preliminary results. The authors also discussed challenges in developing a high-performing QA system for COVID-19 research and suggested solutions to improve performance.

2.9.1 Comparative table

The following table 2.1 summarizes the related works discussed previously and compares them to our proposed work based on different comparaison criteria.

Work	**Pretrained Model**	**Further pretrain Dataset**	**Finetuning Dataset**	**Further finetuning Dataset**	**Evaluation Dataset**	**Evaluation Metrics**	**Score**
Moller et al. [14]	RoBERTa		SQUAD	COVID-QA	COVID-QA	F1	**59.53**
						EM	**25.90**
Tang et al. [63]	T5		MS Marco		COVIDQA	Precision	0.282
						Recall	0.404
						MRR	0.415
Ngai et al. [69]	ALBERT-base		SQUAD		COVIDQA	F1	23.37
						EM	13.04
	BERT large		SQUAD		COVIDQA	F1	26.32
						EM	11.59
Reddy et aL.[68]	RoBERTa large		SQUAD	CORD-19	COVID-QA	F1	**59.4**
						EM	**34.0**

Lee et al. [65]	DenSPI		SQUAD	Natural Questions	COVID-19	EM_{sent}	0.7736
Dan Su et al. [66]	BioBert	CORD-19	SQUAD		COVID19	MRR	0.288
						Precision	0.177
						Recall	0.423
	HLTC-MRQA (XLNet)	CORD-19			COVID19	MRR	0.291
						Precision	0.169
						Recall	0.415
	Ensemble	CORD-19			COVID19	MRR	0.318
						Precision	0.192
						Recall	0.477
Alzubi et al. [67]	BERT		SQUAD	CORD-19	CORD-19	EM	81.5
						F1	88.3
	Distil-BERT		SQUAD	CORD-19	CORD-19	EM	80.6
						F1	87.3
Proposed model	BERT		SQUAD	COVID-QA	COVID-QA	EM	0.30
						F1	0.58
	RoBERTa		SQUAD	COVID-QA	COVID-QA	F1	**64.87**
						EM	**38.61**

Table 2.1: Comparative table of related works

2.10 Conclusion

This chapter has provided an overview of deep learning and neural networks, focusing on the main architectures of ANNs. We discussed the formal neuron and basic models of ANNs, as well as neural network learning with the backpropagation algorithm.

Furthermore, we explored the use of neural networks in NLP, including RNNs, LSTMs, BLSTMs, Transformers, and their model architecture and attention mechanisms. We also discussed TL, a technique that allows the use of pre-trained models to improve the performance of new tasks, with a focus on TL in NLP and transformer-based models in QA systems, such as BERT, ELECTRA, GPT, and T5.

Finally, we reviewed related works in this field, providing an overview of the current state-of-the-art techniques. this chapter has demonstrated the significant impact of DL and transformer architectures on various domains, especially in NLP and QA.

Chapter 3

Proposed QA Model

3.1 Introduction

In this chapter, we will discuss the proposed approach for QA using transformers for the biomedical domain. The goal of this solution is to enable models to understand and answer medical-related questions asked in natural languages.

The system architecture of the proposed QA model encompasses various components and processes. It involves the utilization of the SQUAD and COVID-QA datasets, which serve as the foundation for training and evaluating the model. These datasets provide a diverse range of questions and contexts, allowing for a comprehensive assessment of the QA model's performance.

The chapter explores the inclusion of language models, specifically BERT and RoBERTa, in the development of the proposed QA model. These pre-trained models have demonstrated remarkable performance in NLP tasks and form an important part of the architecture. The chapter also encompasses data exploration, which involves gaining insights into the datasets and understanding their characteristics, distribution, and potential challenges.

The preprocessing stage is essential in preparing the data for training. This includes tokenization, which breaks down the text into smaller units for analysis, determining the end index of answers to accurately locate the relevant information, and converting answer positions to token-based positions for model compatibility.

The fine-tuning process involves setting up the model architecture and training it on the available data. This step allows the model to adapt and learn from the specific domain

and task requirements. Training the model involves optimizing the model parameters and adjusting the hyper-parameters to achieve the best possible performance.

3.2 Overall Architecture

QA systems can be useful for the medical and biomedical research communities to stay up-to-date when new literature is rapidly growing. However, developing accurate QA systems using small datasets can be challenging due to the limited diversity and complexity of the data. This challenge is particularly pronounced in specialized tasks and topics such as COVID-19, where relevant datasets are often scarce. This can make it difficult to train a QA system that can accurately answer questions about this topic.

One solution to address this challenge is to perform two rounds of fine-tuning on pre-trained transformer-based models: initial fine-tuning followed by further fine-tuning. First, we fine-tune the model on the SQuAD dataset to enable the model to learn general QA patterns. This enhances the model's accuracy in answering a broader range of questions. Next, we further fine-tune the model on the COVID-QA dataset to enhance its performance on the specific domain of COVID-19. This enables the model to learn more domain-specific features and patterns relevant to answering COVID-related questions.

The subsequent sections will provide a detailed discussion of the pre-processing techniques employed and the hyper-parameters used during the fine-tuning process. Figure 3.1 presents an overview of the pipeline architecture for our proposed approach.

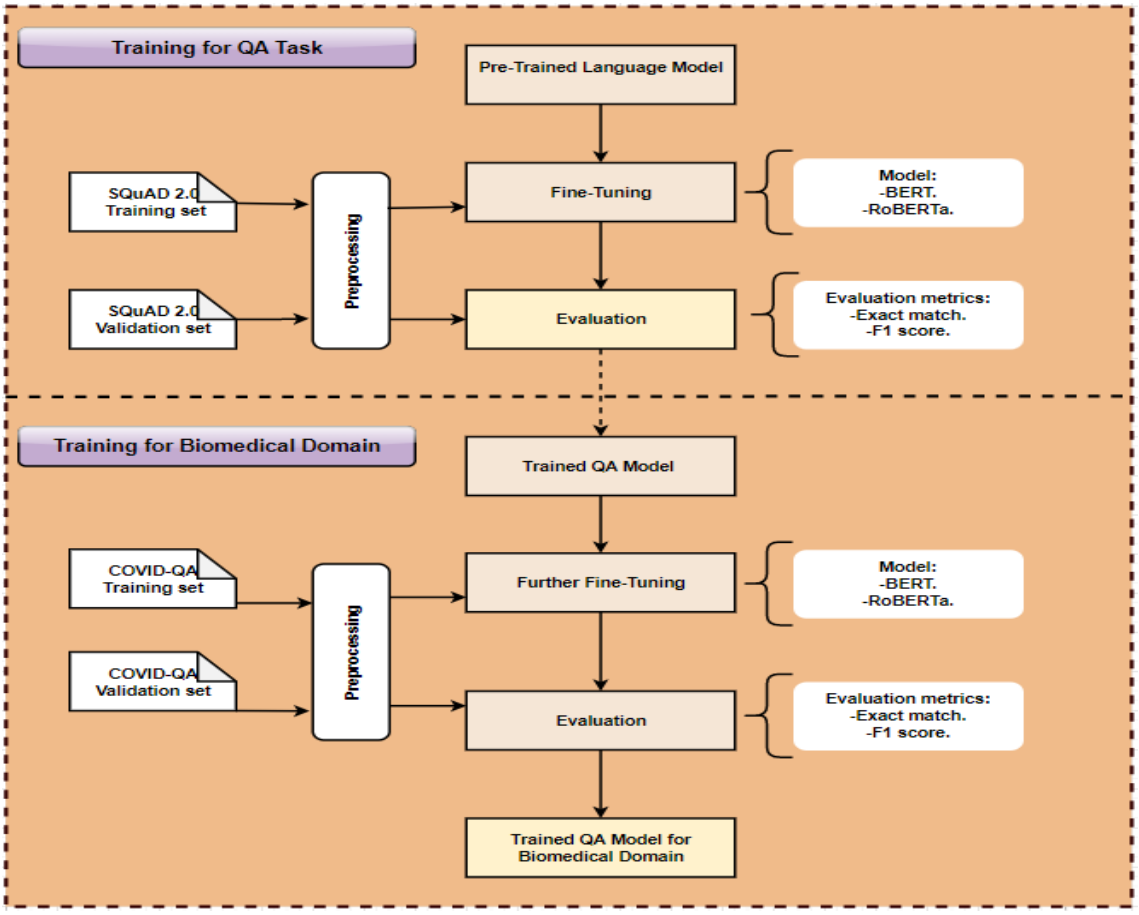


Figure 3.1: General pipeline of the training steps of the proposed models

3.3 System Architecture

Answering questions using transformer-based models such as BERT employs a series of essential steps to provide accurate responses to user queries. Figure 3.2 illustrates the overall process of QA.

Preprocessing: Questions and contexts, which are extracted from the dataset, are tokenized and converted into a sequence of tokens. Special tokens like [CLS] (classification) and [SEP] (separator) are added to mark the beginning and the separation of the questions and contexts.

