ANALYSIS OF ANSWERING QUESTIONS USING AI BY CATEGORIZATION METHODS FOR TEXT

KUTLU ERMAN ÖZGİL

NOVEMBER 2021

ÇANKAYA UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

DEPARTMENT OF COMPUTER ENGINEERING
MASTER'S THESIS IN COMPUTER ENGINEERING

ANALYSIS OF ANSWERING QUESTIONS USING AI BY CATEGORIZATION METHODS FOR TEXT

KUTLU ERMAN ÖZGİL

NOVEMBER 2021

ABSTRACT

ANALYSIS OF ANSWERING QUESTIONS USING AI BY CATEGORIZATION METHODS FOR TEXT

ÖZGİL, Kutlu Erman
M.Sc. in Computer Engineering

Supervisor: Assist. Prof. Roya Choupani
November 2021, 58 pages

Question Answering (QA) is a Computer Engineering area which consists of multi-disciplinary fields Artificial Intelligence (AI), Information Retrieval (IR), and Natural Language Processing (NLP). The main aim of these QA systems is to build systems that can answer questions asked by humans in a natural language according to the given passage. This process was challenging for earlier computers because of the hardware limitations and lack of software models needed to complete the tasks, which took a very long time to complete. Today, Computer Hardware advancements, especially in GPU units, made it possible to complete tasks in parallel much faster. Also, the recent improvements and research in AI models and software made it possible to use Pre-Trained models to achieve this goal much faster.

In this thesis, one of the most popular models by Google, BERT (Bidirectional encoder representations from transformers), is Fine-Tuned, and the limitations are explored. A case study is made to understand how this Fine-Tuned model can help people in any area given. The results showed that working with large models and data sets still takes longer times for the training parts, and the Fine-Tuned Bert model performs better for the specific task it was designed.

Keywords: Question Answering, Natural Language Processing, Information Retrieval

ÖZ/ÖZET

METİN İÇİN KATEGORİZASYON YÖNTEMLERİYLE AI KULLANARAK SORU CEVAPLAMALARININ ANALİZİ

ÖZGİL, Kutlu Erman
Bilgisayar Mühendisliği Yüksek Lisans Tezi

Danışman: Dr. Öğr. Üyesi Roya CHOUPANII
Kasım 2021, 58 sayfa

Soru Cevaplama (QA); Yapay Zeka (AI), Bilgi Erişimi (IR) ve Doğal Dil İşleme (NLP) gibi çok disiplinli alanlardan oluşan bir Bilgisayar Mühendisliği alanıdır. Bu soru cevaplama sistemlerinin temel amacı, insanlar tarafından sorulan sorulara, verilen pasaja göre doğal bir dilde cevap verebilecek sistemler oluşturmaktır. Soru Cevaplama, donanım sınırlamaları ve tamamlanması çok uzun zaman alan görevleri tamamlamak için gereken yazılım modellerinin eksikliği nedeniyle önceki dönemlerde bilgisayarlar için zor bir görevdi. Günümüzde Bilgisayar Donanımı'nın hızlanması, özellikle de GPU birimlerindeki gelişmeler, paralel olarak görevleri çok daha hızlı tamamlamayı mümkün kılmıştır, ayrıca AI modellerinde ve yazılımlarında son zamanlarda yapılan iyileştirmeler ve araştırmalar, bu hedefe daha hızlı ulaşmak için önceden eğitimli modellerin kullanılmasını mümkün kılmıştır.

Bu tezde en popüler modellerden biri olan Google'in geliştirdiği BERT'in İnce Ayarları üzerinde analizler yapılarak sınırları anlamaya çalışıldı ve bu ince ayarlı modelin insanlara verilen herhangi bir alanda nasıl yardımcı olabileceğini anlamak için bir vaka çalışması yaptık. Yapılan çalışma sonucunda, eğitim parçaları için büyük modeller ve veri kümeleriyle çalışmanın hala uzun zaman aldığını ve İnce Ayarlı BERT modelinin tasarlandığı belirli görev için daha iyi performans gösterdiği sonucunu edindik.

Anahtar Kelimeler: Soru Cevaplama, Doğal Dil İşleme, Bilgi Erişimi.

ACKNOWLEGEMENT

I would like to express my sincere gratitude to my father Erol Özgil, my mother Zeliha Özgil, and my brother Göktuğ Özgil for their support and sacrifice to me. I would also like to thank my wife Hülya Özgil, for her complete support and my 2,5-year-old daughter İris Özgil for giving me the motivation for my thesis.

Special thanks to my Supervisor, Asst. Prof. Roya Choupani for the excellent guidance and providing me with an excellent atmosphere to conduct this research and her husband Asst. Prof. Reza Hassanpour who has always been an inspiring and a good model teacher for me. My special gratitude goes to the rest of the thesis committee, Assoc. Prof. Kasım Öztoprak and Asst. Prof. Abdül Kadir Görür for the encouragement and insightful comments.

I would also like to thank Asst. Prof. Özgün Selvi for the lectures and amazing ideas he gave me, also his wife Bilge İşlek Selvi and their two sons Özgün Borabey Selvi, Özgün Göktuğ Selvi for their understandings through this times.

I would also like to thank Okan Yüksel and his wife Elif Yüksel for their support and great ideas about my thesis subject. Also I would like to thank Leyla Kırlı and her husband Caner Kırlı and their two sons Ediz Kırlı, Yalın Kırlı for their support and understanding.

I would also like to thank my colleagues at work Emre Mersin and his wife, Elif Mersin, Gizem Burcu Kaya, Özge Aktan, and her husband, Fatih Aktan, for their support and great ideas at work during the hard-working times.

I would also like to thank the people who work in this QA area and gave much of their time improving these models, explaining them on their blogs, and creating youtube videos about this topic, trying to teach and share their knowledge with us. I recommend following Chris McCormick with his tutorials on youtube to clearly understand the BERT model and reading the visualization of transformers by Jay Almar to understand the model. I also recommend following the Google development team for their researches on BERT which they share their usage examples and new ways to adopt this technology. Without the researchers of these genius minds, this thesis work could not be completed.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

SYMBOLS:

MB	:Mega Bytes
M	:Million

ABBREVIATIONS

AI	:Artificial Intelligence
CNN	:Convolutional Neural Networks
RNN	: Recurrent neural networks
LSTM	:Long Short-Term Memory
IR	:Information Retrieval
IE	:Information Extraction
NLP	:Natural Language Extraction
BERT	:Bidirectional Encoder Representations from Transformers
SQuAD	:Stanford Question Answering Dataset
EM	:Exact Match
ML	:Machine Learning
QA	:Question Answering

INTRODUCTION

Question Answering is the ability of a computer to answer a given question in a natural human-understandable form. With the recent developments in this field and new models being introduced by different researchers, AI and Question Answering (QA) Sytems made considerable advancements in this field of research. With the introduction of “Attention Is All You Need” by Google [1], which later formed the basis of Transformers, more researchers started studying this topic. We can observe that recent studies focused more on this latest state-of-the-art Transformers model and created BERT-like models to gain advancement and improved scores. If we look at SQUAD[2] based QA models comparison, we can see that in late 2018 the Exact Match (EM) scores rose from 74 to 80 with its introduction. Later alternate versions like SemBERT[3], ROBERTA[4], SpanBERT[5], DistillBERT[6], ALBERT[7] took the scores higher up to 90’s and still improving. Also with the help of combining multiple models like SA-Net and ALBERT or Fine-Tuning BERT model helped in this advancement and their combination with other models.

The research has improved on QA tasks and even surpassed the human scores in this area, which showed promising results for the future of AI Computers. The advantage of this QA system is getting a direct answer from the source without the need to search, which gains time. On the other hand, there are still some disadvantages, like training these models takes much time and consumes a considerable amount of energy to achieve this QA Task.

For the future of this research Question Answering, we can see newly proposed models like T5 [8], which uses Text to Text transforming, eliminating the need for complex vectorial calculations but still needs more improvement and research. Also, XLNET [9], a generalized autoregressive pretraining model, was proposed giving promising results for QA Tasks' future. However, even with these models, we still can not achieve perfect scores, even on a single task such as Question Answering, and in the future, we dream of seeing Wide AI machines that can perfectly do multiple tasks. However, there is still a gap that the researchers must fill in before reaching those aims.

In this thesis, I tried to Fine-Tune a BERT model for a Case Study, and I tried to get better results on this specific area for Question Answering systems. The main question on my mind was how can we turn these studies into a functional QA System that can help people, and this brought the Idea of the Covid 19 QA System as a case study. So I started with gaining the knowledge stated in this thesis and then developing the QA System for Covid-19 to help people during this Pandemic. I prepared the dataset using the pre-trained BERT model, I Fine Tuned BERT, and trained on Covid Dataset to get better results. All the steps are stated in this thesis report with details. There are also many other researchers trying to create a perfect Codiv QA System. Also, there is a challenge, "COVID-19 Open Research Dataset Challenge (CORD-19)" by Kaggle; so many professionals and researchers compete in this area, so even if I am not competing, my case study can be considered one of them.

After the training with Fine Tuned BERT, we can check our results with universal metrics, Exact Match (EM), to analyze how well our model performed before and after the training to see our improvement.

CHAPTER I

GENERAL KNOWLEDGE

1.1 BACKGROUND

In general, Question Answering has been part of human interaction for many years, and it can be considered one of the main ways in human communication. Our ancestors used to communicate and answer questions with their hands and body. With the development of vocal cords and languages, modern-day people can use their voices and talk to answer questions.

The main idea of Question Answering is to get the desired information about an occasion. For example, a mother can ask her daughter where she was or why she was late, and her daughter will respond with an answer like I was at school or I was late because I missed the bus and had to walk home from school.

