A TURKISH LANGUAGE FACTOID QUESTION ANSWERING SYSTEM BASED ON DEEP LEARNING NEURAL NETWORKS

by
Cavide Balkı Gemirter

Submitted to Graduate School of Natural and Applied Sciences
in Partial Fulfillment of the Requirements
for the Degree of Master of Science in
Computer Engineering

Yeditepe University
2020

A TURKISH LANGUAGE FACTOID QUESTION ANSWERING SYSTEM BASED ON DEEP LEARNING NEURAL NETWORKS

APPROVED BY:

Asst. Prof. Dr. Dionysis Goularas ……
(Thesis Supervisor)
(Yeditepe University)

Prof. Dr. Emin Erkan Korkmaz ……
(Yeditepe University)

Assoc. Prof. Dr. Ahmet Cüneyd Tantuğ ……
(Istanbul Technical University)

DATE OF APPROVAL: …. /…. /2020

ACKNOWLEDGEMENTS

I would like to express my special thanks to my supervisor, Asst. Prof. Dr. Dionysis Goularas, for encouraging my research and for his guidance through each stage of the process. I have been extremely lucky to have a supervisor who cared so much about my work, and who responded to my questions and queries so promptly.

I would also like to thank my committee members, Prof. Dr. Emin Erkan Korkmaz, and Assoc. Prof. Dr. Ahmet Cüneyd Tantuğ for showing keen interest in the subject matter and accepting to read and review this thesis.

I wish to acknowledge the help provided by Prof. Dr. Kemal Oflazer, for sharing his corpus.

I would like to thank Assoc. Prof. Dr. Ahmet Cüneyd Tantuğ for his support during the study.

I am also grateful to Assoc. Prof. Dr. Ali Gökhan Yavuz for the courage he gave me to start the master program.

A very special thank you to my leader, Mehmet Ali Cer, for his encouragement and support in all of my studies.

My sincere thanks go to my team for their help during the generation of the data used in the study.

Finally, I especially would like to thank my family, Zübeyde and Abdulkadir Gemirter. This research and many other things in my life would not have been possible without your support and love.

ABSTRACT

A TURKISH LANGUAGE FACTOID QUESTION ANSWERING SYSTEM BASED ON DEEP LEARNING NEURAL NETWORKS

In the Question Answering (QA) subfield of the Natural Language Processing (NLP) domain, despite the significant progress in frequently used languages such as English and Chinese, there is still a gap for uncommon languages such as Turkish due to inadequate training data. In Turkish many of the difficulties arise from being an agglutinative language and having a rich but complex morphology, such as a comprehensive set of possible suffix tags and diversity of constituent orders in inverted sentences. Consequently, creating a successful Machine Reading for Question Answering (MRQA) system for Turkish has not been possible yet. In this study, we aim to propose an MRQA system for Turkish in the banking domain. To the question, the system generates the best answer, which is the most correct and the shortest span in a given text. With the BERT deep learning technique, we trained a language model for Turkish using massive corpus collections followed by a fine-tuning process for the MRQA task using large QA datasets. To enhance the MRQA skills of the systems, we also translated some open-domain QA datasets from English to Turkish. At the end of the experiments, it is seen that the system's accuracy is higher than other QA solutions for Turkish. Moreover, the proposed method is not specific to Turkish and applicable for numerous NLP tasks of other limited languages.

ÖZET

DERİN ÖĞRENME SİNİR AĞLARINI KULLANARAK TÜRKÇE'DE GERÇEKSİ SORU CEVAPLAYAN BİR SİSTEM

Doğal Dil İşleme (NLP) alanı Soru Yanıtlama (QA) konusunda, İngilizce ve Çince gibi çok kullanılan dillerde kaydedilen önemli ilerlemelere rağmen, yetersiz eğitim verileri nedeniyle Türkçe gibi daha nadir kullanılan diller için hala boşluklar bulunmaktadır. Türkçe'de zorlukların birçoğu, sondan ekli bir dil olması, olası ek kombinasyonlarının fazlalığı ve devrik cümlelerdeki öğe dizimlerindeki çeşitliliği gibi, zengin ama karmaşık bir morfolojiye sahip olmasından kaynaklanmaktadır. Sonuç olarak, Türkçe için henüz başarılı bir Soru Cevaplama için Makine Okuması (MRQA) sistemi geliştirmek mümkün olmamıştır. Bu çalışmada, bankacılık alanında kullanılabilecek bir Türkçe MRQA sistemi önermeyi hedefliyoruz. Sistem, sorulan soruya, verilen metindeki en doğru ve en kısa cevabı üretmektedir. BERT derin öğrenme tekniğini uygulayarak, kapsamlı korpus koleksiyonları kullanıp Türkçe için bir dil modeli eğittik, ardından büyük QA veri kümeleri kullanarak MRQA görevi özelinde iyileştirmeler yaptık. Sistemin MRQA becerisini arttırmak için bazı genel içerikli QA veri setlerini de İngilizce'den Türkçe'ye çevirip eğitime dahil ettik. Deneylerin sonunda, Türkçe için, eğittimiz sistemin doğruluğunun diğer QA çözümlerinden daha yüksek olduğu görülmektedir. Ayrıca, önerilen yöntem sadece Türkçe'ye özgü değildir ve diğer yaygın olmayan dillerin çeşitli NLP görevlerine de uyarlanabilir durumdadır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS/ABBREVIATIONS

AI	Artificial intelligence
ARCD	Arabic reading comprehension dataset
BERT	Bidirectional encoder representations from transformers
CWE	Contextualized word embeddings
DL	Deep learning
EM	Exact match
FFN	Feed-Forward network
IR	Information retrieval
LM	Language model
LSTM	Long short-term memory
MLM	Masked language model
MRC	Machine reading comprehension
MRQA	Machine reading for question answering
NE	Named entity
NER	Named entity recognition
NLI	Natural language inference
NLP	Natural language processing
NN	Neural network
NSP	Next sentence prediction
OOV	Out of vocabulary
QA	Question answering
POS	Part of speech
RNN	Recurrent neural network
SQuAD	Stanford question answering dataset
TF-IDF	Term frequency-inverse document frequency
TPU	Tensor processing unit
WSD	Word sence disambiguation

1. INTRODUCTION

With the increasing use of Artificial Intelligence (AI) over recent years, AI algorithms have yielded better results than the old traditional solutions in varied domains. Due to the growing training data that can be used in studies and the advancement of artificial intelligence-specific hardware technology, such as the discovery of Tensor Processing Units (TPUs) that work in parallel, advanced neural networks have become achievable. Especially in the Natural Language Processing (NLP) domain, deep learning approaches have reaped much success than the statistical methods used before artificial intelligence. The main reasons for the success of the modern deep learning approaches are; the language models and contextualized word representations. Using extensive corpus data, language models are build, which summarizes the relationships of the words and the main rules of the language. These language models support polysemy and create different word embeddings for homophones considering their context. These advanced deep learning models can only be experienced in commonly used languages due to the difficulties of finding training data.