Embedding: The tokenized questions and contexts are embedded into a vector space. The embeddings are learned during the pre-training process. These embeddings capture the contextual meaning of the tokens.

Encoding: The embedded questions and contexts are encoded into a sequence of

vectors using a transformer encoder. The transformer encoder consists of multiple layers that process the tokens and capture their contextual representations.

Attention: The encoded questions and contexts are attended to each other using a self-attention mechanism called the Transformer's multi-head attention. The self-attention mechanism calculates attention weights for each token based on its relationship with other tokens, capturing the importance of each token in the context.

Answer prediction: The attended questions and contexts are passed through a task-specific layer, often a combination of linear and softmax[1] layers, to predict the start and end positions of the answer within the context. These layers map the encoded representations to probability distributions over all possible answer spans.

Post-processing: The start and end positions predicted by the model are used to extract the answer span from the context. The answer span is then cleaned, processed, and returned as the final answer.

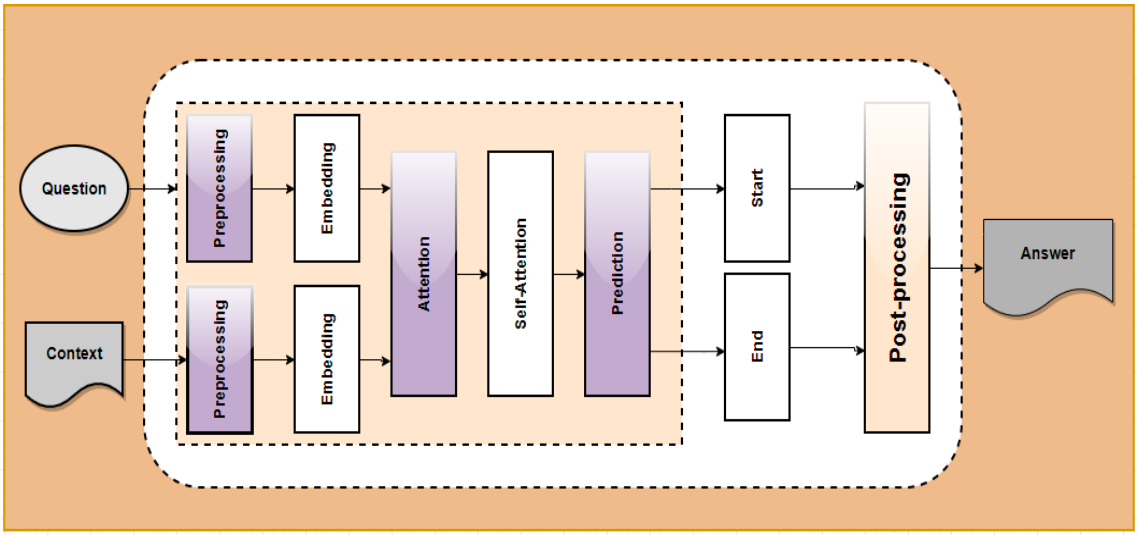


Figure 3.2: General system architecture and steps of transformer-based models

3.4 Used Datasets

To train our models, we utilize two datasets: SQuAD for fine-tuning and COVID-QA for further fine-tuning.

[1]is a mathematical function that is used in machine learning and statistics to convert a vector into a probability distribution.

3.4.1 Stanford Question Answering Dataset (SQuAD)

SQuAD is a collection of QA pairs derived from Wikipedia articles, where the correct answers can be any sequence of tokens in the given text. Created through crowdsourcing, SQuAD offers diversity in its questions and answers, making it a valuable resource for training models. SQuAD 1.1 comprises 107,785 QA pairs from 536 articles, while SQuAD 2.0 includes additional unanswerable questions [33]. By fine-tuning BERT on the SQuAD V2 dataset, the model becomes proficient in identifying crucial contextual cues in the input text and generating accurate answers for a wide range of questions.

3.4.2 COVID-QA

COVID-QA is a QA dataset consisting of 2,019 QA pairs that have been annotated by volunteer biomedical experts [14]. The questions and answers are based on scientific articles specifically related to COVID-19. By incorporating this dataset into the fine-tuning process, our model gains a deeper understanding of domain-specific features and patterns that are crucial for accurately answering COVID-related questions. This additional fine-tuning step enhances the model's performance in addressing queries related to the biomedical and COVID-19 domains.

3.4.3 Data Exploration

Analyzing and exploiting the datasets is helpful for us in the data prepossessing step and in choosing good hyper-parameter values for our models. These have proved to be extremely influential in determining the performance of our model in the training process.

Format

Both the SQuAD and COVID-QA datasets follow a similar format as shown in Fig 3.3, which consists of the following fields:

'Id': Unique identifier for each data instance.

'Title': Title of the article or document containing the context.

'Context': The passage or section from the document that contains the information relevant to the question.

'Question': The question asked based on the given context. Questions in both datasets are mostly factoid questions.

'Answers': A list of answer objects, where each object contains the 'text' field representing the answer text and the 'answer_start' field indicating the character offset in the context where the answer starts.

```
DatasetDict({
    train: Dataset({
        features: ['id', 'title', 'context', 'question', 'answers'],
        num_rows: 130319
    })
    validation: Dataset({
        features: ['id', 'title', 'context', 'question', 'answers'],
        num_rows: 11873
    })
})
```

Figure 3.3: Dataset format

The following table 3.1 illustrates a concrete example taken from the SQuAD v2.0 dataset for illustration purposes:

Id	56de3aeccffd8e1900b4b6af
Title	Southern Europe
Context	" The Late Middle Ages represented a period of upheaval in Europe. The epidemic known as the Black Death and an associated famine caused a demographic catastrophe in Europe as the population plummeted. Dynastic struggles and wars of conquest kept many of the states of Europe at war for much of the period. In the Balkans, the Ottoman Empire, a Turkish state originating in Anatolia, encroached steadily on former Byzantine lands, culminating in the Fall of Constantinople in 1453 ."
Question	What disease plagued Europe during the Late Middle Ages?
Answers	'text': ['the Black Death'], 'answer_start': [87]

Table 3.1: Input Example from SQuAD v2.0

And the following table 3.2 illustrates an example taken from the COVID-QA dataset.

Id	262
document id	630
Context	"... Mother-to-child transmission (MTCT) is the main cause of HIV-1 infection in children worldwide. Given that the C-type lectin receptor, dendritic cell-specific ICAM-grabbing non-integrin-related (DC-SIGNR, also known as CD209L or liver/lymph node–specific ICAM-grabbing non-integrin (L-SIGN)), can interact with pathogens including HIV-1 and is expressed at the maternal-fetal interface, we hypothesized that it could influence MTCT of HIV-1..."
Question	What is the main cause of HIV-1 infection in children?
Answers	'text': ['Mother-to-child transmission (MTCT)'], 'answer_start': [370]

Table 3.2: Input Example from COVID-QA

Characteristics

Positive questions and negative questions in the context of QA refer to answerable questions and unanswerable questions, respectively. In the SQuAD dataset, as depicted in Figure 3.4, there are 86,821 positive questions, which are questions with corresponding answers, and 43,498 negative questions, which are questions without any feasible answer. In contrast, all the questions in the COVID-QA dataset are answerable, meaning that each question has a relevant answer associated with it. Incorporating unanswerable questions into the training process has several benefits. By exposing QA models to both answerable and unanswerable questions, the models can become more robust and better aligned with real-world QA scenarios.

Unanswerable questions imply that the answer is an empty list. We must take this into consideration when calculating the start and end of the answer.

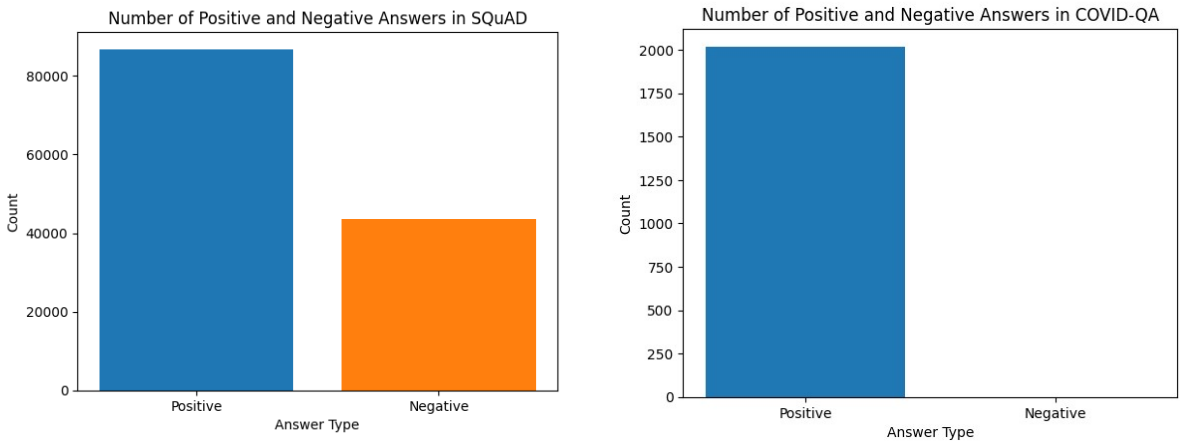


Figure 3.4: Positive and negative questions

Context Length

The structure of the context in the SQuAD and COVID-QA datasets differs significantly as figure 3.5 shows. In SQuAD, the contexts are subsections or paragraphs extracted from Wikipedia articles, while in COVID-QA, the contexts consist of the full research articles. As a result, the contexts in COVID-QA are significantly longer than the contexts in SQuAD.