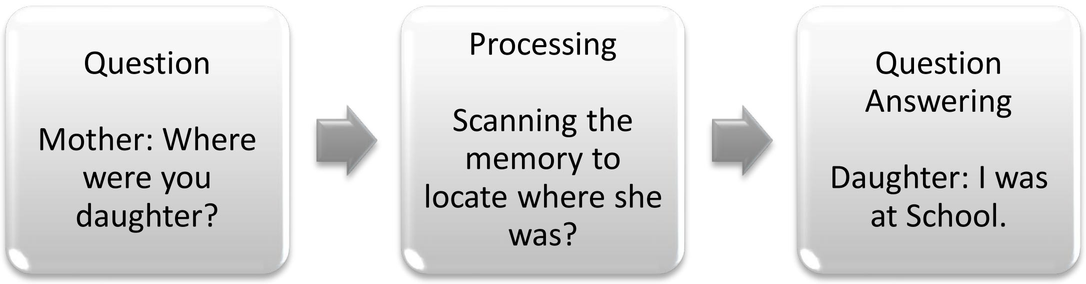

Question
Mother: Where were you daughter?
Processing
Scanning the memory to locate where she was?
Question Answering
Daughter: I was at School.

Figure 1: Example of Basic Question Answering

After the development of Computers, people tried to use this development in so many ways such as Mathematical computation, Financial computation, Word editing, and other uses were introduced, which gained popularity very fast. Alan Turing, mainly referred to as the creator of Modern Computer, had an idea of the first Question Answering system back in the 1950s and is also considered the basis of Artificial Intelligence. Turing proposed an “imitation Game” known as the Turing test, in which a human communicates with a machine via a teletype interface and asks questions about it.[10]

The main idea of the test was to put an AI Question Answering Computer and a Human on one side and make the user ask questions to random one of them and try to guess whether the answerer was a Computer or a Human being. Alan Turing believed that most the questions askers wouldn't be able to distinguish between the Human answerer and the Computer answerer one day.

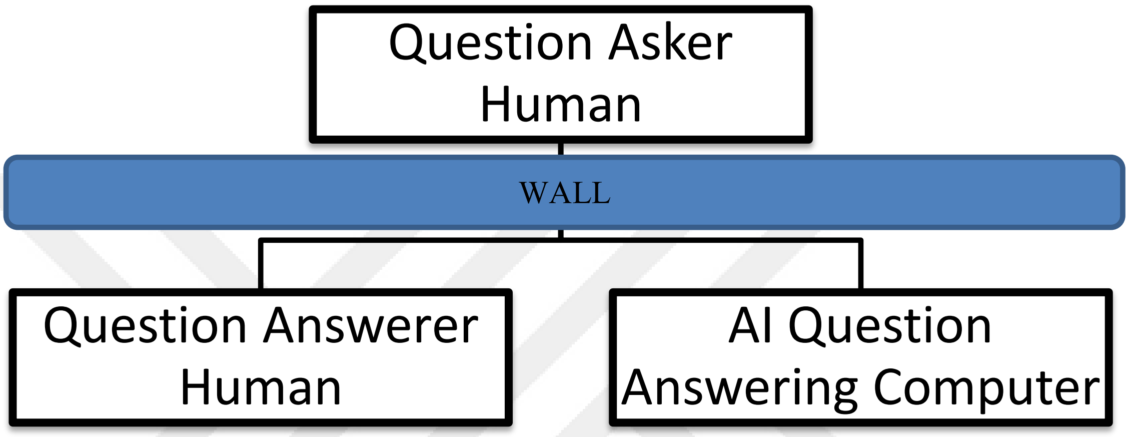


Figure 2: Alan Turing "Immitation Game" Visualization

1.2 HISTORY OF QUESTION ANSWERING

A comprehensive set of Question Answering System exists and was tried to accomplish different tasks in a State-Of-The-Art manner. One of the oldest question answering systems was BASEBALL[11] which attempted to answer baseball game questions yearly.

Over time, other QA Systems, such as ELIZA, was introduced in 1964, considered the first of the Chatbots—created to demonstrate the superficiality of communication between humans and machines. Eliza simulated conversation using a "pattern matching" and substitution methodology that gave users an illusion of understanding on the part of the program but had no built-in framework for contextualizing events.

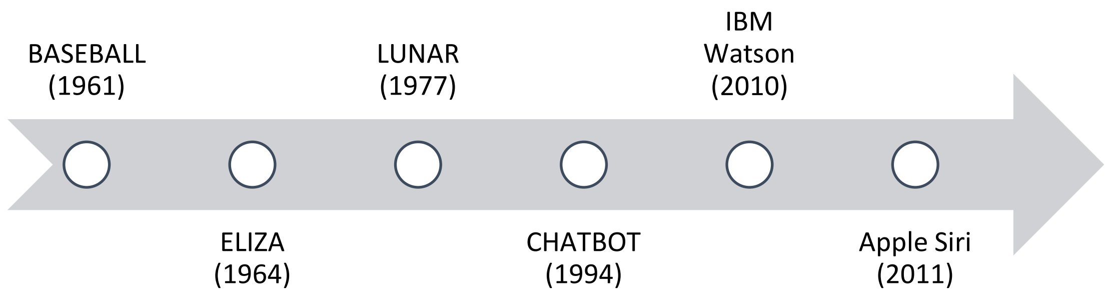


Figure 3 : QA Systems Timeline

There were also other QA Systems SHRDLU (1972), PHLIQA (1976), LUNAR (1977), GUS (1977), CHATBOT (1994), FAQ Finder (1995), ASK Jeeves (1996), START (1997), JAVELIN (2002), AnswerBag (2003), A9.com (2003), MIT's Jupiter System (2005), Yahoo! Answers (2005), Blurtit (2006), Evi (2007), Wolfram Alpha (2009), IBM Watson (2010), QUORA (2010), Apple Siri (2011), Google Now (2012), PARLIO (2014) [10]

With the advancements in technology, later works used Knowledge Bases for available domains, and Expert Systems started gaining popularity which was also used as part of some QA Systems.

1.3 TYPES OF QUESTIONS

1.3.1 Open Domain Questions

Open Domain Question Answering Systems are not restricted to any specific domain and provide a short answer to a question addressed in natural language. [12]

This means that the question can be of any area. Usually, these questions' answers can be found in broad knowledge areas like Encyclopedias, the web, or Wikipedia like sites.

An example question can be: Who won the Nobel Prize in Literature 2021?

The answer can be found searching the web in multiple web pages, so one can see that the winner is Abdulrazak Gurnah.

1.3.2 Closed Domain Questions

In the Closed domain QA system, there is a restriction of domain based on web, and questions are related to a specific domain. [12]

This actually means that questions will be from a specific domain like Law, Medical, Sport, Animals, Plants, or any specific area.

An example question can be of Sport Domain from the BASEBALL QA System: Where did the Red Sox play on July 7? [11]

Since the domain is specific on an area and the QA Systems only answer the domain-specific questions, it can be considered a smaller set when compared.

1.3.3 Factoid Type Questions

The factoid type questions commonly begin with wh-word, and examples can be what, which, when, who, how. [12]

As suggested by [13], these questions each have their question meaning and answer type accordingly. They shared a table indicating this relationship in their paper, which would help understand these question types in detail.

On the other hand, these questions usually have specific answers, and the correctness of the answers can be easily calculated.

This is the most used questioning type in our daily lives, and in this thesis work, we will be using factoid-type questions to train and evaluate our model with countable metrics.

1.3.4 List Type Questions

The list-type questions need a list of facts or entities as answers. [13]

These questions usually have many answers with changing dates or occurrences.

To understand this Question type better, an example would be: What is the book written by J. R. R. Tolkien?

The answer contains a list of books which all are written by the same writer on different dates.

1.3.5 Confirmation Questions

Confirmation questions need answers in the form of yes or no. [13]

These questions usually reside on the information of the answerer on the asked question. Usually, the question asker also waits for a detailed response other than the simple answer given by the questioner.

To understand this type of question, we may have an example: Did you do your homework today?

The answer can be yes or no, and maybe an explanation will follow to explain the situation.

1.3.6 Complex Questions

Complex Questions require multiple, different types of information, and giving answers is complicated. [13]

These questions usually have a deeper understanding, and many answers are gathered from multiple and changing sources, and the answer can be short or long depending on the needs.

To understand this kind of question better, an example would be: How can we solve the world's electricity problem?

The answer can be multiple non-dependent answers, such as We can build power plants, use solar energy, research outer space for energy, etc.

1.4 OBJECTIVES

The primary objective of this study is to understand how AI-based Question Answering Systems works, how it is integrated with different fields such as Information Retrieval (IR), Information Extraction (IE), and Natural Language Processing (NLP) works in collaboration to achieve State-of-The-Art results. Also, analyzing the increase in QA Systems performance and accuracy showed that Googles BERT model had a significant impact on QA tasks to achieve higher accuracy. In this, we try to learn BERT Fine Tuning and apply it to a case study to see how QA systems can be helpful to people for tasks.

1.5 ORGANIZATION OF THE THESIS

This thesis contains five chapters. All the necessary information about creating a QA System and Fine-Tuning the BERT Model on a case study is explained with the results.

Chapter 1 is an introduction to the history of QA Systems and the objectives of this thesis.

Chapter 2 introduces AI and Machine Learning and will give a basic understanding of the models used in this thesis.

Chapter 3 is an introduction to NLP and BERT model used in this thesis.

Chapter 4 is an introduction to Datasets and explains how the case study's dataset was prepared.

Chapter 5 explains how a Fine-Tuned BERT model can be applied to a case study and explains the results.

CHAPTER II

AI AND MACHINE LEARNING

2.1 ARTIFICIAL INTELIGENCE

Artificial intelligence (AI) can be summarized as the intelligence demonstrated by machines. The first idea of a clever artificial machine was proposed as Darthmouth Summer Research Project by John McCarty, and Marvin L. Minsky who were considered the founders of the Artificial Intelligence field. [14]

Today, with the current research in AI, we can see many areas in which AI is adapted and used such as Virtual Assitansts like Siri, Visual Image Recognition for identifying photos, Classification of data, Expert Systems, and so on.

The general term of Artificial Intelligence also covers the main fields of Machine Learning and Deep Learning, which use different models for more focused tasks and applications.