Especially with the increase in the number of electronic documents and the skills of the search engines, nowadays, access to information is effortless, obtaining the set of documents containing the answers to the questions is simple. Instead of stating all documents, one of the research topics that natural language processing and deep learning researchers have been actively working on in recent years is to scan the documents for finding the best-fit answer.

Machine Reading for Question Answering (MRQA), which scans several documents to find the best span for answering the question, is a vital task for applications such as question answering (QA) systems and search engines. An outstanding MRQA system can find answers to different types of questions from documents in a wide range of domains, rather than particular question types or a closed-domain.

In the MRQA field, despite the significant progress in frequent languages such as English and Chinese, there is still a gap for rare languages such as Turkish as a result of inadequate training data. Furthermore, in Turkish many of the difficulties arise from being an agglutinative language and having a rich but complex morphology, such as a comprehensive set of possible suffix tags and diversity of constituent orders in inverted sentences. Hence,

creating an accomplished open-domain MRQA system for Turkish has not been possible yet.

In Turkish, in order to find answers to questions in a given text, researchers figured out various studies, most of which are rule-based approaches. After identifying the question type like when who or what, in the text, predefined patterns of the classified question type look through the phases, for obtaining the candidate answers. The answers to "when" questions most probably include a time or date expression; therefore, phrases containing years, months, weeks, and days are a successor to be the answer. In a similar way, the answers to "who" questions most probably include a Named Entity (NE) that denotes a person, place, or organization. To increase the success of these average systems, analysts should find out diverse patterns and append to the system, which after while damaging the operating speed of the system. The proposed Turkish MRQA solutions mostly support only specific question types (where, when, who, what, and why), while free-form questions are still unsolved problems.

In recent years, by the increasing number of data sets available for Machine Reading Comprehension (MRC) studies, much progress is made in English. The Stanford Question Answer Data Set (SQuAD) [1], published by Rajpurkar in 2016, is a fundamental resource for MRC researches. Rather than the traditional rule-based approaches, researchers applied many neural network approaches to the MRC problem utilizing the SQuAD data set. Bidirectional Encoder Representatives from Transformers (BERT) [2] published by Google in 2019, revolutionized natural language processing. BERT has achieved very high accuracy in the MRQA problem by providing the state-of-the-art language model for English, anew using SQuAD data.

Although some multilingual models are trained for uncommon languages, there is still a problem of insufficient data and a lack of experience in how these algorithms can be applied accurately to the language. Moreover, Turkish has a challenging grammar for most of NLP tasks because of being an agglutinative language and having a rich but complex morphology, such as a comprehensive set of possible suffix tags and diversity of constituent orders in inverted sentences. Considering all these stated circumstances, the objective of the work performed under this thesis is to create a language model for Turkish, followed by fine-

tuning the model for the QA task. Creating a language model from scratch will be experienced, providing solutions to the problems encountered in the process. Moreover, obtaining sufficient training data for the QA task will be tough for rare languages such as Turkish.

The study's output will able to find the most correct and shortest answer from a given text to the question asked. The process is called MRQA, in the use of the QA systems and search engines. Previous studies and reports have shown that a performance-satisfactory MRQA system for Turkish has not been developed yet; almost all are statistical models.

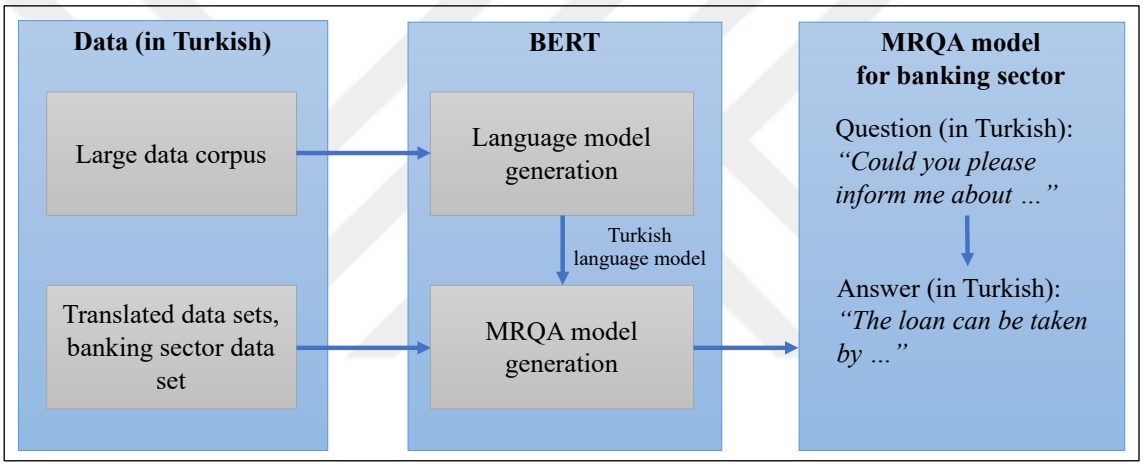

Figure 1.1. Diagrammatic representation of the study.

Nowadays, in the NLP domain, highly successful systems are based on architectures similar to BERT network. Utilizing large corpus, BERT trains a language model consisting of the contextualized word representations. The language model has a general knowledge of the grammar and semantic relationships of words; afterward, this model can be easily adapted to any chosen NLP task with little customization. For a language, training a model from scratch specific to the language is more successful when compared to the multilingual model published by BERT. In considering all these factors, a language model will be created from scratch for the Turkish language by the BERT deep learning methods and will be customized for the MRQA task that will serve in the banking sector. Figure 1.1 shows the block diagram of the process.