This disparity in context length has implications for model training. Therefore, when training models on these datasets, it is important to consider the context length when choosing the hyper-parameters.

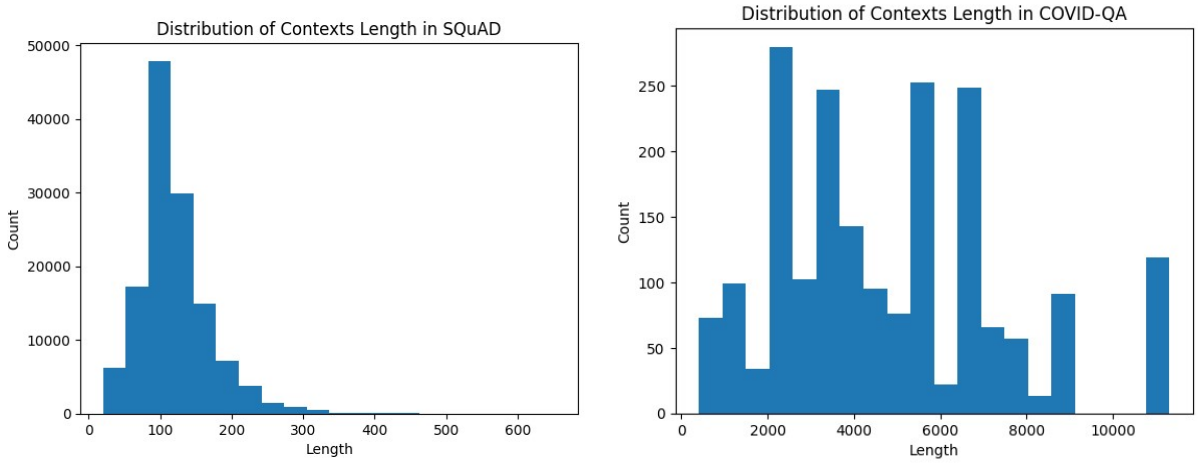


Figure 3.5: Distribution of Context Length

Questions and Answers Length The question types in the SQuAD and COVID-

QA datasets are similar as figure 3.6 shows. However, there are some stark differences in the lengths of the answers. In SQuAD, 97.6% of the answers consist of five or fewer tokens. In contrast, only 35% of the answers in COVID-QA have fewer than five tokens. The rest of the answers are even longer than that, with the longest clocking in at 144 tokens.

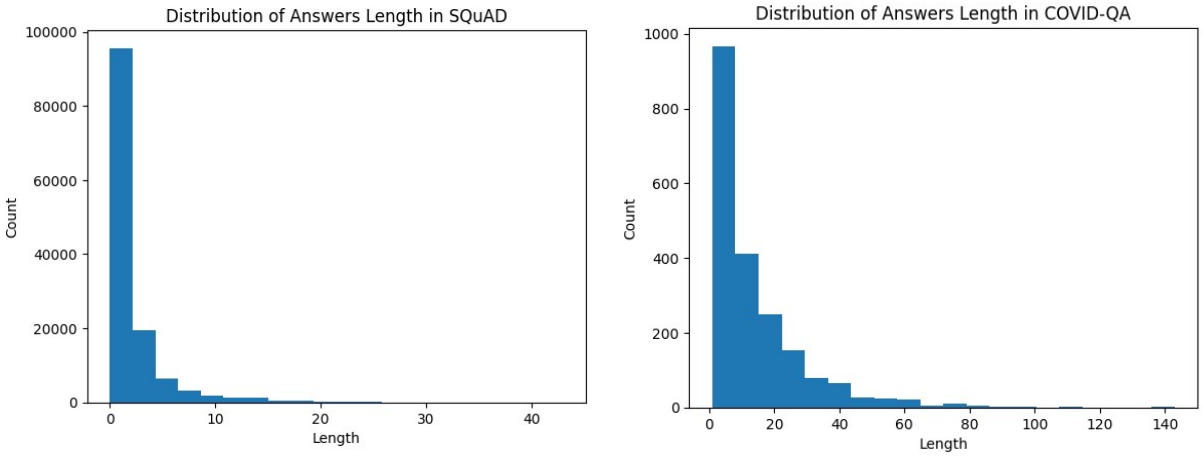


Figure 3.6: Distribution of Answers Length

3.4.4 Used Datasets for Evaluation

To evaluate the performance of our QA models, we utilize the SQuAD v2 dataset, which serves as a reliable benchmark for understanding and extracting answers from contextual information. Additionally, we employ the COVID-QA dataset to further assess the capabilities of our models in answering questions related to Covid-19. We measure the performance of our QA models using the same evaluation metrics as SQuAD v2, namely the macro-averaged F1 score and the EM. These metrics allow us to evaluate the effectiveness of our models in accurately extracting answers from the given context.

3.5 Building up the Proposed QA Model

3.5.1 Language Models

Bert

The BERT model is built using a stack of bidirectional Transformer encoders [59]. Its architecture is defined by three important parameters: the number of layers (transformer

blocks), the hidden size, and the number of self-attention heads in each transformer block. These parameters play a crucial role in defining the distinct versions of BERT, which exhibit variations in capacity and functionality. Among the notable versions, two of the most important ones are:

BERT Base: It uses 12 Transformer layers, and 110 million parameters, and is suitable for training on small-scale datasets [70]. For our work, we employed this version of BERT.

BERT Large: It uses 24 Transformer layers, and 340 million parameters, and is suitable for large-scale datasets with complex tasks [70].

This pre-trained NLP model has shown its effectiveness across different tasks, including QA. BERT-based QA systems are typically composed of three primary components: input encoding, contextual encoding, and answer generation. These components work together to process the input, understand its context, and generate accurate answers [71].

In the input encoding component, the question and context passage are tokenized into numerical representations using WordPiece tokenization. The tokens are then represented using **token embeddings** [2] Additionally, **segment embedding** [3] is added to differentiate between the question and context passage. The input encoding component combines the token and segment embeddings to capture the semantic meaning of the words and their relative position in the question and context passage.

In the contextual encoding component, BERT generates contextual embeddings for each token by taking into account the surrounding context of the token using a transformer architecture. The transformer architecture allows BERT to capture the relationships between each word and its surrounding context, resulting in a more accurate and nuanced understanding of the input text. BERT generates these embeddings by processing the token and segment embeddings generated in the input encoding component.

In the answer generation component, the contextual embeddings generated in the previous step are used to predict the start and end positions of the answer within the context passage. This is done using attention mechanisms and fully connected layers. The predicted start and end positions are then used to extract the answer from the passage, which is the output of the BERT-based QA system. The answer generation component

[2]Are numerical vectors that represent the meaning of each token, taking into account its position in the sentence.

[3]Is a numerical vector that is added to each token to indicate which segment it belongs to (the question or the context passage).

uses the contextual embeddings to capture the meaning of the words in their context and predict the most likely answer [11], as shown in figure 3.7.

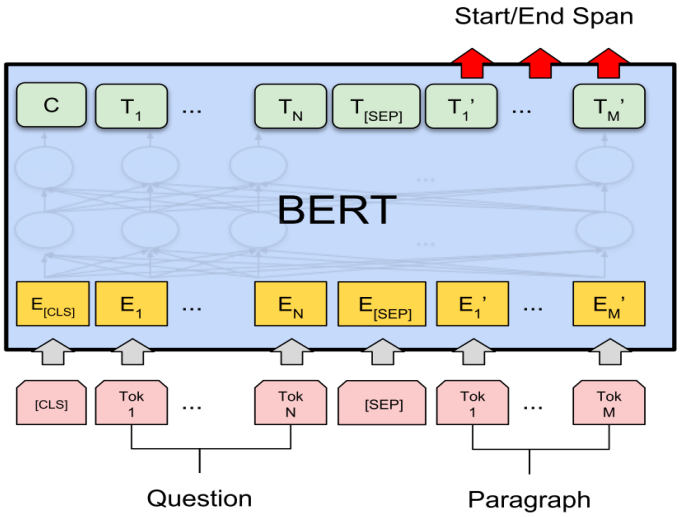


Figure 3.7: Building a Question Answering System with BERT [11]

RoBERTa

RoBERTa-based QA (Robustly Optimized BERT Pretraining Approach) [72] is an improved version of BERT-based QA, which is also a QA system based on pre-trained NLP models. While it shares the same underlying mechanism as BERT, except that it uses a byte-level BPE (Byte Pair Encoding) as a tokenizer. RoBERTa introduced notable improvements when it was introduced in 2019. These enhancements include training on larger datasets (with a total size of approximately 160 GB), utilizing a larger batch size of 8,000 with 300,000 steps, and removing the NSP Task due to observations that it did not provide significant benefits. Another improvement is the use of dynamic masking instead of static masking. As a result, RoBERTa has outperformed BERT in various NLP tasks, including natural language understanding and QA, and has achieved high rankings in several data science competitions [73].