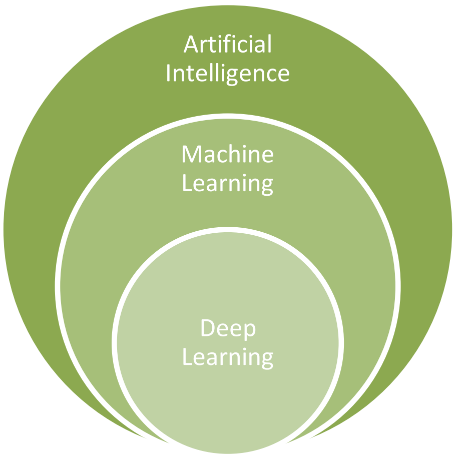


Figure 4: Artificial Intelligence, Machine Learning, and Deep Learning Visualized

2.1.1 Narrow AI

Narrow AI, sometimes referred to as Weak AI, has been created for specific tasks. At the same time, Artificial General Intelligence is designed to match human-level intelligence in terms of its broadness and adaptability.[15]

Because these AI are trained on specific tasks, the applications of these Narrow AI are limited, and today when we look around, we can see many applications that we use today with or without knowing.

Such examples are Search Engines, Advertisements specific to users, Google Translate, Youtube subtitle generation, Self-driving cars as Tesla, Virtual Assistants like Siri. All are trained to implement a specific AI Task.

2.1.2 General AI

As also stated earlier, Artificial General Intelligence is designed to match human-level intelligence in its broadness and adaptability.

This type of AI will be closer to humans because it can learn many different tasks and improve on these to achieve Human success. Since there is no limitation of the task, this type of AI acts more like a child's brain that can learn anything it is trained for.

Examples of these systems are not present, and a massive amount of research needs to be done on this field to create a truly Wide AI, which may take years or decades to accomplish.

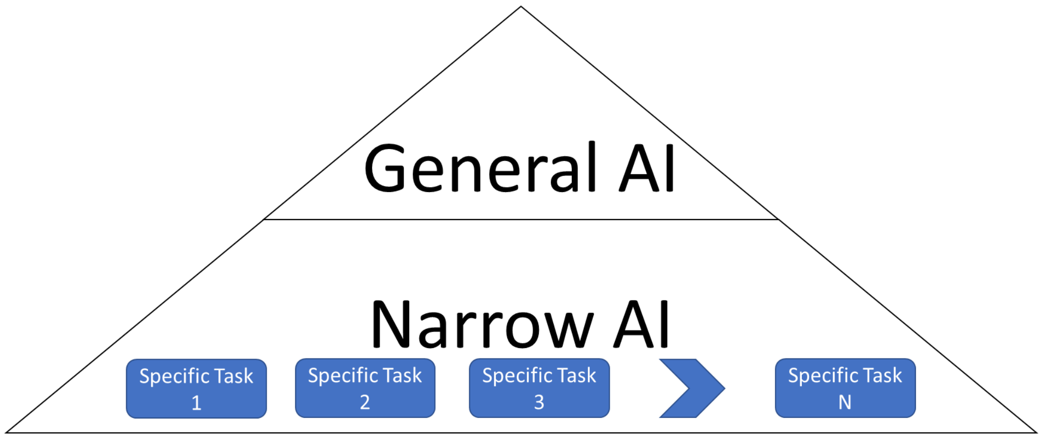


Figure 5: General AI and Narrow AI Visualization

2.2 MACHINE LEARNING

The term Machine Learning was introduced in 1959 by Arthur Samuel, an IBM worker in computer gaming and artificial intelligence. [16]

As a general definition, Machine Learning is the study of computer algorithms to achieve State-Of-The-Art results involving Neural Networks, Data Mining, Generalization, and Statistics usage to analyze this data predict future outcomes.

2.2.1 Supervised Learning

In general, supervised learning is a machine learning task which maps inputs and outputs together. The given inputs and outputs are considered training data for the model to work according to a mapping function, mainly in a vector to get the desired outputs.

There are many algorithms used, such as Support-Vector Machines, Linear Regression, Decision Trees, Neural Networks, etc.

The disadvantage of this model is the amount of time needed to prepare the dataset as inputs and outputs, which takes much effort.

In this work for the case study of Fine-Tuned BERT model, we will be dealing with Supervised Learning datasets, which have the Passages, Questions, Answers, Beginning of the Answer, and Ending of the Answer, which actual Humans prepare.

2.2.2 Unsupervised Learning

On the other hand, Unsupervised Learning consists only of the input data without the outputs. Hence, the model tries to get the patterns by self-analyzing the given inputs and getting the desired output using this pattern.

To gain these patterns, models use to analyze similarities through the input data and group these data.

The obvious advantage of these models is the less effort to prepare the inputs and outputs given to the model.

2.2.3 Reinforcement Learning

Reinforcement Learning is more like a trial and error case, where the model makes several attempts to gain better results at a task. Each more correct route is considered an input to the new tryout to achieve the best possible State-Of-The-Art results.

This model has advantages to improve over time and gets a better understanding using the given inputs.

2.3 MACHINE LEARNING MODELS

In general, neural networks link nodes with an input layer, some hidden middle layer, and an output layer. It is considered one of the milestones in AI Systems that resembles an actual human brain-like structure.

Neural networks are considered to be one of the best-performing machine learning algorithms. They have brought great success in artificial intelligence, such as in the field of computer vision, where their task is image processing and pattern recognition, and, for example, in sound processing and speech recognition. [17]

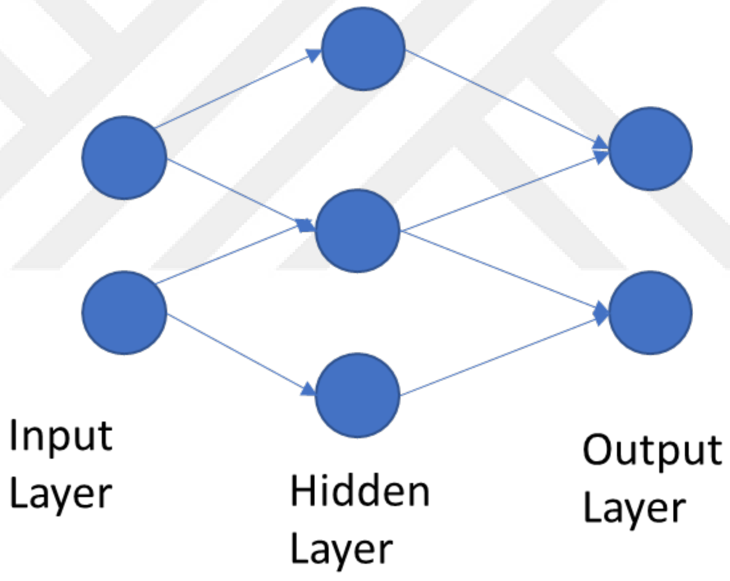


Figure 6: Neural Network Visualization

2.3.1 Convolutional Neural Network

Convolutional neural networks (CNN) represent a specific type of forwarding neural network containing a layer of neurons for the convolution operation. [17]

CNN aims to apply a filter as a layer to decrease the number of outputs below the number of inputs while keeping the number of inputs the same and outputs the same for every case.

The primary usage for CNN is mainly in the visual imagery field, such as Image recognition.

2.3.2 Recurrent Neural Network

Recurrent neural networks (RNN) contain cyclic connections that make them a more powerful tool to model such sequence data than feed-forward neural networks, and RNN's have demonstrated great success in sequence labeling and prediction tasks such as handwriting recognition and language modeling. [18]

Because the input resides on the previous information and an auto-correcting mechanism was added as part of the Neural Network, this was a big step for Natural Language Processing.

2.3.3 Long Short-Term Memory (LSTM)

LSTM is also a type of RNN. Compared to other neural networks, the LSTM network does not consist of interconnected neurons but memory blocks connected in layers. [17]

With this added support of memory block, it quickly became a massive success in NLP fields such as Question Answering.

The memory blocks contain memory cells with self-connections storing the temporal state of the network in addition to special multiplicative units called gates to control the flow of information. [18]

2.3.4 Transformers

RNN and LSTM were considered the State-Of-The-Art approaches until "Attention" based models like transformers came in. It all started with the paper titled "Attention Is All You Need," which became very popular quickly.

Transformer, a model architecture eschewing recurrence and instead relying entirely on an attention mechanism to draw global dependencies between input and output, allows for significantly more parallelization and can reach a new state of the art in translation quality after being trained for as little as twelve hours on eight P100 GPUs. [1]

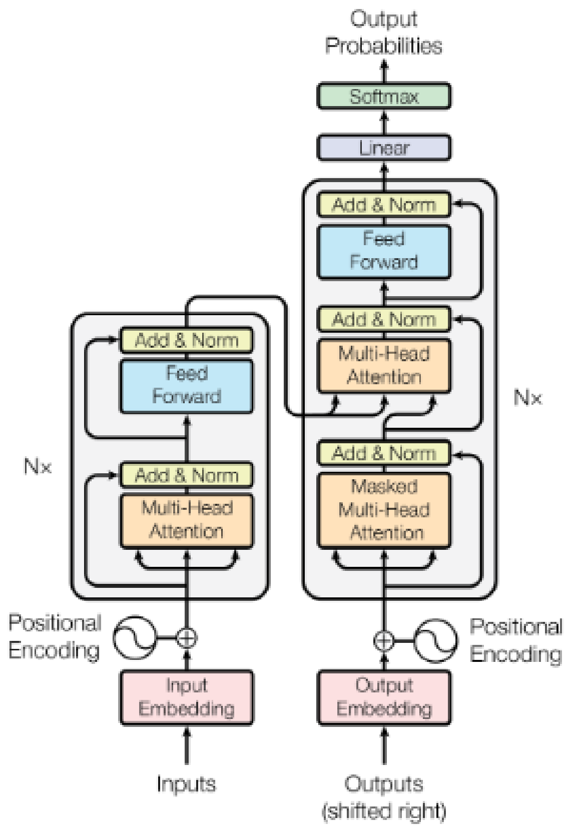


Figure 7 : The Transformer – Model Architecture from [1]

CHAPTER III

NATURAL LANGUAGE PROCESSING

3.1 NLP GENERAL KNOWLEDGE

Natural Language Processing (NLP) is how computer and human interaction is involved around computers trying to understand human language. It is considered to be part of Computer Engineering and Artificial Intelligence areas.

The main aim of NLP is to create computers capable of understanding and processing natural language. It is mainly used in solving problems like speech recognition, question answering, machine translation, and text mining areas, and the list continues to grow every day with new needs.