This study aims to apply deep learning approaches to the MRQA problem in Turkish, in answering questions over a given Turkish text to achieve high accuracy, as in English. The training procedure of the Turkish language model base on the official English BERT model published by Google. The proposed model achieved 55,26 percent exact match (EM) and 67,07 percent F1-Score for open-domain data sets, which is the union of SQuAD and NewsQA data sets translated to Turkish. The proposed model achieved an average score of 54,09 percent exact match (EM) and 79,01 percent F1-Score for closed-domain banking sector QA data set created in Turkish.

The thesis is arranged as follows; Section 2 gives an idea in the field of MRQA by mentioning the relevant studies in the literature. Section 3 describes MRQA, BERT, Transformers, and Attention in detail. Session 4 is the section describing the methodology, including the data and the training procedures. Section 5 contains details of the software pieces used in preprocessing data steps in this research. In Section 6, the comparison of the current study with other Turkish and other language models and evaluation results of different scenarios are mentioned. Finally, Section 7 concludes the findings and contains some information on future studies.

2. RELATED WORK

In Turkish questioning (QA) systems, in most of the researches to improve the skills of the search engines, two modules take part in front and behind the search engines, as seen in Figure 2.1. Comprehending the operating principles of the search engines, to present more fruitful results to the questions asked, the front module multiplexes the query or, on the contrary, deletes needless parts from the query. Rather than showing all result documents to the user, in order to find the optimum search outcome, the behind module filters the results of the search engine [3-6]. The most popular application is AnswerBus [7], which runs on the Bing search engine and serves in English by default. Though, using machine translation to English, AnswerBus also offers searches in several other languages.

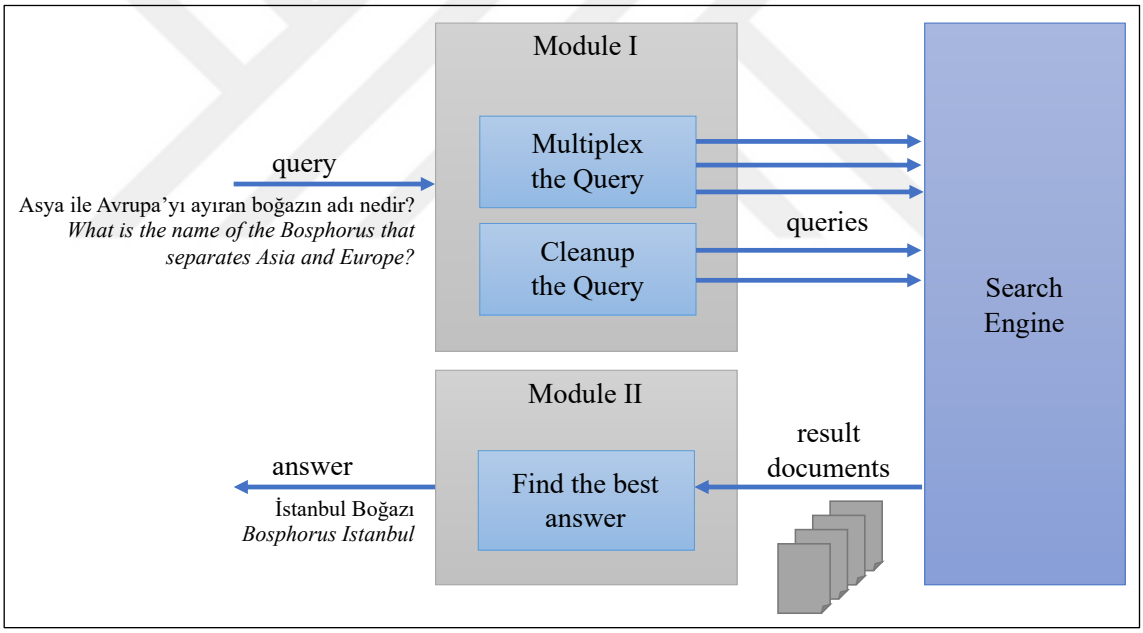

Figure 2.1. Enhancing search engines using MRC.

The front module advances the query, initially classifying the question via a predefined table that holds all potential representations, as seen in Figure 2.2. For building plain variations of the query, Turkish language processing libraries, such as Zemberek [8] or Treebank [9-10], parses the sentences and tokenizes the words to find the stems (Figure 2.3). Table 2.1 is an example of multiplexing a query; the second and third rows are derivative forms of the original query. Another approach excludes prepositions, conjunctions, stop words, and using the thesaurus expands the query by attaching the synonyms of the terms.

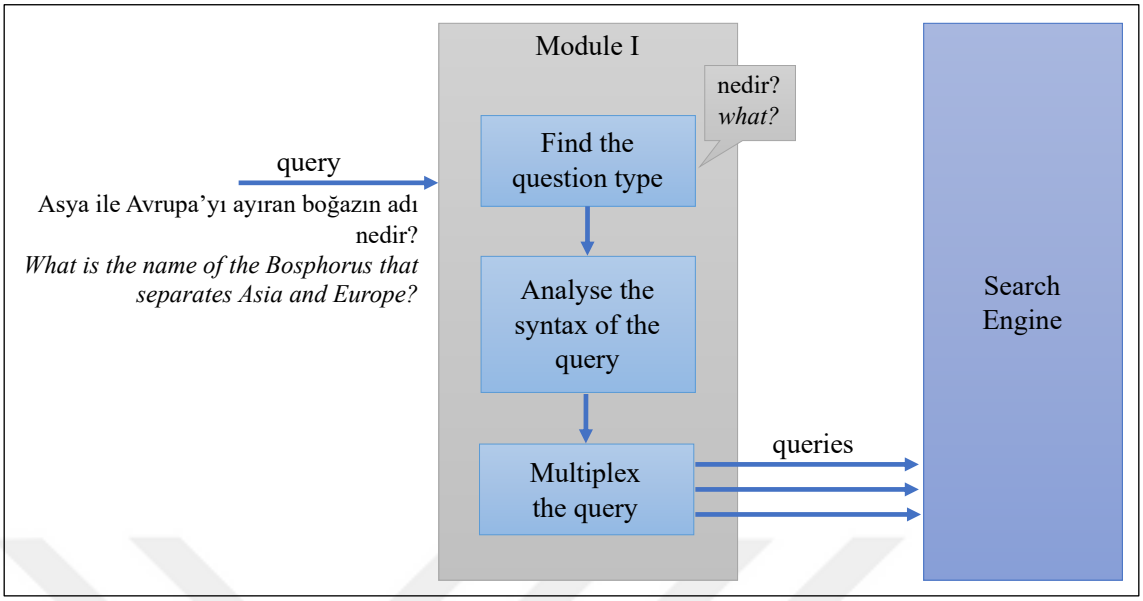

Figure 2.2. Module-I details.