In our work, we used these two language models in our solution that are readily available and can be directly loaded from Hugging Face [74], which is an open-source software library and community that offers a straightforward and user-friendly interface for working with state-of-the-art NLP models. It provides a convenient way to access and use these models, allowing us to leverage their advanced capabilities for our specific tasks.

Hyper-parameters

Table 3.3 presents the hyper-parameters we used for fine-tuning the two pre-trained models, BERT and RoBERTa, on the SQuAD and COVID-QA datasets.

The **'batch_size'** is a hyper-parameter that determines the number of samples processed in each training iteration. We set the batch size to a specific value, which determines how many examples are processed in parallel before updating the model weights.

The **'Max_length'** hyper-parameter represents the maximum size of input tokens. Any tokens exceeding this length are truncated to fit within the specified limit.

The **'Doc_stride'** refers to the stride size used when splitting documents into smaller chunks for processing. This parameter determines the overlapping or non-overlapping nature of the chunks.

The **'max_answer_length'** hyper-parameter defines the maximum allowed length for generated answers. Any generated answer longer than this limit is truncated or modified accordingly.

For optimization, we used the **Adam optimizer** with **weight decay** and a specific **learning rate**. The weight decay helps in controlling the growth of the model's parameters during training, while the learning rate determines the step size for updating the model weights.

All other hyper-parameters were kept as default settings, meaning they were not modified and used the predefined values provided by the model architecture.

Models	**BERT**		**RoBERTa**	
Datasets	SQUAD	COVID-QA	SQUAD	COVID-QA
max_length	384	512	384	512
doc_stride	128	170	128	170
train_batch_size	3	3	3	3
max_answer_length	30	300	30	300
learning Rate	1e-5	2e-5	1e-5	2e-5
num_train_epochs	2	2	2	2

Table 3.3: Hyper-parameters Comparison of Model

These hyper-parameters were carefully chosen and fine-tuned to achieve the best per-

formance for our models on the SQuAD and COVID-QA datasets.

3.5.2 Preprocessing

Before we can input the texts into our model, we need to perform preprocessing on them. This is to ensure that the texts are in a format that the model can understand. Preprocessing typically includes the following steps:

Tokenization

The tokenizer performs several tasks. Firstly, it breaks down the text into individual tokens, including words, punctuation marks, and special characters. Additionally, it adds special tokens like "[CLS] and [SEP]" for BERT, "<s> and </s>" for RoBERTa, at the beginning and end of the text. These special tokens play an important role in helping the model distinguish the question from the context during the process.

For this purpose, we use the Huggingface Transformer Tokenizer which will ensure that we get a tokenizer that corresponds to the model architecture we want to use.

One important aspect of preprocessing is handling very long documents in QA tasks. Typically, in other tasks, we truncate long documents when they exceed the model's maximum sentence length. However, in QA, removing part of the context may result in losing the answer we are searching for.

To address this, we employ a strategy where one example in our dataset can produce multiple input features, each of which is equal to or shorter than the maximum length of the model (controlled by the hyper-parameter **max_length**). This allows us to retain the context and minimize the risk of losing the answer. Additionally, to ensure we don't miss the answer if it lies at the point where we split a long context, we introduce some overlap between the generated features. This overlap is controlled by the hyper-parameter **'doc_stride'**. By incorporating overlap, we increase the chances of capturing the answer within one of the input features.

Another issue that we have to take care of is avoiding truncating the question. In fact, the question remains intact while we focus on handling the length of the context. This is achieved by setting the truncation parameter to **"only_second"**, which means only the second part (the context) will be truncated if it exceeds the maximum length specified by a hyper-parameter.

Finally, we configure the parameter **return_overflowing_tokens** to **True** with the tokenizer. it enables the generation of a list of features that may exceed the maximum length limit. This functionality is particularly useful when dealing with long documents or contexts In addition to this, the tokenizer also provides us with the **offset_mapping**=**True** for each token. The **offset_mapping** gives us the character-level start and end offsets for each token, allowing us to establish a mapping between the tokens and their corresponding positions in the original text.

Determining the End Index of Answers

As we observe in the datasets, the answers are represented as dictionaries containing the answer text and an integer denoting the start index of the answer within the context. However, BERT (and RoBERTa) models necessitate both the start and end positions of the answer. Since SQuAD or COVID-QA datasets do not provide the end index of the answer within the context, we need to determine it ourselves.

Sometimes, the answers in the dataset may be off by one or two characters. To account for this, we need to make adjustments to the start and end positions of the answers when mapping them to the tokenized context. This ensures that we accurately align the answer boundaries with the corresponding tokens.

Converting Answer Positions to Token-based Positions

Next, we convert the start and end positions of the answers from character-based positions to token-based positions. When working with the Hugging Face Tokenizers library, we utilize the built-in **char_to_token()** method. This method allows us to map the character positions to their corresponding token positions in the tokenized context. By doing so, we ensure that the answer boundaries are aligned with the appropriate tokens for further processing.

3.5.3 Fine-tuning

Setting the Model

Now that our data is prepared for training, we can proceed to download the pretrained model and fine-tune it. For our QA task, we will utilize the **AutoModelForQues-**

tionAnswering class, which is designed specifically for this purpose. Similar to the tokenizer, we can use the **from_pretrained** method, which will automatically download and cache the model for us. In addition to the model and tokenizer, we also need to create an optimizer for the fine-tuning process. We will use the PyTorch optimizer **AdamW**, which incorporates gradient bias correction and weight decay.

To set up the optimizer, we need to specify the **learning rate (lr)** and pass the model parameters to it.

Training the Model

To train the model, we utilize the **train** method, which trains the model for a specified number of epochs. Each epoch represents a complete pass through the training data. After each epoch, the model's performance is evaluated on the validation data. The evaluation of the validation data helps us determine whether to continue training or stop training based on the model's performance.

3.5.4 Evaluation

To evaluate our model, we need to associate the predictions of the model with specific segments of the context. The model generates logits[4] that indicate the probable starting and ending positions of the answers.

We have one logit for each feature and each token. The most obvious thing to predict an answer for each feature is to take the index for the maximum of the start logits as a start position and the index for the maximum of the end logits as an end position.

This will work great in a lot of cases, but what if this prediction gives us something impossible: the start position could be greater than the end position, or point to a span of text in the question instead of the answer.

To handle this, we need to first get the model's predictions for the start and end positions of the answers. We can then use these start and end logits to generate a list of possible answers. We can filter out any answers that are not valid, such as those with a start position greater than the end position or those that point to a span of text in the question instead of the answer.

[4]Are the raw output values that predict logits for the start and end positions of the answer span within the given context.

Once we have a list of valid answers, we can sort them by their score in descending order and keep the best one. The score is calculated by adding the start and end logits for each answer.

Finally, the best answer is added to the list of predictions. This process is repeated for each set of start and end logits in the raw predictions, resulting in a list of predictions for the given input.

To compute the metrics, we used the official evaluation script for SQuAD version 2.0. This script takes as input the list of predictions and the ground truth answers, and it outputs the F1 score and EM metrics.

3.6 Conclusion

This chapter has presented the proposed approach for QA using transformers for the biomedical domain. The system architecture of the proposed QA model was described, highlighting the utilization of language models such as BERT and RoBERTa. Additionally, the chapter explored the step-by-step process of building the model, which included data exploration, preprocessing techniques like tokenization and answer position determination, and the subsequent fine-tuning and training procedures. Finally, the evaluation of the proposed QA model was presented, showcasing its performance.

In the next chapter, we will delve into a comprehensive review and discussion of the results obtained from the evaluation of the proposed QA model. Furthermore, we will address the challenges and difficulties encountered throughout the development and implementation process.

Chapter 4

Experimental Results and Discussion

4.1 Introduction

In this chapter, we analyze and discuss our research findings, building upon the groundwork laid in the preceding chapters.

We begin by providing an overview of the experimental setup used in our study, including the software and hardware utilized.

Next, we delve into the results and discussion section. We highlight the significant performance improvements achieved through the fine-tuning process. We examine the effects of further fine-tuning and compare the performances of the BERT and RoBERTa models. We analyze performance trends through curves and explore the distribution of prediction answer lengths. We then focus on visualizing attention mechanisms and conduct a comparison between actual and predicted answers of our model. Finally, we address the limitations and challenges encountered during our research.

4.2 Experimental Setup

The following Table 4.1 provides a comprehensive summary of the experimental setup used in this study, encompassing the hardware and the software employed and formed the foundation of our experimental setup, enabling rigorous testing and evaluation of the proposed models.