3.2 BERT

Bidirectional Encoder Representations from Transformers (BERT) is designed to pre-train deep bidirectional representations from the unlabeled text by jointly conditioning both left and proper context in all layers. The pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for many tasks, such as question answering and language inference, without substantial task-specific architecture modifications. [19]

With this explanation from the creators, we can further analyze its structure. BERT is a pre-trained model trained on the entire Wikipedia, which contained 2,500 million words and a Book Corpus of 800 million words.[19] Rumors indicated that this training took more than four days to complete, even on modern high-end computers. By indicating bidirectional, the Bert model captures both sides of the context to the left and the right. Bert also converts words to vectors to achieve their vectorial values. This separates the Bert model from its alternatives, such as GPT, which captures left-to-right, and ELMo, which uses independent left-to-right and right-to-left LSTMS.

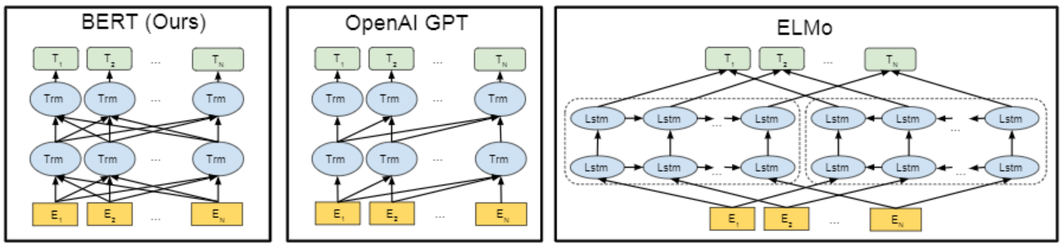


Figure 8 : BERT, GPT and ELMo comparison from [19]

3.2.1 BERT Word Embeddings

Bert base has a vocabulary of over 30.000 words and characters from its pre-trained state.[19] This word consists of whole words, partial words, and single characters. Also, numbers are present in this vocabulary. When BERT gets a word, it divides it into smaller pieces in its vocabulary and appended "##" to the middle ones. An example would be the word jumping, which will be split as "jump" and "##ing" and so on.

3.2.2 BERT Pre Training

BERTS pretraining consists of two unsupervised tasks, namely Masked LM and Next Sentence Prediction.

3.2.2.1 Masked LM (MLM)

In order to train a deep bidirectional representation, we simply mask some percentage of the input tokens at random and then predict those masked tokens.[19]

This means that by masking a word and predicting the masked word, the model improves its prediction mechanism for finding the right masked words.

3.2.2.2 Next Sentence Prediction (NSP)

In general terms, this training aims to predict the following coming sentence and try to make a relationship between these two sentences.

Specifically, when choosing the sentences A and B for each pretraining example, 50% of the time, B is the following actual sentence that follows A (labeled asIsNext), and 50% of the time, it is a random sentence from the corpus (labeled asNotNext). [19]

3.2.3 BERT Architecture

Bert is a transformer model indicating that it uses many encoding layers on an input to get the specific output. If we look at the Pre-trained BERT model, we can see that mainly there are two models stated.

BERT base (L=12, H=768, A=12, Total Parameters=110M) and BERT large (L=24, H=1024, A=16, Total Parameters=340M). [19]

3.2.4 BERT Text Processing

In order to find the exact locations of the words, BERT uses Position Embeddings. To separate sentences from each other, BERT uses Segment Embeddings, and in order to separate words, BERT uses Token Embeddings.

Token Embeddings consists of [CLS], which indicates the beginning of the sentence and is followed by [SEP] for every separate sentence and ends with [SEP] token.

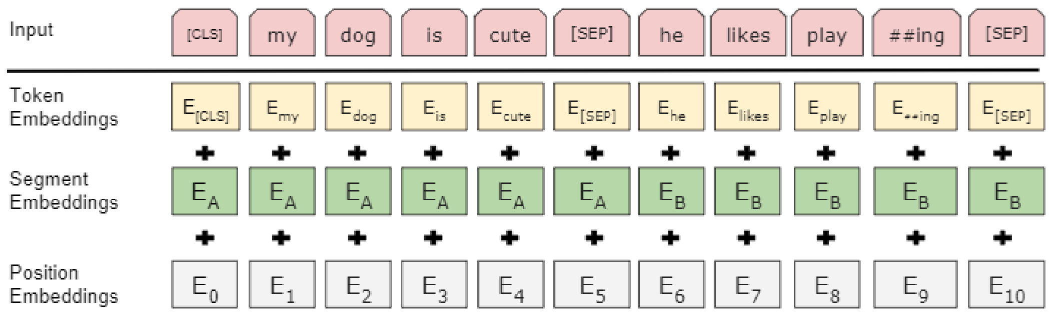

Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

Figure 9 : Token Embedding, Segment Embedding and Position Embedding Visualized from [19]

3.2.5 Fine-Tuning Bert

Bert can be fine-tuned for many tasks according to needs; for this work, we will be focusing more on the question Answering task fine-tuning.

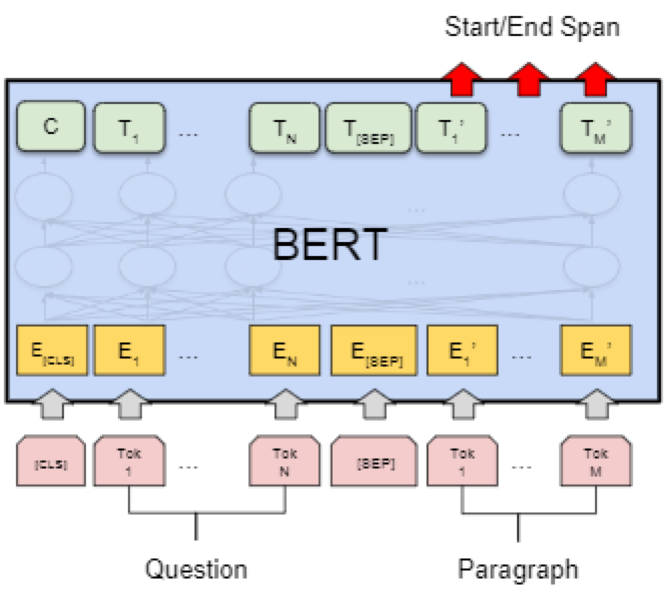


Figure 10 : Question Answering Task Fine-Tuning form [19]

CHAPTER IV

DATASET

4.1 DATASET

A dataset indicates a collection of data in the area of research grouped under labels. Many sites are offering the needed data to researchers, which the main ones are “datasetsearch.research.google.com” and “https://huggingface.co/docs/datasets/.”

4.1.1 SQUAD Dataset

SQUAD dataset is the one that was used to pre-train the BERT model, so understanding this data set will help us understand how BERT reacts to asked questions.

Stanford Question Answering Dataset (SQuAD) is a new reading comprehension dataset consisting of 100,000+ questions posed by crowd workers on a set of Wikipedia articles. The answer to each question is a segment of text from the corresponding reading passage. [2]

The SQUAD dataset can be found at https://rajpurkar.github.io/SQuAD-explorer/, and it can also be explored using the web browser for passages, questions, and answers to these questions.

The primary SQUAT Dataset was later renamed SQUAD 1.1, and another set of questions were added to create the newer version of SQUAD 2.0, which consists of 50.000 more questions that were not answerable from the passage.

Each new State-of-the-Art model has a chance to test itself against these SQUAD 1.1 and 2.0 models, and the higher Exact Match (EM) and F1 scores are listed on the leaderboard.

4.1.2 Analyzing Case Study Dataset

The dataset we will be using is from deepset.ai, Covid-19 Dataset, which is in JSON format and can be downloaded from the link: https://github.com/deepset-ai/COVID-QA/blob/master/data/question-answering/COVID-QA.json

This data has a size of 4.21 MB contains Context, Questions, Answers, and Starting Index of Questions, and its structure can be found below.

```
{
  "data": [{
            "paragraphs": [{
                      "qas": [{
                                "question": "What is the main cause of HIV-1 infection in children?",
                                "id": 262,
                                "answers": [{
                                          "text": "Mother-to-child transmission (MTCT) is the main cause of HIV-1 infection in children worldwide. ",
                                          "answer_start": 370
                                }],
                                "is_impossible": false
                      }],
                      "context": "Functional Genetic Variants in DC-SIGNR Are Associated ..... ",
"document_id": 630
            }]
  }]
}
```

Moreover, it contains a total of 2014 rows of comparable data used for training tasks.

4.1.3 Preparing Case Study Dataset

Since the data is in JSON format, we need to load it using the pythons JSON library, and this gives us 146 columns of data to be processed. Then the data needs to be read by the program, and to create the Pandas Data frame, and we need to divide it according to the text labels as below:

```python
questions = []
answers = []
context=[]
answer_start=[]
answer_end=[]

for i in range(146):
  tempvar = train[[i]]
  temptext = tempvar[i]
  tempanswer = (temptext[0])
  tempresult = tempanswer['paragraphs']
  question=[]
  answer=[]
  tempresult=tempresult[0]
  tempquestion = tempresult['qas']
  for y in range(len(tempquestion)):
    context.append(tempresult['context'])
    tempquestionandanswer = (tempquestion[y])
    if 'question' in tempquestionandanswer.keys():
      questions.append(tempquestionandanswer['question'])
      anslist = tempquestionandanswer['answers']
      anslist = anslist[0]
      answers.append(anslist['text'])
      answer_start.append(anslist['answer_start'])
      answer_end.append(int(anslist['answer_start'])+len(anslist['text']))
```

This gives us a series of data needed for our analysis which we can convert to Pandas Data Frame object.

Since the whole set contains very long Context values and training with those values takes too much time, I limited the values to 200 characters for context and another 200 characters for the answers to these questions. Also, working with 2014 rows required too much ram and GPU power, so I also narrowed the dataset to use 300 rows in total for the Case Study.

```python
context2 = []
answers2 = []

for c, s, e, a in zip(context, answer_start, answer_end, answers):
    context2.append(c[s:s+200])
    answers2.append(a[:200])
data2 = pd.DataFrame(data=list(zip(questions,
answers2,context2,answer_start,answer_end)),
columns=['questions','answers','context','starting_index','ending_index']).head(
300)
```

We now have the Data Frame with shapes 300,5 containing all the needed information, so now we are ready to divide the 300 Rows of Data Frame to perform training operations.