Word	Dictionary Item	Morphemes
asya	[Asya:Noun,Prop]	asya:Noun+A3sg
ile	[ile:Conj]	ile:Conj
avrupayı	[Avrupa:Noun,Prop]	avrupa:Noun+A3sg+yı:Acc
ayıran	[ayırmak:Verb]	ayır:Verb|an:PresPart→Adj
boğazın	[boğaz:Noun]	boğaz:Noun+A3sg+ın:Gen
adı	[ad:Noun]	ad:Noun+A3sg+ı:P3sg
nedir	[Ne:Noun,Abbrv]	ne:Noun+A3sg|Zero→Verb+Pres+A3sg+dir:Cop
?	[?:Punc]	?:Punc

Figure 2.3. Morphological disambiguation example [3].

Table 2.1. Multiplexing the query example [3].

	Original text in Turkish	**English translation**
The original query	Asya ile Avrupa'yı ayıran boğazın adı nedir?	What is the name of the Bosphorus that separates Asia and Europe?
Query 1	Asya ile Avrupa ayıran boğazın adı	the name of the Bosphorus that separates Asia and Europe
Query 2	Asya ile Avrupa ayır boğaz ad	name Bosphorus separate Asia and Europe

In the candidate document results of the search engines, utilizing the class of question identified by the first module and its predefined answer patterns, the second module traverses the phrases that are resembling to be the answer to the question (Figure 2.4). The answer should include at least half of the query words and also existing named entities of the

question. In Table 2.1, the query words are *Asya*, *Avrupa*, *boğaz* and *ayır*; the named entities are *Asya* and *Avrupa*. Additionally, if the question requests a numerical value, the answer should include numbers.

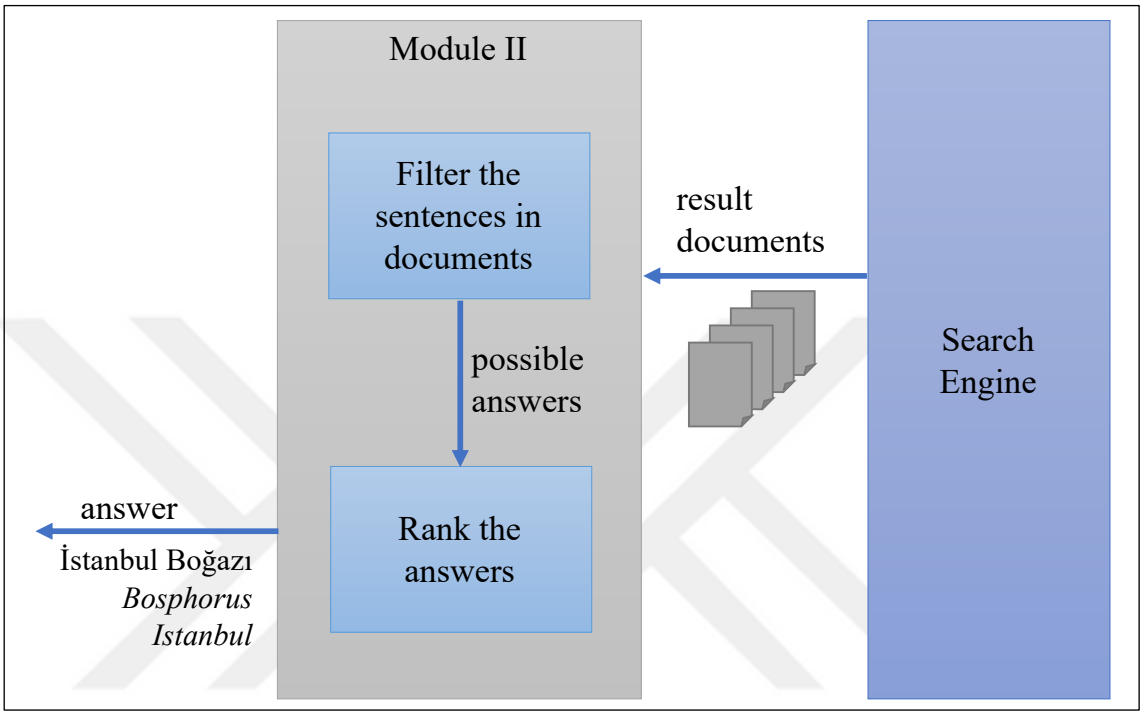

Figure 2.4. Module-II details.

After observing the potential answer statements, the following ranking formula puts in order the candidates to pick the closest match to the real answer. In Equation 2.1, is the number of query words existent in the potential answer, is the number of words in the query and is the total logical distances of words in the potential answer and query.

$$P = \left\lfloor \frac{10 * C_k * (C_k - 1)}{T_p * S_k} \right\rfloor \tag{2.1}$$

For feature extraction and named entity recognition, another study performs predefined pattern matching algorithms on the result documents (Figure 2.5). If question class is *when*, the answer is probably a combination of date or time, the system applies the patterns that are in the third and fourth rows of the table. If question class is *who*, the answer is probably a named entity, which is a person or a group, and the system applies the patterns that are in the first and sixth rows of the table.

Entity Name	Sample Pattern	# of Patterns
Name	\\p{Lu}+\\p{L}+\\s\\p{Lu}+\\p{L}+	4
Length	\\d+[.,]\\d+.kilometre	24
Time	[0-2]\\d:\\d\\d	1
Date	[0-3][0-9]\\D[0-1][0-9]\\D\\d\\d\\d\\d	55
Price	\\d+[.,]\\d+.[$]	75
Organization	\\p{Lu}+.\\p{L}+.[lL][tT][dD]	12
Percentage	%\\s\\d+[.,]\\d+	7
Weight	\\d+.kilogram	14
Location	\\p{L}+.caddesi	20
Total Pattern Usage :		**212**

Figure 2.5. Sample patterns [6].