Hardware and Training	**Personal Computer**	• CPU: Intel(R) Core(TM) i5-7200U • RAM: 16GB • Disk: 1TB SSD
		• CPU: Intel(R) Core(TM) i7-5200U • RAM: 16GB • Disk: 1TB
	Cloud Tools	**Google Colaboratory (Colab), by Google :** • CPU: Intel(R) Xeon(R) • GPU: Tesla T4, 16GB • RAM: 12.7GB • Disk: 107.7GB
		Kaggle Notebooks, by Kaggle : • CPU: Intel(R) Xeon(R) • GPU: Tesla P100-PCIE-16GB • RAM: 13 GB • Disk: 107.37 GB
Software and Librairies	**Programming Language**	Python 3
	Librairies	**Hugging Face Datasets:** a Python library for loading and preprocessing datasets. It provides a simple and unified interface for loading datasets from a variety of sources, such as CSV files, JSON files, and HDF5 files. It also provide a variety of preprocessing functions, such as tokenization, normalization, and filtering.
		Hugging Face TokenizersFast: state-of-the-art tokenizers library, optimized for both research and production. It provides an implementation of today's most used tokenizers in Transformers, with a focus on performance and versatility
		Hugging Face Transformers: a popular open-source Python library for NLP tasks. It provides a number of pretrained transformers models and a framework for fine-tuning these models on custom tasks.
		PyTorch: an open-source machine learning framework using Python, based on the Torch library. It is used for a wide variety of tasks, including natural language processing, computer vision, and robotics. PyTorch is known for its flexibility and ease of use.

Table 4.1: The experimental setup used

4.3 Results and Discussion

To present the results obtained from our models, we provide a detailed explanation of the findings as outlined in Table 4.2, Figures 4.1, and 4.2 . We evaluated our model based on two standard metrics: F1 score and EM score. We used the same evaluation dataset, COVID-QA, for all of our experiments.

The BERT model, fine-tuned on COVID-QA without any further fine-tuning, achieved an F1 score of 0.33 and an EM score of 0.14. When we fine-tuned it on SQuAD without any further fine-tuning, it achieved an F1 score of 0.04 and an EM score of 0.07. After fine-tuning the BERT model on the SQuAD dataset and further fine-tuning it on COVID-QA, we observed an improvement in performance, with the F1 score increasing to 0.58 and the EM score improving to 0.3.

The RoBERTa model, after being fine-tuned only on the COVID-QA dataset, achieved an F1 score of 0.48 and an EM score of 0.22. Similarly, when we fine-tuned the model on the SQuAD dataset without any further fine-tuning, it maintained a similar performance with an F1 score of 0.48 and an EM score of 0.18. However, by performing additional fine-tuning on the SQuAD dataset and then on the COVID-QA dataset, we observed a significant improvement in performance. The F1 score increased to 0.64, indicating a better ability to provide accurate answers, and the EM score improved to 0.38, indicating a higher rate of exact matches between the predicted and ground truth answers.

Our previous experiments allowed us to choose the RoBERTa model fine-tuned on SQuAD and further fine-tuned on COVID-QA. This model achieved the best performance on the COVID-QA dataset.

Model	**Fine-tuning**	**Further Fine-tuning**	**F1**	**EM**
BERT-base-uncased	COVID-QA		0.19	0.05
	SQuAD		0.07	0.04
	SQuAD	COVID-QA	0.58	0.30
RoBERTa-base	COVID-QA		0.48	0.22
	SQuAD		0.48	0.18
	SQuAD	**COVID-QA**	**0.64**	**0.38**

Table 4.2: Performance Comparison of Models

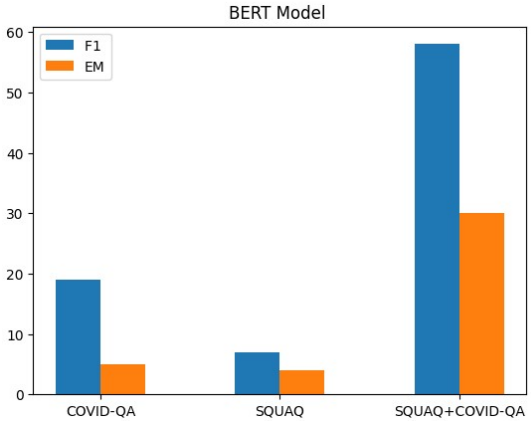


Figure 4.1: Result Comparison Of BERT Model

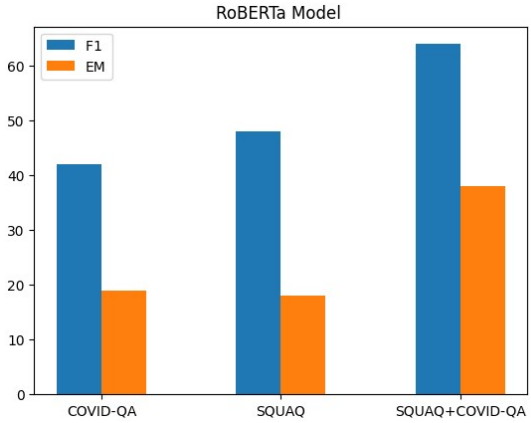


Figure 4.2: Result Comparison Of RoBERTa Model

After training the models for 5 epochs, we analyzed the progression of the EM and F1 score, as shown in Figures 4.3 and 4.4. Both metrics gradually increased during the initial two epochs, indicating that the model was able to learn and improve its performance during this period. However, beyond these two epochs, the scores reached a plateau, suggesting that the model had already captured the majority of the relevant information from the training data.

It is worth mentioning that the proposed model was trained for two epochs, as mentioned previously in Chapter 3. This decision was made based on the observation that the best results in terms of the EM and F1 scores were achieved within this limited training timeframe.

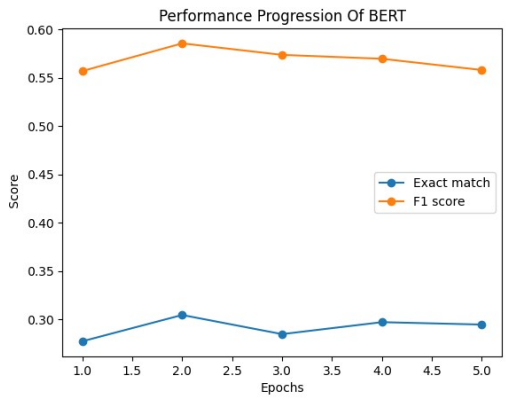


Figure 4.3: BERT Result curve

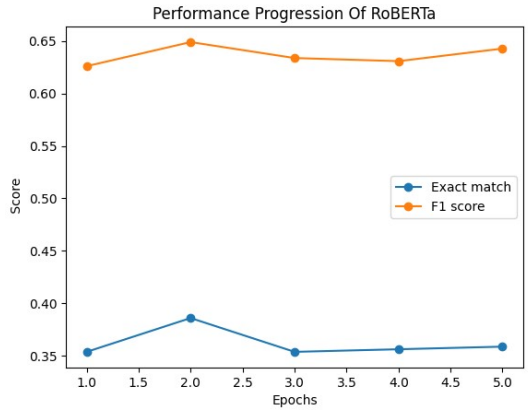


Figure 4.4: RoBERTa Result curve

Figure 4.5 illustrates the distribution of predicted answers length by the model. The figure showcases that the model is capable of predicting answers of varying lengths, ranging from short to long. This indicates the model's ability to adapt and generate answers of different lengths based on the context and question at hand. It is important to note that the length of the predicted answers is influenced by the context and question being addressed.