4.1.4 Splitting Train, Dev, and Test

Since we now have our data frame, we can divide our set into three sets named Train, Dev, and Test.

Train : The Train Dataset is used to train the model with the given Question and Answer sets. This will be the most significant part of the Data Frame.

Dev : The Dev Dataset is used to improve the learning of the model, and for each training, we can check against the dev set to see the improvements.

Test : The Test Set is used after the full training is completed to test for predictions of the trained model and see the improvements of training.

In order to split our training sets, we need to use the SKlearns train_test_split model. This model splits the data frame into 2 random parts, such as train and test data. So in order to get the 3 splits we need to divide the test part again. I tried a %80 Train, %10 Dev, and %10 Test parts for this thesis study. To achieve this programmatically, I first divided the model to %80 Train to %20 Dev, then divided that %20 part to two %50 part, each corresponding to %10 percent of the entire data frame. To reproduce the same results, I selected the random state as my birth year. Otherwise, the results will change for every run.

```
from sklearn.model_selection import train_test_split

train_size=0.8

X = data2.drop(columns = ['ending_index']).copy()
y = data2['answers']

X_train, X_rem, y_train, y_rem = train_test_split(X,y, train_size=0.8,
random_state=1983)

test_size = 0.5
X_valid, X_test, y_valid, y_test = train_test_split(X_rem,y_rem, test_size=0.5,
random_state=1983)
```

Since we used 300 rows and %80 Train set gets 240 rows, %10 Dev gets 30 rows and %10 Test gets 30 rows.

```python
i = 0

string_train_data = "["

for index, row in X_train.iterrows():
    string_train_data += "{ \"context\" : \""
    string_train_data += row['context']
    string_train_data += "\", \"qas\" : [ {\"id\": \""
    string_train_data += str("{0:0>5}".format(i))
    string_train_data += "\", \"is_impossible\": false, \"question\" : \""
    string_train_data += row['questions']
    string_train_data += "\", \"answers\" : [{ \"text\" : \""
    string_train_data += row['answers']
    string_train_data += "\", \"answer_start\": "
    string_train_data += str(1)
    string_train_data += "}]}]},"
    i += 1
string_train_data[:-1]
string_train_data += "]"

with open('train_data.json', 'w') as f:
    f.write(string_train_data)
```

The output of this file is a multiline JSON file. In order to fix it, we need to remove the last comma, convert it to a single line and add "\" in front of the middle """ to make it a valid JSON file. We repeat these steps for Dev and Test to get the necessary JSON files.

```python
i = 0

string_eval_data = "["

for index, row in X_valid.iterrows():
    string_eval_data += "{ \"context\" : \""
    string_eval_data += row['context']
    string_eval_data += "\", \"qas\" : [ {\"id\": \""
    string_eval_data += str("{0:0>5}".format(i))
    string_eval_data += "\", \"is_impossible\": false, \"question\" : \""
    string_eval_data += row['questions']
    string_eval_data += "\", \"answers\" : [{ \"text\" : \""
    string_eval_data += row['answers']
    string_eval_data += "\", \"answer_start\": "
    string_eval_data += str(1)
    string_eval_data += "}]}]},"
    i += 1
string_train_data[:-1]
string_eval_data += "]"

with open('eval_data.json', 'w') as f:
    f.write(string_eval_data)
```

We got the Dev set as JSON.

```python
i = 1

string_predict_data = "["

for index, row in X_test.iterrows():
    string_predict_data += "{ \"context\" : \""
    string_predict_data += row['context']
    string_predict_data += "\", \"qas\" : [ { "
    string_predict_data += " \"question\" : \""
    string_predict_data += row['questions']
    string_predict_data += "\", \"id\": \""
    string_predict_data += str("{0:0>5}".format(i))
    string_predict_data += "\"}]},"
    i += 1
string_predict_data += "]"

with open('predict_data.json', 'w') as f:
    f.write(string_predict_data)
```

We get the Test set with this command now that our JSON files are ready and we have all the data needed to Fine-tune our BERT model.

CHAPTER V

QUESTION ANSWERING

5.1 QUESTION ANSWERING WITH FINE TUNED BERT

The end of 2018 was a big year for the BERT model because researchers could work on pre-trained models on certain specific tasks. These tasks include sentence-level tasks such as natural language inference and paraphrasing, which aim to predict the relationships between sentences by analyzing them holistically, and token-level tasks such as named entity recognition and question answering, where models are required to produce fine-grained output at the token level. [19]

5.1.1 Advantages of Fine Tuning

The main advantages of BERT Model can be summarized from mainly three perspectives. The first advantage is the ability to quickly develop the fine-tuned model on a specific task because of the pre-trained data already there. If we tried to build the model from scratch and train it, even on high-end supercomputers, it takes too much time to train from scratch.

The second advantage is the need for fewer data to achieve better results. Because our model is pre-trained any add-on will require a small set of data and will be time saving since preparing the data takes too much time.

The third advantage is that even with little fine-tuning and minor data given, the model can perform better or equivalent results to other models, which were completely developed from scratch.

Also, by fine-tuning BERT on specific domains and tasks, new variant models are developed, which even outperformed the original BERT implementation discussed earlier in this thesis.

5.1.2 Preparing The Environment

Since AI and Machine Learning (ML) development needs high-end processing units like GPU and high ram usages, user PCs or laptops are usually not sufficient. For this reason, google developed its free research platform Colab which can be accessed through the link : https://colab.research.google.com/

Google Colab offers 12 Gigabyte RAM and CPU, GPU or TPU usage according to the needs, so to use this platform, we signed in to Colab with our google account.

Many programming languages can be used for Question Answering tasks, but because of the many libraries presented, we choose Python as our programming language. Also, to run our code live on servers, we used Python on Jupyter Notebooks, which made step-by-step implementation possible.

While running the code implementation, we also imported Tensorflow, Numpy, Transformers, Simple Transformers, PyTorch, Pandas, Json, Logging modules.

5.1.2.1 Huggin Face Transformer Library

The Huggin Face library is a library consisting of State of the art Transformers models such as BERT. It contains support to nearly all Pre-Trained Transformer Models, and since it is a free library, researchers download and work on different pre-trained models on different NLP tasks to improve the performance.

Huggin Face Library can be accessed under link https://huggingface.co/, and it contains Models, Datasets, Resources for different NLP tasks. These tasks include Question Answering, Summarization, Text Classification, Text Generation, Token Classification, Translation, Sentence Similarity, Conversational, and Future Extraction.

5.1.2.2 Simple Transformers Library

The Simple Transformers Library is built on top of Huggin Face Library and added a much easier interface with additional support for working with NLP models much faster and easier. We will also be using this library for our training, evaluation, and testing purposes during this thesis.

5.1.2.3 Loading Dataset

Since we already prepared our Train, Dev, and Test files, here we will only be loading the sets with the help of Pythons Json library.

To load the training Dataset:

```
import JSON
with open(r"train_data.json", encoding='utf-8', mode="r") as read_file:
    train = json.load(read_file)
```

To load the Evaluation Dataset:

```
with open(r"eval_data.json", "r") as read_file:
    test = json.load(read_file)
```

And to load the test Dataset we will be using:

```
with open(r"predict_data.json", "r") as read_file:
predict = json.load(read_file)
```

Now are data sets are loaded for Fine Tuning our BERT model and training.

5.1.2.4 Tokenizing Inputs

BERT Model uses tokenization to convert text values into token id's which can be used later in vectorial calculations. In order to do this BERT model separates the given input into meaningful Word Embedding pieces in its vocabulary of 30,000 words, as stated earlier. It adds "##" sign to the words separated except the first piece or the word. Below is an example of a tokenized text for Bert. In order to use BERT tokenizer we have to import the BertTokenizer form the transformers library, as shown below.

```
from transformers import BertTokenizer

tokenizer = BertTokenizer.from_pretrained('bert-large-uncased-whole-word-
masking-finetuned-squad')
question = "Who is the chairman of Cankaya University?"
answer_text = "Çankaya University (Turkish: Çankaya Üniversitesi) is a private
university in Ankara, Turkey. It was established on July 9, 1997 by the Sıtkı
Alp Education Foundation.[1] The university began its teaching in the Fall 1997
semester. Sıtkı Alp is the chairman of the board of trustees. English language
is predominant medium of teaching, learning and research at the Çankaya
University."
print(tokenizer.convert_ids_to_tokens(tokenizer.encode(question)))
print(tokenizer.encode(question))
```

The output of running this code gives us this output.

```
['[CLS]', 'who', 'is', 'the', 'chairman', 'of', 'can', '##kaya', 'university',
'?', '[SEP]']
[101, 2040, 2003, 1996, 3472, 1997, 2064, 20718, 2118, 1029, 102]
```

This example shows how tokenization works for BERT model and gives us an understanding of the model.

5.1.2.5 Formatting Special Characters

While tokenizing sentences, if you observe, there are unique values like [101] and [102] which stand out from the rest.

The first token of every sequence is always a unique classification token ([CLS] or 101). [19]

Sentence pairs are packed together into a single sequence. We differentiate the sentences in two ways. First, we separate them with a special token ([SEP] or [102]). Second, we add a learned embed-ding to every token indicating whether it belongs to sentence A or B. [19]

Also, if we check it in multiple text cases, we can see that all sentences begin with [CLS], then Sentence A, the [SEP], and the secondSentence B, which also ends with a [SEP] unique character.

Ther is also a Padding token ([PAD] or [0]) used to make all entries the same fixed size.

```
question = "Example question sentence A?"
answer_text = "Example answer text which is long."

input_ids = tokenizer.encode(question, answer_text)

print(tokenizer.convert_ids_to_tokens(input_ids))
print(input_ids)
```

This Gives us the output as below:

```
['[CLS]', 'example', 'question', 'sentence', 'a', '?', '[SEP]', 'example',
'answer', 'text', 'which', 'is', 'long', '.', '[SEP]']
[101, 2742, 3160, 6251, 1037, 1029, 102, 2742, 3437, 3793, 2029, 2003, 2146,
1012, 102]
```

5.1.2.6 Limitations Explained

BERT has a limitation of 512 tokens as an input, so in order to do the training for QA task, the total number of Question plus Context, when divided into Bert tokens should not pass this 512 token limit. They are sampled such that the combined length is ≤512 tokens. [19]

Also, BERT expects all the given data to have the same amount of tokens in trainings, we use Padding to add empty tokens to match the size of the inputs to be the same for every input.