A research study discovers the answer patterns of the question classes by crawling the Web. After generating question and answer pairs manually for every question class, the researchers search the created data set on the Web. The system collects the documents which include both question-answer pairs, afterward breaks into sentences to derive regular expressions of the answer pattern for the question class. In the *"the capital city of Turkey*" example, the crawler searches *Ankara AND Turkey* together on the Web. After assembling the sentences that include both, the system automatically forms the regular expressions in Figure 2.6, replacing *Turkey* with Q (Question) and *Ankara* with A (Answer).

- <Q>'nin başkenti olan <A>
- <A>, <Q>
- <Q>'nin başkenti olan <A> şehri
- başkent <A>, <Q>

Figure 2.6. Regular expression examples.

After the searching process, when the candidate documents are ready, the system executes the collected answer patterns via five various forms, as seen in Table 2.2: raw string, raw string with answer type, stemming string, stemming string with answer type, and named entity tagged string.

Table 2.2. Answer pattern results of "Türkiye'nin başkenti Ankara" [6].

Type	**Answer pattern**
Raw String	<Q>'nin başkenti <A>
Raw String with Answer Type	<Q>'nin başkenti <A-NECity>
Stemming String	<Q> başk <A>
Stemming String with Answer Type	<Q>'nin başk <A-NECity>
Named Entity Tagged String	Replace Ankara with Turkish Named Entity Tagger result

For the Geography lesson of Turkish-speaking high school students, some researchers [11] designed a closed-domain factoid QA system (HazırCevap) to support their education and find answers to their questions about their course of study. The system is akin to IBM Watson DeepQA technology; the Question Analysis module obtains the focus and class of the question. In the example of "What is the name of the largest plain in Turkey?", the focus is the name of a plain and the question class (QClass) is ENTITY.PLAIN. For focus extraction, the application uses a combination of a rule-based model (Distiller) and a Hidden Markov Model (HMM) based statistical model that uses a variation of the Viterbi algorithm. Distiller uses the dependency tree, as seen in Figure 2.7. Viterbi algorithm marks every word with two states: FOC means the word is a focus part, and NON means the word is a not focus part (Figure 2.8). After all, the system uses a combination of these two results.

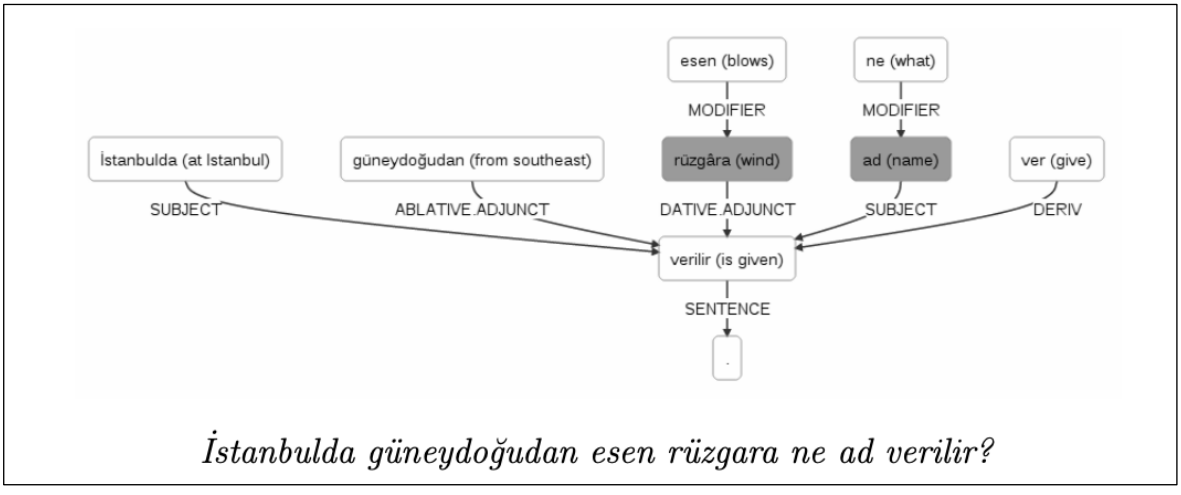

Figure 2.7. Distiller example [11].

forward serialization (->)				
Dış	ticaret	diğer	ad	ne
(external)	*(trade)*	*(other)*	*(name)*	*(what)*
FOC	FOC	NON	FOC	NON

Figure 2.8. Viterbi algorithm example [11].

For question classification, the application uses a rule-based model because of the success in finding coarse classes. Some sample phrases and their classes are in Figure 2.9.

Coarse Class	**Sample Phrases**
DESCRIPTION	"ne isim verilir", "temel sebebi nedir"
NUMERIC	"kaç", "ne kadar"
ENTITY	"rüzgar tipi", "dağı hangisidir", " hangi ova"
TEMPORAL	"hangi tarihte", "kaçıncı yüzyıl"
LOCATION	"hangi bölge", "nerede", "nereye"
ABBREVIATION	"açılım", "kısa yazılışı"
HUMAN	"kimdir", "kimin", "hangi kültür"

Figure 2.9. Sample phrases and their classes [11].

The system preprocesses the query by performing morphological analysis and disambiguation using the Turkish NLP Pipeline [12]. Feds query analysis, QClass and focus into search engines such as Indri and Apache Lucene, which are connected to the knowledge base (Figure 2.10). Similar to previous studies, HazırCevap removes stop words and punctuation marks.

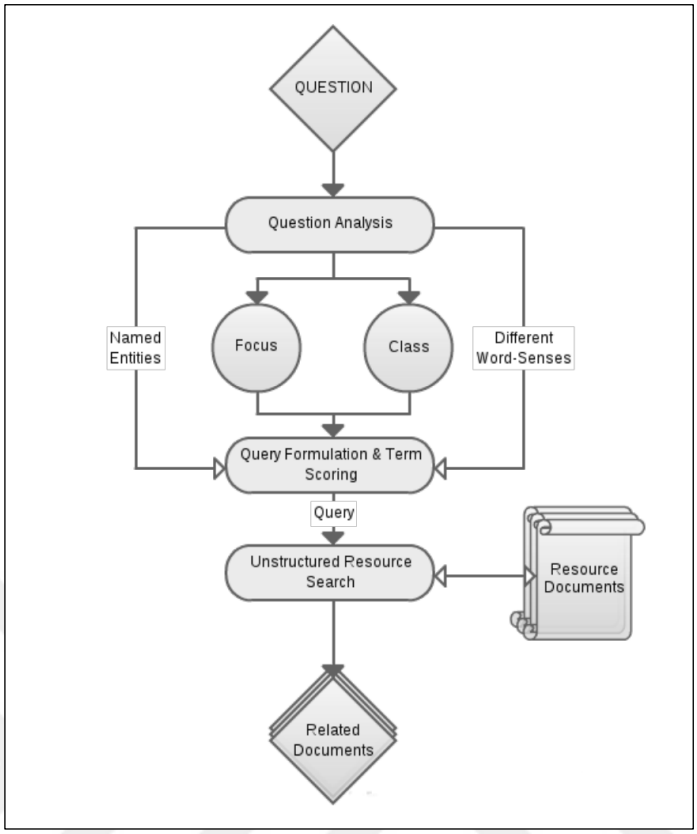

Figure 2.10. HazırCevap information retrieval module [11].