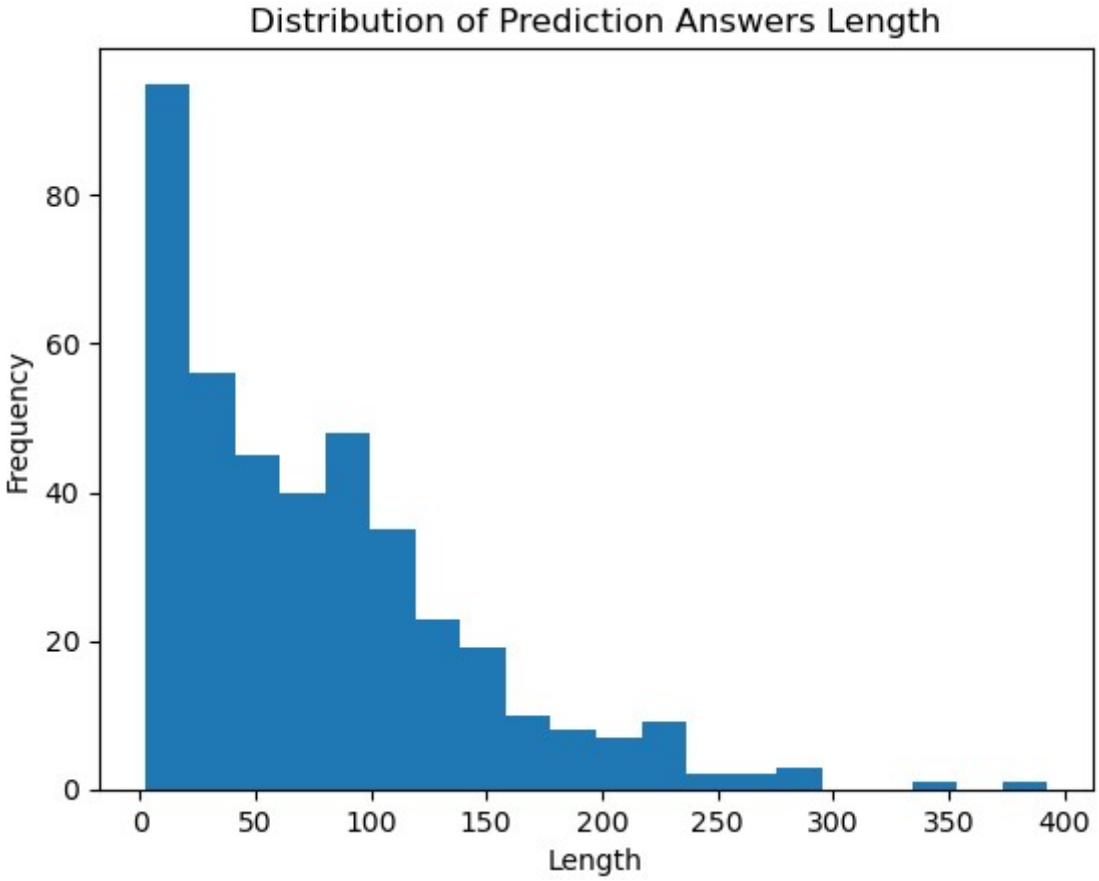


Figure 4.5: Distribution of Prediction Answers Length

Figures 4.6 and 4.7 show that as the length of the answer increases, both the average F1 score and EM score tend to augment. This suggests that longer answers provide more information that can be leveraged by the model, leading to improved performance in terms of both the F1 score and EM score. However, the figures also highlight the presence of variability in the scores, even for answers of the same length. This variability can be attributed to a number of factors, including:

The varying levels of difficulty among the questions. Some questions may inherently pose more challenges, resulting in lower scores, while others may be comparatively easier,

leading to higher scores.

The model's performance is influenced by factors such as question complexity, linguistic nuances, and potential ambiguity, all of which can impact its ability to consistently provide correct answers.

The results suggest that the model generates accurate and informative answers as the length of the answer increases. However, the model's performance is not perfect, and it is important to consider that the model may not always be able to answer them correctly.

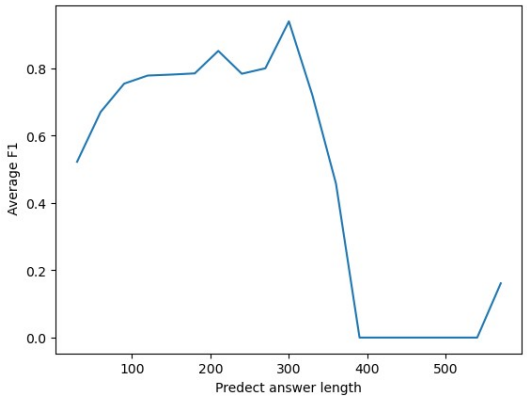


Figure 4.6: Avg F1 Based on Predicted Answers Length

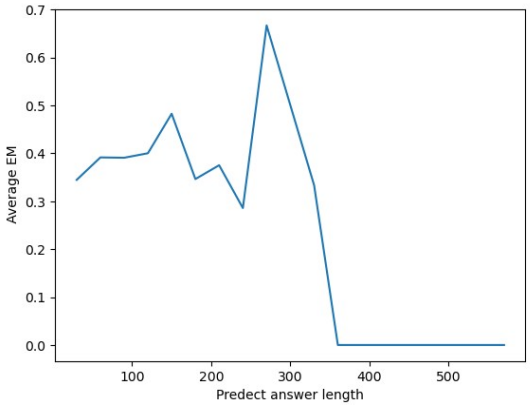


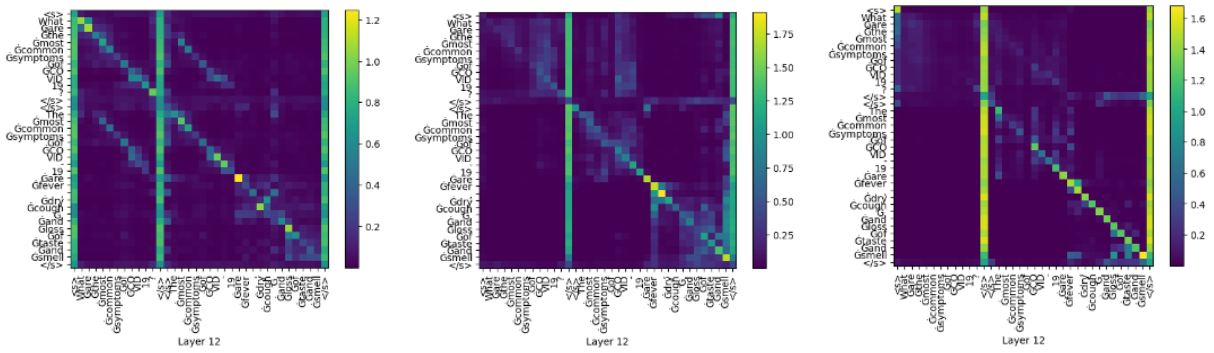
Figure 4.7: Avg EM Based on Predicted Answers Length

4.4 Visualizing Attention

In addition to performance metrics, analyzing the attention mechanism of QA models can provide insights into how the model attends to different parts of the input text when generating answers. Attention mechanisms allow the model to focus on relevant information and disregard irrelevant or redundant information. By visualizing the attention weights, it is possible to understand which parts of the input text are most influential in generating the model's predictions.

The **output_attentions** tensor represents attention matrices, also known as attention probabilities, for all 12 layers and all 12 heads. It represents a softmax-normalized dot product between the key and query vectors. In the literature, it has been used as an important indicator of how much a token relates to another token in the text. In the case of the QA model, it indicates which tokens relate to each other in the question, text, or answer segment.

Below Fig 4.8 shows visualizing token-to-token attention scores for the last layer over the three steps base model, fine-tuning on the SQuAD, and further fine-tuning on the COVID-QA. We see that the attention augments over the steps. As depicted in Figure 4.8a, the base model allocated relatively equal attention to all input tokens, including both the question and context. Conversely, Figure 4.8b demonstrates that the model fine-tuned on SQUAD exhibited attention primarily focused on the context tokens relevant to the question. Although this model provided correct answers, it is evident that the model fine-tuned on COVID-QA, as shown in Figure 4.8c, displayed the most optimal attention.



(a) RoBERTa Base

(b) RoBERTa Base Finetuned on SQUAD

(c) RoBERTa Base Further Fine-tuned on COVID-QA

Figure 4.8: Visualizing Token-To-Token Attention Scores for Three Versions of RoBERTa Model

4.5 Test of the proposed QA model

To test our proposed QA model for the biomedical domain, a list of queries was prepared, consisting of random questions like:

- Why is the disease being called coronavirus disease ?
- What is the main cause of HIV-1 infection in children ?
- What are the most common symptoms of COVID-19 ?
- Why have antiretroviral medications had limited benefit in treating influenza ?

Table 4.3 shows the outputted answers to these questions by the proposed Roberta model. We see a total success of the QA task. The proposed model is able to correctly answer all the given questions.

Question	**Real Answer**	**Our Model's Answer**
Why is the disease being called coronavirus disease?	It is part of the coronavirus family of viruses that cause respiratory infections	It is part of the coronavirus family of viruses that cause respiratory infections
What is the main cause of HIV-1 infection in children?	Mother-to-child transmission (MTCT)	Mother-to-child transmission (MTCT)
What are the most common symptoms of COVID-19?	fever, dry cough, and loss of taste and smell	fever, dry cough, and loss of taste and smell
Why have antiretroviral medications had limited benefit in treating influenza?	drug resistance and frequent antigenic mutation	drug-resistance and frequent antigenic mutation

Table 4.3: Testing the proposed Biomedical QA Model

4.6 Limitations and Challenges

During our research and experiments, we have encountered several challenges and limitations that can we summarized as follows: First, we used the base version of the models due to resource constraints. The base version of models has a limited number of parameters and may not capture as much fine-grained information as the larger versions, such as BERT Large or BERT Huge. As a result, our model's performance may be relatively lower compared to what could be achieved with a larger model.

Second, the model is trained on English data, so it may not be able to understand or QA in other languages. To improve its ability to answer questions in multiple languages, it can be trained on multilingual datasets.