5.1.2.7 Padding

Since BERT has a limitaion that every input Question and Answer pair should have the same fixed size, we need to add padding for the short sentences and truncate the long ones so every entry should have the same amount of tokens, an example of how padding is made can be found below.

```
from keras.preprocessing.sequence import pad_sequences

MAX_LEN = 20
input_ids2 = []
input_ids2.append(input_ids)
input_ids2 = pad_sequences(input_ids2, maxlen=MAX_LEN, dtype="long",
                           value=0, truncating="post", padding="post")
print(tokenizer.convert_ids_to_tokens(input_ids2[0]))
print(input_ids2[0])
```

The output of this code is :

```
['[CLS]', 'example', 'question', 'sentence', 'a', '?', '[SEP]', 'example',
'answer', 'text', 'which', 'is', 'long', '.', '[SEP]', '[PAD]', '[PAD]',
'[PAD]', '[PAD]', '[PAD]']
[ 101 2742 3160 6251 1037 1029  102 2742 3437 3793 2029 2003 2146 1012
  102    0    0    0    0    0]
```

5.1.2.8 Attention Masks

Attention masks are masks used to separate the real token values from the padded values. For every token, BERT uses this Attention Mask in an AND gate respective, so if the Attentiton Mask value is zero (0) then it does not have a weight, if it is 1 then it has a weight. A sample of how the Attention mask works can be found below.

```
attention_mask = []

for val in input_ids2:
    temp_att_mask = [int(token_id > 0) for token_id in val]
    attention_mask.append(temp_att_mask)

    print(tokenizer.convert_ids_to_tokens(input_ids2[0]))
    print(attention_mask[0])
```

The output of this code is :

```
['[CLS]', 'example', 'question', 'sentence', 'a', '?', '[SEP]', 'example',
'answer', 'text', 'which', 'is', 'long', '.', '[SEP]', '[PAD]', '[PAD]',
'[PAD]', '[PAD]', '[PAD]']
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0]
```

5.1.2.9 Training Model

To Sum up all until now, we have created our Train, Dev, and Test JSON files, and showed the internal workings of how BERT prepares the given input for the training purposes by combining Question and Context Pasages together, Tokenizing them to its Word Embeddings, Adding Special [CLS] and [SEP] tokens, then adding

Padding [PAD] or truncating them to required length and to create the needed Attention Masks to identify Padding tokens.

So basically, we would use the pre-trained BERT Model and add our Case Study Covid 19 Dataset to improve its question-answering scores.

We will be using the Simple Transformers Library for this task, which was introduced before in this thesis.

First, we install the Simple Transformers Library and load the Train, Dev, Test JSON Files:

```
!pip install simpletransformers

import json
with open(r"train_data.json", encoding='utf-8', mode="r") as read_file:
    train = json.load(read_file)

with open(r"eval_data.json", "r") as read_file:
    test = json.load(read_file)

with open(r"predict_data.json", "r") as read_file:
    predict = json.load(read_file)
```

Next we will be loading BERT model and its pretrained "bert-base-cased" model and make required configurations:

```
model_type="bert"
model_name= "bert-base-cased"
model_args = QuestionAnsweringArgs()
model_args.train_batch_size = 1
model_args.evaluate_during_training = True
model_args.n_best_size=3
model_args.num_train_epochs=5
```

Colab platform has a maximum available memory of 12 Gigabyte and if we select the batch size bigger than 1 then PyTorch will give an error indicating insufficient GPU Ram memory, so we should keep the batch size value smaller as PyTorch actually reserves the whole training operations at the beginning of the training process.

And for the detailed configurations, we can use the Training Arguments parameter. The values I used can be seen in the below code:

```
train_args = {
    "reprocess_input_data": True,
    "overwrite_output_dir": True,
    "use_cached_eval_features": True,
    "output_dir": f"outputs/{model_type}",
    "best_model_dir": f"outputs/{model_type}/best_model",
    "evaluate_during_training": True,
    "max_seq_length": 128,
    "num_train_epochs": 5,
    "evaluate_during_training_steps": 1000,
    "wandb_project": "Question Answer Application",
    "wandb_kwargs": {"name": model_name},
    "save_model_every_epoch": False,
    "save_eval_checkpoints": False,
    "n_best_size":3,
    "train_batch_size": 2,
    "eval_batch_size": 1,
}
```

Next we create our model,

```
model = QuestionAnsweringModel(
    model_type,model_name, args=train_args
)
```

And Start the training

```
model.train_model(train, eval_data=test)
```

The training process takes 3:39 minutes on a Tesla K80 GPU provided by Google Colab platform. The result of this training will be discussed in detail at the Results part of this thesis.

Also, to understand how the model predicts where the answer starts and ends, we can look at the figures below.

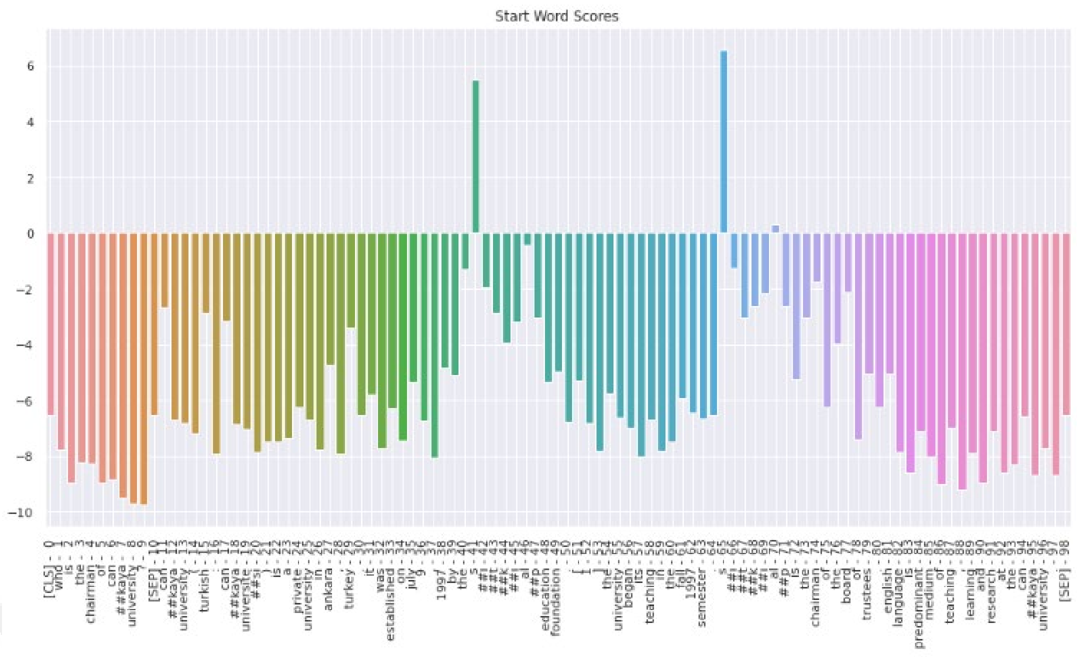


Figure 11 : Showing The Starting Values For Tokens

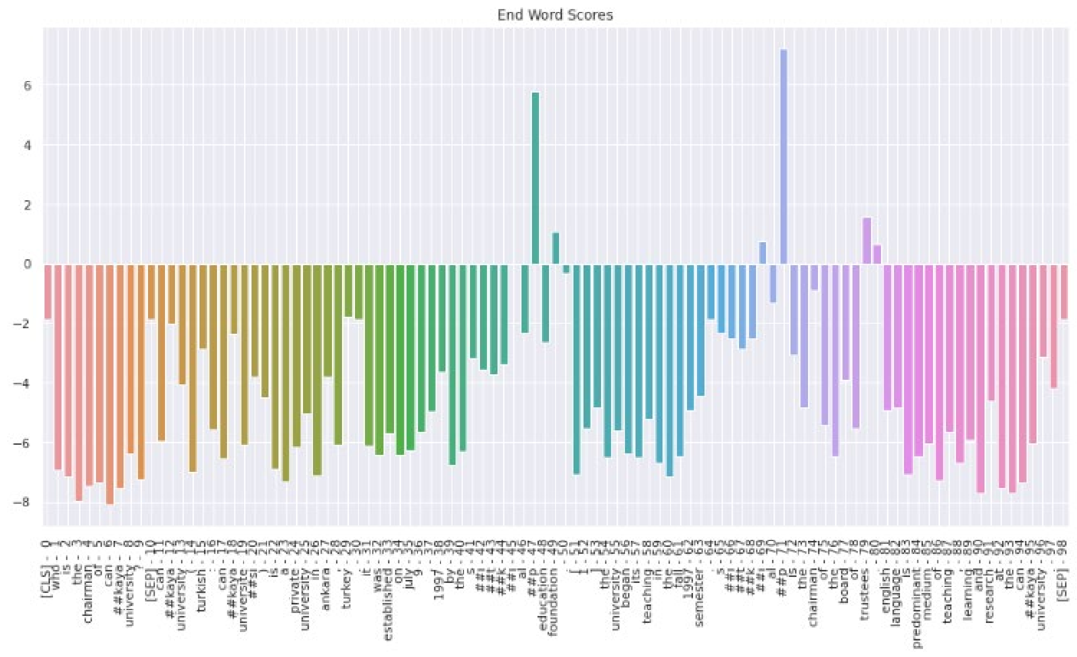


Figure 12 : Showing The Ending Values For Tokens

5.1.2.10 Validating Model

The Validation process is the process of checking how well the model performs during or after the training. Since we configured the model to evaluate during training, after each Epoch of training, the model will be Validated to see the newly trained model's performance. This process repeats for all 5 epochs which we configured.