After large comparison datasets such as SQuAD, NewsQA [13], HotspotQA [14] are available, as a result of rapid improvements, the MRQA problem has substantially been solved for English. The SQuAD data set is generated by Stanford University, which contains more than 100,000 real question-answer pairs created by humans over 536 Wikipedia passages. It is available in 2 versions; in version 1.1, the answers to each question are apparent; however, in version 2.0, some questions have no answer. NewsQA data set is generated by Microsoft Research, which contains 120K question-answer pairs created by humans over CNN news articles.

Bi-Directional Attention Flow (BiDAF) [15] published in 2016, is a multi-stage hierarchical process that has different levels of granularity for representing the context as character-level (Char-CNN), word-level (GLoVE), and contextual embeddings. Instead of using a fixed-size vector for summarizing the paragraph, for every time step, the system computes the attention. For preventing early summarization and information loss, BiDAF uses a bidirectional attention flow mechanism for query-aware context representation: query to context and context to query which are generated from a similarity matrix (two boxes on the

right in Figure 2.11). Query2Context attention detects which query words are most related to the context words, and Context2Query discovers which context words are most relevant to the query words. With the representations of previous layers, the attended vector flows through the model layer (green and orange lines that goes into the modeling layer). This method prevents the information loss caused by early summarization. BiDAF uses memory-less attention, which means that the attention of every time step takes only the query and context of the current step as inputs, rather than directly depending on the attentions of previous time steps.

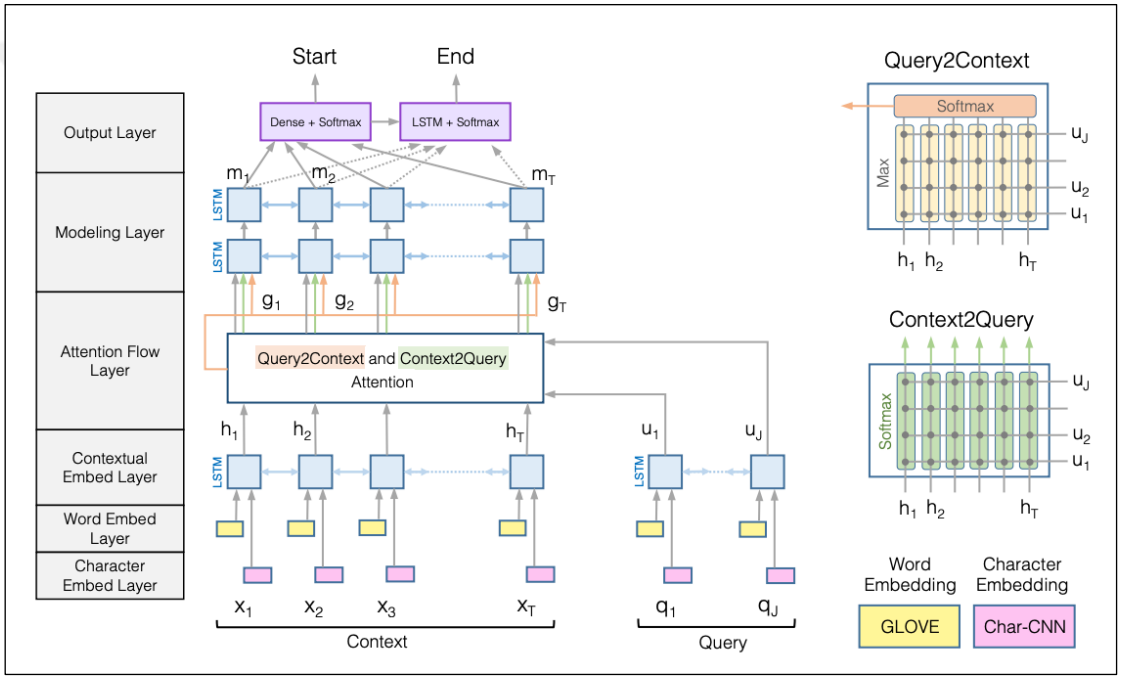

Figure 2.11. BiDAF architecture [15].

Pre-trained language representations were available with ELMo [16] and Generative Pre-trained Transformer (OpenAIGPT) [17]. ELMo is a feature-based approach that uses task-specific architectures that are pre-trained on a large text corpus, and ELMo supports six different NLP tasks, including question answering. OpenAIGPT is a fine-tuning approach: generative pre-training of a language model on a diverse corpus of unlabeled text, then discriminative fine-tuning for each particular task. Both are unidirectional; a left-to-right approach where each token has joined the previous tokens in Transformer's self-attention layers [18]. Uni-direction negatively effects some tasks for which it is crucial to walk in either direction in the context, such as questions answering.

In 2018, Bidirectional Encoder Representations from Transformers (BERT) was published by Google. Similar to OpenAIGPT, BERT uses a fine-tuning approach; but in contrast to ELMo and OpenAIGPT, BERT is bidirectional. BERT uses Masked Language Model (MLM) and Next Sentence Prediction (NSP); on pre-training, that masks and then predicts some randomly selected words, by using self-attention based Transformers. In this way, the model uses both the left and right contexts. BERT is easy to use and announced the state-of-the-art results for SQuAD v1.1: for ensemble model 87.4 percent exact match and 93.2 percent F1-Score, for single model 85.1 percent exact match and 91.8 percent F1-Score. The technical details of the BERT will be given in the next section.