Additionally, the model is trained on SQuAD, which enables it to answer questions effectively. However, its performance may be limited when it comes to answering questions in specific domains other than COVID-19. To improve the model's ability to handle multiple domains, it can be trained on multi-domain datasets. By training the model on multilingual and multi-domain datasets, we can improve its ability to answer questions in different languages and about different topics.

Finally, One significant challenge was the extensive time required to run the notebook.

We had to wait for several hours, sometimes up to nine hours, to complete the training and see the results of changing some hyper-parameters.

To train the model, we needed GPU resources. We turned to cloud solutions, such as Google Colab or Kaggle which is paid. However, this introduced new problems. There were instances where our Google account connection was unexpectedly disconnected or our GPU resources were depleted, resulting in wasted time and effort. Also, when we tried to train larger versions of models, the RAM would saturate. We also encountered this problem when training the COVID-QA dataset, which has a very long context. This forced us to choose a maximum input length.

4.7 Conclusion

We presented in this chapter the experimental results and discussion of our trained question-answering model specifically for COVID-19-related queries. We began by outlining the experimental setup employed in our study.

The subsequent Results and Discussion section presented a comprehensive analysis of the model's performance. A significant aspect of our analysis involved visualizing attention, which allowed us to gain a deeper understanding of the factors that influence further fine-tuning.

Furthermore, we presented examples of the testing of the proposed QA model, and the exploration of limitations and challenges provided valuable insights into the model's performance and underlying mechanisms.

Conclusion and Future Perspectives

The field of Automatic QA had experienced significant evolution in recent years. This was due to the incorporation of deep learning and neural network architectures, such as recurrent neural networks (RNNs), long short-term memory networks (LSTMs), bidirectional LSTMs (BLSTMs), and transformers. The use of transfer learning techniques had further enhanced the performance of QA systems and models, enabling improved question answering across various domains, including the biomedical field, COVID-19-related queries, and many other subjects.

In this thesis, we proposed a training method for Question Answering (QA) using transformer models, leveraging deep learning techniques specifically in the context of the COVID-19 domain. We provided a comprehensive overview of QA, covering its historical background, classifications, and evaluation methodologies. We extensively discussed the significance of Deep Learning (DL) and transformer architectures in the field of Natural Language Processing (NLP) and thoroughly reviewed existing works related to QA.

We presented the proposed QA model, including its overall architecture, system design, and detailed explanations of the datasets used for training and evaluation. The construction process of the model encompassed language models, preprocessing techniques, and fine-tuning procedures.

Throughout this thesis, we introduced the field of QA by defining and explaining the general process for training transformer-based models. The experimental results showcased the effectiveness and potential of this approach. However, we acknowledge the limitations and challenges encountered during the research. To address these limitations and overcome the identified challenges, further investigations and advancements in the field are warranted.

In future research and development of QA using deep learning techniques, we aim to explore the set of models that utilize different deep learning layers and attention mechanisms.

Training with larger models and more GPUs. This will allow the system to learn more complex relationships between words and phrases, and it will also allow us to train the models faster.

Training with other datasets, such as BioASK. This will help the system answer a wider range of questions, including those that are not related to COVID-19.

Adding a document retrieval module, This module will allow the system to retrieve relevant documents or passages that contain potential answers to questions. This will help the system provide more comprehensive and informative answers.

Bibliography

[1] D. Jurafsky and J.H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall series in artificial intelligence. Pearson Prentice Hall, 2009.

[2] Ramachandra Rao and Sanjay Kamath. *Question Answering with Hybrid Data and Models*. PhD thesis, Université Paris-Saclay, 2020.

[3] AI, machine learning, deep learning - schéma • Regional-IT — regional-it.be. `https://www.regional-it.be/detached/cette-intelligence-artificielle-que-nous-ne-comprenons-plus/ai-machine-learning-deep-learning-schema/`. [Accessed 15-Jun-2023].

[4] Ivan Nunes Da Silva, Danilo Hernane Spatti, Rogerio Andrade Flauzino, Luisa Helena Bartocci Liboni, Silas Franco dos Reis Alves, Ivan Nunes da Silva, Danilo Hernane Spatti, Rogerio Andrade Flauzino, Luisa Helena Bartocci Liboni, and Silas Franco dos Reis Alves. *Artificial neural network architectures and training processes*. Springer, 2017.

[5] Mohammed Saber, El Rharras Abdessamad, Saadane Rachid, Hatim Kharraz Aroussi, and Mohammed Wahbi. Artificial neural networks, support vector machine and energy detection for spectrum sensing based on real signals. *International Journal of Communication Networks and Information Security*, 11, 04 2019.

[6] Deep learning - section 3. `https://note.zbmain.com/ainote/dlnew/deeplearning/section3/`. Accessed: March 17, 2023.

[7] Khalid Nassiri and Moulay Akhloufi. Transformer models used for text-based question answering systems. *Applied Intelligence*, pages 1–34, 2022.

[8] Woohyun Kim, Yerim Han, Kyoung Kim, and Kwan-Woo Song. Electricity load forecasting using advanced feature selection and optimal deep learning model for the variable refrigerant flow systems. *Energy Reports*, 6:2604–2618, 11 2020.

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[10] Momina Qazi, Muhammad Usman Shahid Khan, and Mazhar Ali. Detection of fake news using transformer model. pages 1–6, 01 2020.

[11] Munazza Zaib, Dai Hoang Tran, Subhash Sagar, Adnan Mahmood, Wei Emma Zhang, and Quan Z. Sheng. Bert-coqac: Bert-based conversational question answering in context. *CoRR*, abs/2104.11394, 2021.

[12] National library of medicine. Retrieved from `https://www.nlm.nih.gov/bsd/stats/cit_added.html`.

[13] Khalid Nassiri and Moulay Akhloufi. Transformer models used for text-based question answering systems. *Applied Intelligence*, pages 1–34, 2022.

[14] Timo Möller, Anthony Reina, Raghavan Jayakumar, and Malte Pietsch. COVID-QA: A question answering dataset for COVID-19. In *Proceedings of the 1st Workshop on NLP for COVID-19 at ACL 2020*, Online, July 2020. Association for Computational Linguistics.

[15] Ali Mohamed Nabil Allam and Mohamed Hassan Haggag. The question answering systems: A survey. *International Journal of Research and Reviews in Information Sciences (IJRRIS)*, 2(3), 2012.

[16] Dan Moldovan, Marius Pasca, Sanda Harabagiu, and Mihai Surdeanu. Performance issues and error analysis in an open-domain question answering system. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 33–40, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics.

[17] Amit Mishra and Sanjay Kumar Jain. A survey on question answering systems with classification. *Journal of King Saud University-Computer and Information Sciences*, 28(3):345–361, 2016.

[18] Andrew Lampert. A quick introduction to question answering. *Dated December*, 2004.

[19] Dan Moldovan, Sanda Harabagiu, Marius Pasca, Rada Mihalcea, Roxana Girju, Richard Goodrum, and Vasile Rus. The structure and performance of an open-domain question answering system. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, pages 563–570, Hong Kong, October 2000. Association for Computational Linguistics.

[20] Ravindra Hegadi, Deepa Yogish, and Manjunath T N. A survey of intelligent question answering system using nlp and information retrieval techniques. *International Journal of Advanced Research in Computer and Communication Engineering*, 5:536–540, 05 2016.

[21] Ajitkumar M Pundge, SA Khillare, and C Namrata Mahender. Question answering system, approaches and techniques: a review. *International Journal of Computer Applications*, 141(3):0975–8887, 2016.

[22] Marco Antonio Calijorne Soares and Fernando Silva Parreiras. A literature review on question answering techniques, paradigms and systems. *Journal of King Saud University-Computer and Information Sciences*, 32(6):635–646, 2020.

[23] Marco Antonio Calijorne Soares and Fernando Silva Parreiras. A literature review on question answering techniques, paradigms and systems. *Journal of King Saud University - Computer and Information Sciences*, 32(6):635–646, 2020.

[24] Raffaella Bernardi, Valentin Jijkoun, Gilad Mishne, and Maarten Rijke. Selectively using linguistic resources throughout the question answering pipeline. 01 2004.

[25] Sadid A Hasan and Oladimeji Farri. Clinical natural language processing with deep learning. *Data Science for Healthcare: Methodologies and Applications*, pages 147–171, 2019.

[26] Zhen Huang, Shiyi Xu, Minghao Hu, Xinyi Wang, Jinyan Qiu, Yongquan Fu, Yuncai Zhao, Yuxing Peng, and Changjian Wang. Recent trends in deep learning based open-domain textual question answering systems. *IEEE Access*, 8:94341–94356, 2020.

[27] Mourad Sarrouti, Abdelmonaime Lachkar, and Said El Alaoui Ouatik. Biomedical question types classification using syntactic and rule based approach. In *2015 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K)*, volume 1, pages 265–272. IEEE, 2015.

[28] Bolanle Ojokoh and Emmanuel Adebisi. A review of question answering systems. *Journal of Web Engineering*, 17(8):717–758, 2018.