In order to evaluate the model, we can use the below command:

```
result, texts = model.eval_model(test)
```

5.1.2.11 Testing Model on Test Set

After the model completes its training and gets evaluated, we need to check its performance in the Test JSON file.

In order to test the model we use the below command:

```
answers, probabilities = model.predict(predict)
```

5.1.2.12 Results

The results of this Fine Tuned Bert training for the Covid-19 Case Study showed many of the questions were answered as seen at the below table.

Table 1 : Number Of Correct, Similar And Incorrect Questions

QA Labels	No. Of Questions Epoch 1	No. Of Questions Epoch 2	No. Of Questions Epoch 3	No. Of Questions Epoch 4	No. Of Questions Epoch 5
Correct	16	14	16	16	17
Similar	14	16	14	14	13
Incorrect	0	0	0	0	0

During the training we can see that the correct questions mostly have a uprising graph for correct answers as seen at below figure.

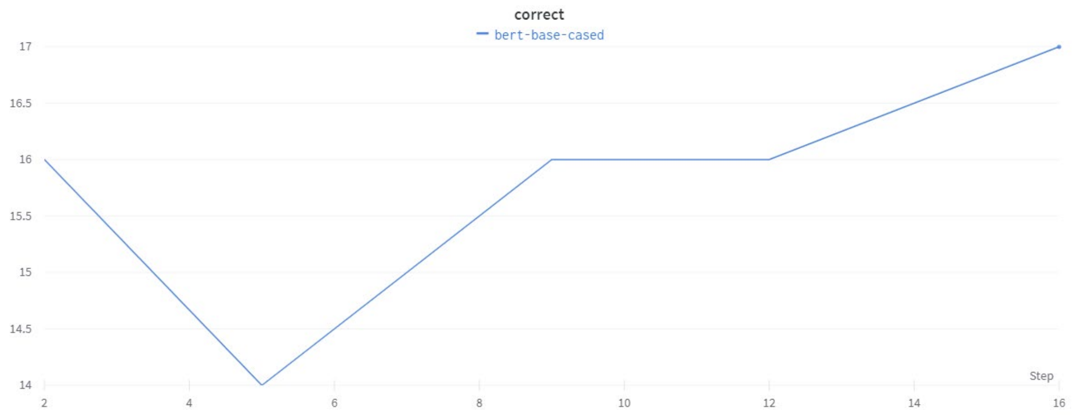


Figure 13 : Number Of Correct Answers Change During Training

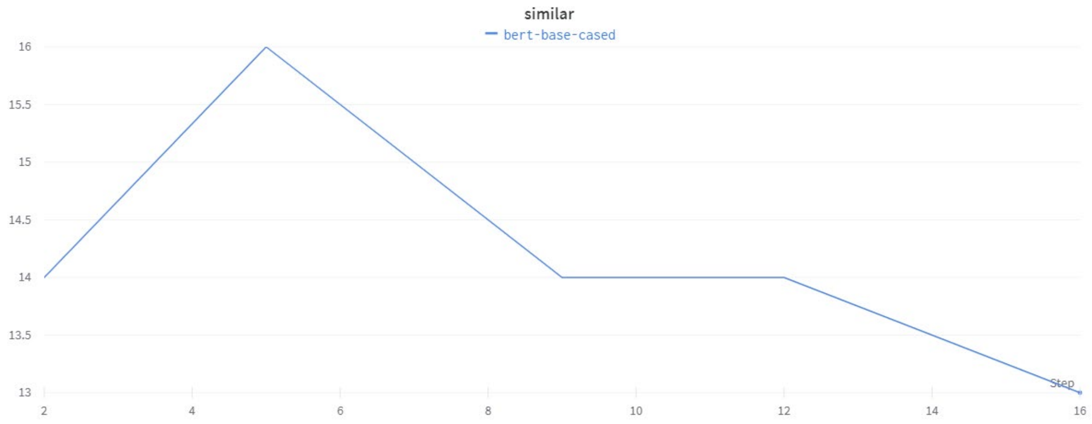


Figure 14 : Number Of Similar Answers Change During Training

This graph shows that at fifth step the model gets 16 similar answers.

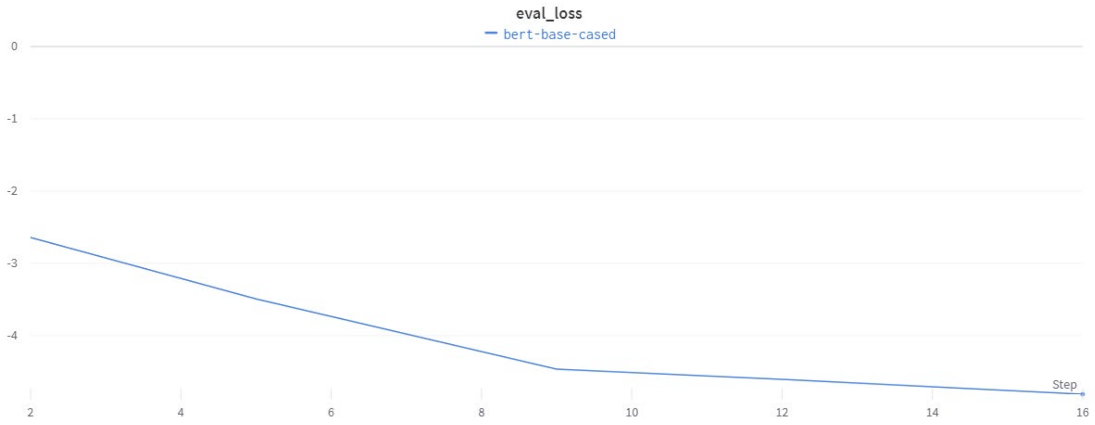


Figure 15 : Evaluation Loss Changes During Training

The evaluation loss shows how good the evaluation was during training. For every correct answer, the evaluation loss is given zero, and when the answer is wrong, it is given a more significant loss, so the perfect model should have a declining graph with a zero at the end of training.

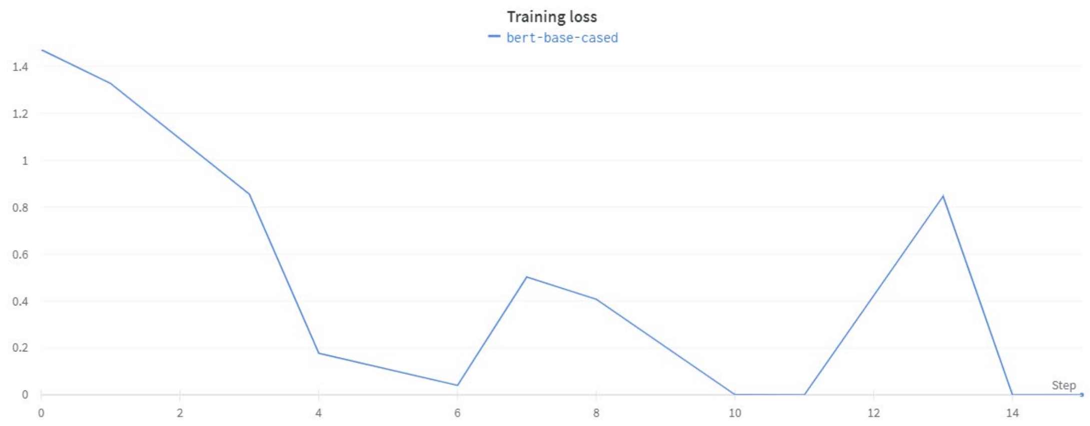


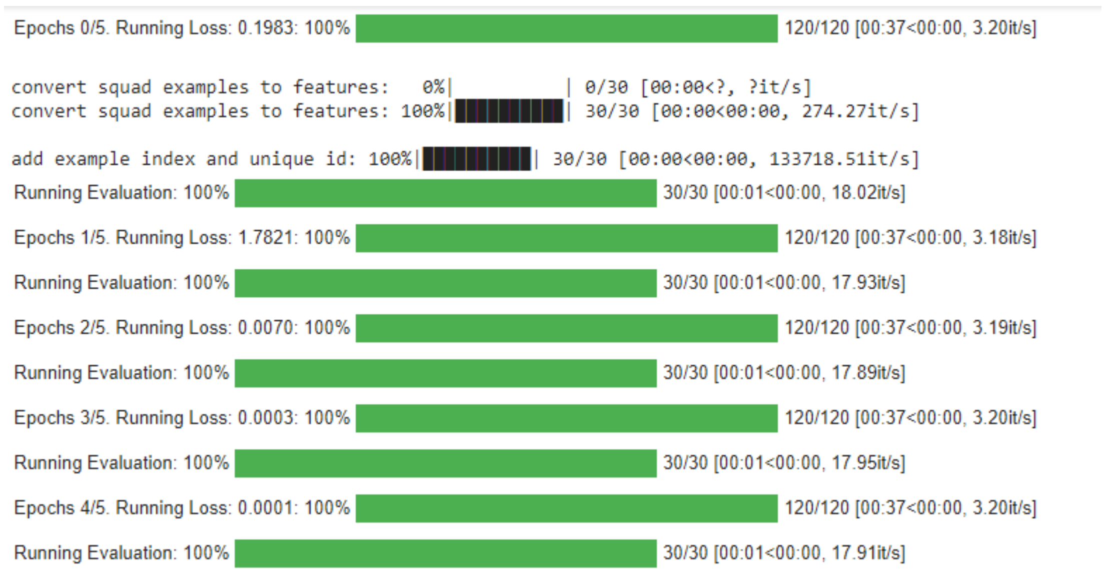
Figure 16 : Training Loss Changes During Training

This figure shows a declining graph from the beginning of the training, and some rising then falling values.

Also the training and evaluation losses can be seen at the below table.