In the MRQA 2019 Shared Task [19], the successes of the systems that offer a general solution to the MRQA problem for English were compared. Eighteen data sets, including SQuAD and NewsQA, were selected. These datasets were divided into training, dev, and test sets, each consisting of 6 datasets. Ten different teams tried to develop models using these data sets with various training methods developed on the BERT baseline. The success of the resulting models was compared to BERT.

After the great success in English, BERT has come into use for other languages. For some languages other than English, their unique models are trained instead of using BERT's existing multilingual model (mBERT) (Table 2.3). Since Arabic has a structure quite different from Latin languages, they have trained a BERT model specific to their language structures, named AraBERT [20], and tested it in several down-stream tasks, including QA. In tests conducted with the Arabic Reading Comprehension Dataset (ARCD), which was previously translated from SQuAD to Arabic, AraBERT has performed close to mBERT. There is an already model that Google has explicitly trained only for Chinese. This model has been tested in the following MRC datasets generated for Chinese: CMRC 2018, DRCD, CJRC [21]. Similarly, models named CamemBERT [22] for French, KoBERT [23] for Korean and ParsBERT [24] for Persian have been trained, but these models do not have a measured score on MRQA.

Table 2.3. Foreign Languages and BERT.

Language	**Model**	**Data set**	**EM**	**F-Score**
Arabic	mBERT	ARCD	34,2	61,3
	AraBERT	ARCD	30,6	62,7
Chinese	BERT-Chinese	CMRC	18,6	43,3
		DRCD	82,2	89,2
		CJRC	55,1	75,2

3. METHODOLOGY

This section gives an overview of the approaches that we have used in the study. Section 3.1 describes MRQA, and Section 3.2 contains modern deep learning techniques used for NLP tasks. Section 3.2.1 presents the details of the BERT architecture. Section 3.2.2 gives information about Attentions and Transformer Networks, which are the base of the BERT neural networks.

3.1. MRQA

Question Answering(QA) aims to create a system that automatically returns intelligent answers to questions that people ask in their natural (human) language. It is one of the research areas of computer science focused on Information Retrieval(IR) and Natural Language Processing(NLP). The information source may be a database, various reports, internet world, or a set of unstructured text documents. Many different methods, such as semantic analysis or text classification, can be used when creating QA systems. MRQA is a subset of QA, aiming to exact the most correct and shortest answer from an unstructured text, to the questions asked in human language.

For systems having question-answering facilities such as search engines or chatbots, machine reading is crucial for answering questions relating to documents. In MRQA, the system acquires the answer to the question in the given text. Average MRQA examples can serve for free-formed questions in a wide range of domains, called open-domain. Search engines are examples of open-domain MRQA systems. In Figure 3.1, an example from the SQuAD data set is presented. The paragraph is on the left; the questions and their answers

list is on the right. *Ground Truth Answers* are the answers taken from different people when the question asked for the paragraph and each answer is present in the paragraph.

Super Bowl 50 was an American football game to determine the champion of the National Football League (NFL) for the 2015 season. The American Football Conference (AFC) champion Denver Broncos defeated the National Football Conference (NFC) champion Carolina Panthers 24–10 to earn their third Super Bowl title. The game was played on February 7, 2016, at Levi's Stadium in the San Francisco Bay Area at Santa Clara, California. As this was the 50th Super Bowl, the league emphasized the "golden anniversary" with various gold-themed initiatives, as well as temporarily suspending the tradition of naming each Super Bowl game with Roman numerals (under which the game would have been known as "Super Bowl L"), so that the logo could prominently feature the Arabic numerals 50.

Question: Which NFL team represented the AFC at Super Bowl 50?
Answer: Denver Broncos

Figure 3.1. MRQA example from SQuAD data set.

The most critical segment in MRQA systems is the type of question. The questions can be in a wide variety of classes. There are five major question classes:

- Factoid Questions: The answer to the question is a single fact.
    - What year was Atatürk born?
    - What is Turkey’s largest lake?
- List Questions: The answer to the question is many facts.
    - What are the big cities of Turkey?
    - What are the names of the Turkish Presidents?
- Causal Questions: These are “what causes” questions.
    - What causes young people to start smoking?
    - What causes cancer?
- Confirmation: These are questions such as “whether or not”.
    - Alan Tuning was British, right?
    - Turkey is in Europe, isn’t it?
- Hypothetical Questions: These are the questions that are not based on reality.

- Let's assume you found the electricity, what would you do first?
- If the computer had never been discovered, what would we use instead?

In this thesis, we place more emphasis on factoid questions, but the study also covers free-formed queries. The data sets used on training and evaluation processes have been produced by human and includes all types of questions.

As mentioned in the literature review section, statistical approaches have been applied for many years to solve MRQA problems. The system classifies the question through predefined patterns, afterward applies the relevant answer finder regular expressions to the documents, for finding the potential answers. These systems only answer specific question types correctly. The common use of deep learning in the NLP field also resulted in improvements in the MRQA problem. There are many satisfying studies on this subject in English.

3.2. DEEP LEARNING FOR NLP

NLP has numerous sub-tasks, each of which needs task-specific extensive training data. One of the most prominent challenges in natural language processing is the training data quality and shortage. In the field of computer vision, this problem was solved by transfer learning, which started to be used with ImageNet. The idea is to train a neural network for general purpose, and then explicitly fine-tune if needed for a down-stream tasks. After seeing the benefits of the method, transfer learning started to be used in the field of natural language processing.

The first task in Figure 3.2, named *'pre-training'*, builds a general-purpose language model from a large amount of massive unlabeled text, called *'corpus'*. For a solid understanding of

the selected language, the system optimizes some parameters from scratch. OpenAI GPT is the first example.