[29] Christof Monz. Document retrieval in the context of question answering. In Fabrizio Sebastiani, editor, *Advances in Information Retrieval*, pages 571–579, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[30] Aïssatou Diallo, Markus Zopf, and Johannes Fürnkranz. Learning analogy-preserving sentence embeddings for answer selection. *arXiv preprint arXiv:1910.05315*, 2019.

[31] Sanjay Kamath Ramachandra Rao. *Question Answering with Hybrid Data and Models*. Theses, Université Paris-Saclay, February 2020.

[32] Kai Ming Ting. *Confusion Matrix*, pages 209–209. Springer US, Boston, MA, 2010.

[33] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics.

[34] Sanjay Kamath, Brigitte Grau, and Yue Ma. An adaption of BIOASQ question answering dataset for machine reading systems by manual annotations of answer spans. In *Proceedings of the 6th BioASQ Workshop A challenge on large-scale biomedical semantic indexing and question answering*, pages 72–78, Brussels, Belgium, November 2018. Association for Computational Linguistics.

[35] Qiao Jin, Bhuwan Dhingra, Zhengping Liu, William W Cohen, and Xinghua Lu. Pubmedqa: A dataset for biomedical research question answering. *arXiv preprint arXiv:1909.06146*, 2019.

[36] Khadidja METTAS and Maissa Fatna RAI. *A Deep Learning Model for Text-based CAPTCHA Breaking*. PhD thesis, , 2020.

[37] J.D. Kelleher. *Deep Learning*. The MIT Press Essential Knowledge series. MIT Press, 2019.

[38] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. Adaptive Computation and Machine Learning series. MIT Press, 2016.

[39] Jyh-Woei Lin. Artificial neural network related to biological neuron network: a review. *Advanced Studies in Medical Sciences*, 5(1):55–62, 2017.

[40] Sonali B Maind, Priyanka Wankar, et al. Research paper on basic of artificial neural network. *International Journal on Recent and Innovation Trends in Computing and Communication*, 2(1):96–100, 2014.

[41] Ivan Nunes da Silva, Danilo Hernane Spatti, Rogerio Andrade Flauzino, Luisa Helena Bartocci Liboni, and Silas Franco dos Reis Alves. *Artificial Neural Network Architectures and Training Processes*, pages 21–28. Springer International Publishing, Cham, 2017.

[42] Alain Droniou. *Apprentissage de représentations et robotique développementale: quelques apports de l'apprentissage profond pour la robotique autonome*. PhD thesis, Université Pierre et Marie Curie-Paris VI, 2015.

[43] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *Towards Data Sci*, 6(12):310–316, 2017.

[44] Anthony Brabazon, Michael O'Neill, and Seán McGarraghy. *Natural computing algorithms*, volume 554. Springer, 2015.

[45] Mingzhe Chen, Ursula Challita, Walid Saad, Changchuan Yin, and Mérouane Debbah. Machine learning for wireless networks with artificial intelligence: A tutorial on neural networks. *arXiv preprint arXiv:1710.02913*, 9, 2017.

[46] By Dustin Stansbury. A gentle introduction to artificial neural networks. 2020.

[47] Tao Lei et al. *Interpretable neural models for natural language processing*. PhD thesis, Massachusetts Institute of Technology, 2017.

[48] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[49] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[50] Michael I Jordan. Serial order: A parallel distributed processing approach. In *Advances in psychology*, volume 121, pages 471–495. Elsevier, 1997.

[51] Charu C Aggarwal et al. Neural networks and deep learning. *Springer*, 10(978):3, 2018.

[52] Sepp Hochreiter and Jürgen Schmidhuber. Lstm can solve hard long time lag problems. *Advances in neural information processing systems*, 9, 1996.

[53] Shu Zhang, Dequan Zheng, Xinchen Hu, and Ming Yang. Bidirectional long short-term memory networks for relation classification. In *Proceedings of the 29th Pacific Asia conference on language, information and computation*, pages 73–78, 2015.

[54] Arighna Chakraborty and Asoke Nath. Scope and challenges in conversational ai using transformer models. 2021.

[55] Andrea Galassi, Marco Lippi, and Paolo Torroni. Attention in natural language processing. *IEEE transactions on neural networks and learning systems*, 32(10):4291–4308, 2020.

[56] Md Tahmid Rahman Laskar et al. Utilizing the transformer architecture for question answering. 2020.

[57] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[58] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR, 2019.

[59] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[60] Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*, 2020.

[61] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

[62] Vladislav Lialin, Kevin Zhao, Namrata Shivagunde, and Anna Rumshisky. Life after bert: What do other muppets understand about language? *arXiv preprint arXiv:2205.10696*, 2022.

[63] Raphael Tang, Rodrigo Nogueira, Edwin Zhang, Nikhil Gupta, Phuong Cam, Kyunghyun Cho, and Jimmy Lin. Rapidly bootstrapping a question answering dataset for covid-19. *arXiv preprint arXiv:2004.11339*, 2020.

[64] Lucy Lu Wang, Kyle Lo, Yoganand Chandrasekhar, Russell Reas, Jiangjiang Yang, Darrin Eide, Kathryn Funk, Rodney Kinney, Ziyang Liu, William Merrill, et al. Cord-19: The covid-19 open research dataset. *ArXiv*, 2020.

[65] Jinhyuk Lee, Sean S Yi, Minbyul Jeong, Mujeen Sung, Wonjin Yoon, Yonghwa Choi, Miyoung Ko, and Jaewoo Kang. Answering questions on covid-19 in real-time. *arXiv preprint arXiv:2006.15830*, 2020.

[66] Dan Su, Yan Xu, Tiezheng Yu, Farhad Bin Siddique, Elham J Barezi, and Pascale Fung. Caire-covid: A question answering and query-focused multi-document summarization system for covid-19 scholarly information management. *arXiv preprint arXiv:2005.03975*, 2020.

[67] Jafar A Alzubi, Rachna Jain, Anubhav Singh, Pritee Parwekar, and Meenu Gupta. Cobert: Covid-19 question answering system using bert. *Arabian journal for science and engineering*, pages 1–11, 2021.

[68] Revanth Gangi Reddy, Bhavani Iyer, Md Arafat Sultan, Rong Zhang, Avi Sil, Vittorio Castelli, Radu Florian, and Salim Roukos. End-to-end qa on covid-19: domain adaptation with synthetic training. *arXiv preprint arXiv:2012.01414*, 2020.

[69] Hillary Ngai, Yoona Park, John Chen, and Mahboobeh Parsapoor. Transformer-based models for question answering on covid19. *arXiv preprint arXiv:2101.11432*, 2021.

[70] BERT — huggingface.co. `https://huggingface.co/docs/transformers/model_doc/bert`. [Accessed 23-May-2023].

[71] Ying-Hong Chan and Yao-Chung Fan. A recurrent BERT-based model for question generation. In *Proceedings of the 2nd Workshop on Machine Reading for Question Answering*, pages 154–162, Hong Kong, China, November 2019. Association for Computational Linguistics.

[72] RoBERTa — huggingface.co. `https://huggingface.co/docs/transformers/model_doc/roberta`. [Accessed 23-May-2023].

[73] Liu Zhuang, Lin Wayne, Shi Ya, and Zhao Jun. A robustly optimized bert pre-training approach with post-training. In *Proceedings of the 20th chinese national conference on computational linguistics*, pages 1218–1227, 2021.

[74] Hugging Face – The AI community building the future. — huggingface.co. `https://huggingface.co/`. [Accessed 23-May-2023].

[75] Dan I Moldovan, Sanda M Harabagiu, Marius Pasca, Rada Mihalcea, Richard Goodrum, Roxana Girju, and Vasile Rus. Lasso: A tool for surfing the answer net. In *TREC*, volume 8, pages 65–73, 1999.

[76] Jurafsky Daniel, Martin James H, et al. *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*. prentice hall, 2007.

[77] Md Tahmid Rahman Laskar, Jimmy Xiangji Huang, and Enamul Hoque. Contextualized embeddings based transformer encoder for sentence similarity modeling in answer selection task. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 5505–5514, Marseille, France, May 2020. European Language Resources Association.

[78] H Jabbar and Rafiqul Zaman Khan. Methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study). *Computer Science, Communication and Instrumentation Devices*, 70:163–172, 2015.

[79] Tanik Saikh, Sovan Kumar Sahoo, Asif Ekbal, and Pushpak Bhattacharyya. Covidread: A large-scale question answering dataset on covid-19. *arXiv preprint arXiv:2110.09321*, 2021.

[80] Google Colab — research.google.com. `https://research.google.com/colaboratory/faq.html?hl=fr#:~:text=Colaboratory%2C%20souvent%20raccourci%20en%20%22Colab,donn%C3%A9es%20et%20%C3%A0%20l'%C3%A9ducation.` [Accessed 24-May-2023].

[81] Kaggle: Your Machine Learning and Data Science Community — kaggle.com. `https://www.kaggle.com/`. [Accessed 26-May-2023].