Table 2 : Train Loss And Eval Loss Table During Training

Epochs No	Train Loss	Eval Loss
Epoch 1	0.19826316833496094	-2.641861979166667
Epoch 2	1.7821340560913086	-3.4935546875
Epoch 3	0.0070018768310546875	-4.463736979166667
Epoch 4	0.00027552247047424316	-4.605794270833333
Epoch 5	0.00014537572860717773	-4.810416666666667



```
Epochs 0/5. Running Loss: 0.1983: 100%                    120/120 [00:37<00:00, 3.20it/s]

convert squad examples to features:   0%|          | 0/30 [00:00<?, ?it/s]
convert squad examples to features: 100%|██████████| 30/30 [00:00<00:00, 274.27it/s]

add example index and unique id: 100%|██████████| 30/30 [00:00<00:00, 133718.51it/s]
Running Evaluation: 100%                    30/30 [00:01<00:00, 18.02it/s]
Epochs 1/5. Running Loss: 1.7821: 100%                    120/120 [00:37<00:00, 3.18it/s]
Running Evaluation: 100%                    30/30 [00:01<00:00, 17.93it/s]
Epochs 2/5. Running Loss: 0.0070: 100%                    120/120 [00:37<00:00, 3.19it/s]
Running Evaluation: 100%                    30/30 [00:01<00:00, 17.89it/s]
Epochs 3/5. Running Loss: 0.0003: 100%                    120/120 [00:37<00:00, 3.20it/s]
Running Evaluation: 100%                    30/30 [00:01<00:00, 17.95it/s]
Epochs 4/5. Running Loss: 0.0001: 100%                    120/120 [00:37<00:00, 3.20it/s]
Running Evaluation: 100%                    30/30 [00:01<00:00, 17.91it/s]
```

Figure 17 : The Training Process Output

Below is a comparison with the model without training. The results are very low as expected since our model does not know the jargon for Covid 19 and does not have the relationship among covid related words. The difference between the Covid 19 Dataset trained BERT and untrained base BERT is big.

Table 3 : Case Study Evaluation Results Without Training

	Correct	Eval Loss	Incorrect	Similar
Answers Through Model	0	0.017513211568196616	19	11

Since the training, the model tries to find the best possible model. In order to get the best results, the model needs to be trained for more epochs and find the best model possible with maximum exact matches.

The exact match scores for this model can be calculated by dividing the number of exact matches by the number of total cases.

Exact Match = 17 / 30 = 0,56

The rest of the answers also match mostly, but they are not exact, and most of the researchers started using the exact match scores, which gives more accurate data.

When compared with other works in the Case Study of Covid-19 QA System we can see some better or worse Exact Match scores. Below we can see three examples with their respectful Exact Match scores next to them.

Table 4 : Comparison With Other Works On Case Study Covid-19 QA Systems

Paper Titles For Three Comparative Work	Exact Match (EM) Score
COBERT: COVID-19 Question Answering System Using BERT (23 Jun 2021)[20]	%80,6
Transformer-Based Models for Question Answering on COVID19 (16 Jan 2021)[21]	%13.04
Developing Answers to Scientific Questions with BERT (2020)[22]	%35.89

CONCLUSION

This thesis aimed to make a better Question Answering system that can help people in the area of question answering by fine-tuning a pre-trained model to perform better results. Based on the literature research and latest state-of-the-art models, it was hard to understand the benefits of pre-trained models with their limitations and how to overcome them. Since the introduction of BERT and its variants, the Question Answering tasks have had a significant improvement in every way, and that is why I wanted to use this model for my QA tasks.

The most significant limitations I had during this work were the limitations of Computer hardware needed for the training to perform. Even using Colab Platform's 12 Gigabayt RAM, I had to limit the batch size to a minimum for the model to work. Also, another limitation was BERT's 512 token size limit which forced me to truncate more extended context to fit in these limitations.

Working on Fine Tuned BERT model showed that the pre-trained BERT model performs well on general text with its general Vocabulary, but when given a specific domain like in the Covid-19 Case Study with medical Jargon, big performance drops can be seen.

Fine Tuning Bert for the Covid-19 Dataset and evaluating on the same Dev set before and after training showed a huge improvement in accuracy of Exact Matches even with as little as 5 Epochs.

Working with big datasets, such as the Case Study dataset of 2014 rows took too much Time, RAM and GPU power and is actually not feasible, so we had to make our dataset smaller such as 300 rows, to speed up the training.

Limitations such as BERT's 512 token limit and configuring batch size to higher values such as 16 or 32 used too much RAM, which crashed the system, so researchers had to find workarounds, as we had to drop train batch size to 2 so we could avoid crashing the system.

Modern QA Systems are highly accurate and can be fine-tuned according to needs for the benefits of helping people in many possible areas.

My work on fine-tuned BERT model showed that the pre-trained BERT model performs well on general text with its general Jargon, but when given a specific domain like in the Covid-19 Case Study with medical Jargon the performance drops can be seen. To overcome this problem, we need to train the Fine Tuned BERT on this specific task, so as expected, the results improved after the training with 5 epochs. Still, to get much better results and the best performing model, more epochs training are needed. Since the BERT model trains with different weights and calculates them in parallel with a random order, the results vary every time the training step is run. However, these different results enable the model to achieve the best model possible.

For the future of QA Systems, we will probably see new data formats which will be used as input data, such as Table, Figure, Images, which are currently not possible with recent models. We will be seeing new ideas and better models such as the T5, which eliminates the need for Vectoral complex calculations with the basic Text to Text Transformers ideas. Also, we will be expecting an improvement in Reasoning and Synthesis for a better understanding of the context and answers for the given context and question pairs. Retrieving the perfect answer for today's QA Systems and models is not possible, but in the future will be seeing improved Retrievers, which will always retrieve the %100 Exact Match with perfect scores. Getting rid of the context and just asking a question to get an answer will be possible in the future with enough training and improved QA Systems. With all the development in this specific QA task, we will be seeing QA Systems as parts of the General AI. We talked about eralier, machines that could learn and answer questions perfectly just like a human would be, maybe better in the future.

As machine learning models improve on specific tasks, the gap of a vast AI machine is closing, but we still have more research to be done. This QA task will become perfect in the future, and the “Imitation Game” will become a reality. Also, virtual assistants like Siri will also improve and help people with their daily lives.

All the codes mentioned for this thesis work will be uploaded to :

https://github.com/ErmanOzg/Ermans-Fine-Tuned-BERT-QA-Studies

REFERENCES

[1] A. Vaswani *et al.*(Jun. 2017), “Attention Is All You Need,”, 31st Conference on Neural Information Processing System [Online].
Available: http://arxiv.org/abs/1706.03762

[2] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang (2016), “SQuAD: 100,000+ Questions for Machine Comprehension of Text.”, Conference on Empirical Methods in Natural Language Processing, [Online].
Available: https://stanford-qa.com,

[3] Z. Zhang *et al.*(Sep. 2019), “Semantics-aware BERT for Language Understanding,”, [Online]. Available: http://arxiv.org/abs/1909.02209

[4] Y. Liu *et al.*(Jul. 2019), “RoBERTa: A Robustly Optimized BERT Pretraining Approach,”, [Online]. Available: http://arxiv.org/abs/1907.11692

[5] M. Joshi, D. Chen, Y. Liu, D. S. Weld, L. Zettlemoyer, and O. Levy (Jul. 2019), “SpanBERT: Improving Pre-training by Representing and Predicting Spans,”, [Online]. Available: http://arxiv.org/abs/1907.10529

[6] V. Sanh, L. Debut, J. Chaumond, and T. Wolf (Oct. 2019), “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter,”, [Online]. Available: http://arxiv.org/abs/1910.01108

[7] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut (Sep. 2019), “ALBERT: A Lite BERT for Self-supervised Learning of Language Representations,”, [Online]. Available: http://arxiv.org/abs/1909.11942

[8] C. Raffel *et al.*(Oct. 2019), “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer,”, [Online].
Available: http://arxiv.org/abs/1910.10683

[9] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. v. Le (Jun. 2019), “XLNet: Generalized Autoregressive Pretraining for Language Understanding, ”, [Online]. Available: http://arxiv.org/abs/1906.08237

[10] D. Yogish , T. N. Manjunath, and R. S. Hegadi (2017), *Survey on Trends and Methods of an Intelligent Answering System*.

[11] B. F. Green JR (1961), “BASEBALL: AN AUTOMATIC QUESTION-ANSWERER, ” *Western Joint IRE-AIEE-ACM Computer Conference*, pp. 219–224.

[12] A. Chandra, O. Reddy, and K. Madhavi (2017), “A Survey on Types of Question Answering System,” vol. 19, no. 6, pp. 19–23. doi: 10.9790/0661-1906041923.

[13] S. Singh, N. Das, R. Michael, and P. Tanwar (2016), “The Question Answering System Using NLP and AI,”. [Online]. Available: http://www.ijser.org

[14] J. McCarthy, M. L. Minsky, N. Rochester, and C. E. Shannon (2006), “A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence,”.

[15] D. Schlegel and Y. Uenal (2021), “A Perceived Risk Perspective on Narrow Artificial Intelligence A Perceived Risk Perspective on Narrow Artificial Intelligence,” PACIS, 2021. [Online]. Available: https://aisel.aisnet.org/pacis

[16] A. L. Samuel (1959), “Some Studies in Machine Learning Using the Game of Checkers,”.

[17] V. Maslej-Krešňáková, M. Sarnovský, P. Butka, and K. Machová (Dec. 2020), “Comparison of deep learning models and various text pre-processing techniques for the toxic comments classification,” *Applied Sciences (Switzerland)*, vol. 10, no. 23, pp. 1–26, doi: 10.3390/app10238631.

[18] H. H. Sak, A. Senior, and B. Google, “Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling.”

[19] J. Devlin, M.-W. Chang, K. Lee, K. T. Google, and A. I. Language (May 2019), “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.” [Online]. Available: https://github.com/tensorflow/tensor2tensor

[20] J. A. Alzubi, R. Jain, A. Singh, P. Parwekar, and M. Gupta (Jun. 2021), “COBERT: COVID-19 Question Answering System Using BERT,” *Arabian Journal for Science and Engineering*, doi: 10.1007/s13369-021-05810-5.

[21] H. Ngai, Y. Park, J. Chen, and M. Parsapoor (Jan. 2021), “Transformer-Based Models for Question Answering on COVID19,”, [Online]. Available: http://arxiv.org/abs/2101.11432

[22] S. He and Z. Bakhtiari (2020), “Developing Answers to Scientific Questions with BERT Category: Natural Language Processing.” [Online]. Available: https://www.kaggle.com/allen-institute-for-ai/CORD-19-research-challenge