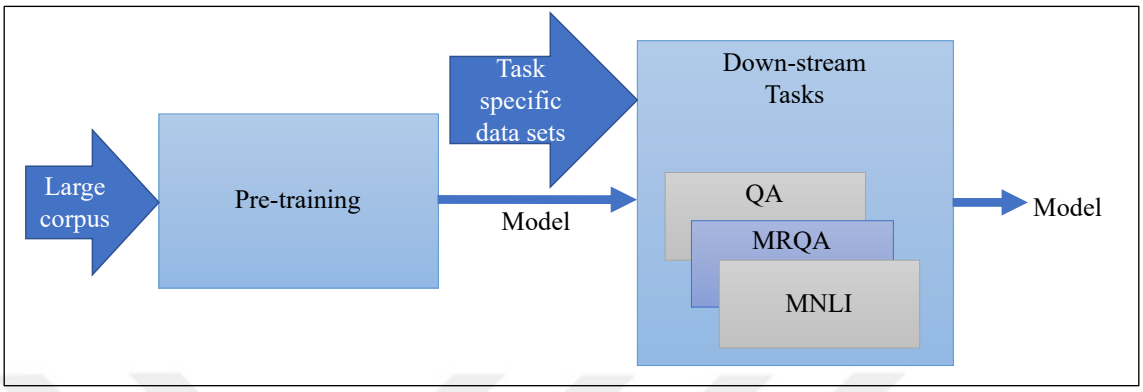

Figure 3.2. Transfer learning for NLP.

Pre-training representations can be context-free or contextual (Figure 3.3). Word2Vec and Glove are examples of context-free models that generate a single vector for every homonymic word. For the *'bank'* example, the word vectors of the *bank* are the same for *'I sat on the bank in the park'* and *'I came to see my bank accounts'*, although the first one is a seat and the second one is a finance office. Contextual models generate word vectors considering the meaning of the sentence. The difference of contextual model vectors from context-free models is that they handle polysemy and generate different word vectors for homonyms, looking at the overall meaning. For the *'bank'* example, the word vectors of the *bank* are not the same.

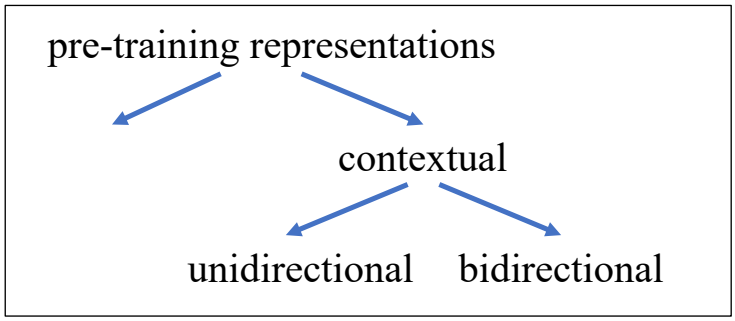

Figure 3.3. Pre-training representations.

When building a language model, the usual approach is using the previous words, called 'unidirectional' language models. Recurrent Neural Networks (RNN) and OpenAI GPT are unidirectional model examples. Bidirectional models use both from the left and right

contexts for seeing how well the word is in the sentence. ELMo is a shallow bidirectional model example. In Figure 3.4, in OpenAI GPT only *'I'* affects the word *'accessed'*, but in both BERT and ELMo *'account'* also has an effect on *'accessed'*.

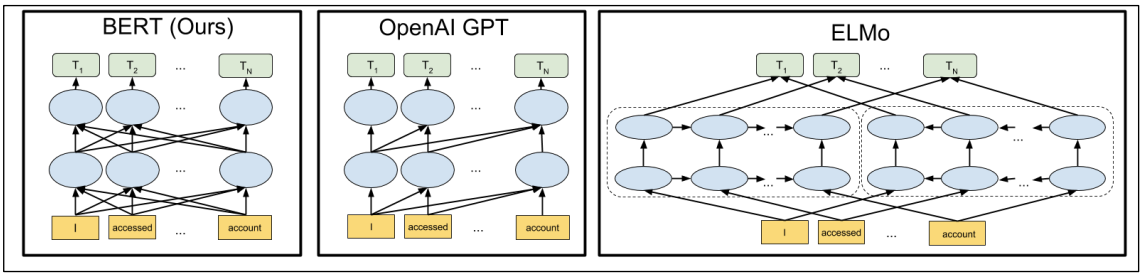

Figure 3.4. Comparison of BERT with other contextual models [2].

For the second task in Figure 3.2, named *'down-stream tasks',* two different methods are present which share the same objective: *feature-based* and *fine-tuning*. The feature-based approach uses task-specific architectures in the form of appending additional features to pre-trained representations. ELMo is an example of feature-based architectures. In the fine-tuning approach, the number of down-stream specific parameters is minimum, and the system improves pre-trained base model to find the optimum values of down-stream task parameters. OpenAI GPT and BERT are examples of fine-tuning architectures.

3.2.1. Bidirectional Encoder Representations from Transformers (BERT)

State-of-the-art models of NLP, such as BERT, RoBERTa, ALBERT, ELECTRA, XLNet, T5, use two significant tasks; pre-training and fine-tuning, as in Figure 3.5. BERT is the first deeply bidirectional unsupervised language representation, which is pre-trained using a plain text corpus. For any language, if BERT is pre-trained using a massive corpus, the output model will have a solid understanding of the selected language. After pre-training for fine-tuning, the system initializes the pre-trained model with learned weights and adjusts the weights for the optimal values of the selected down-stream task. BERT keeps the output models of both pre-training and fine-tuning as same as possible.

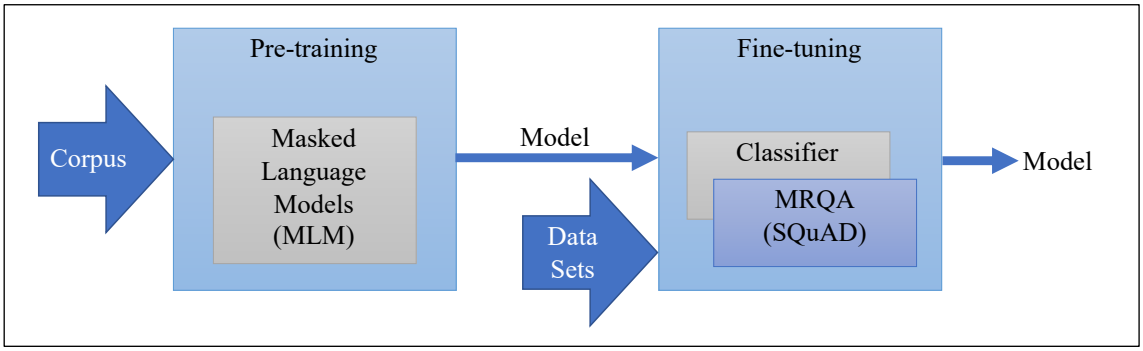

Figure 3.5. Training steps of BERT.

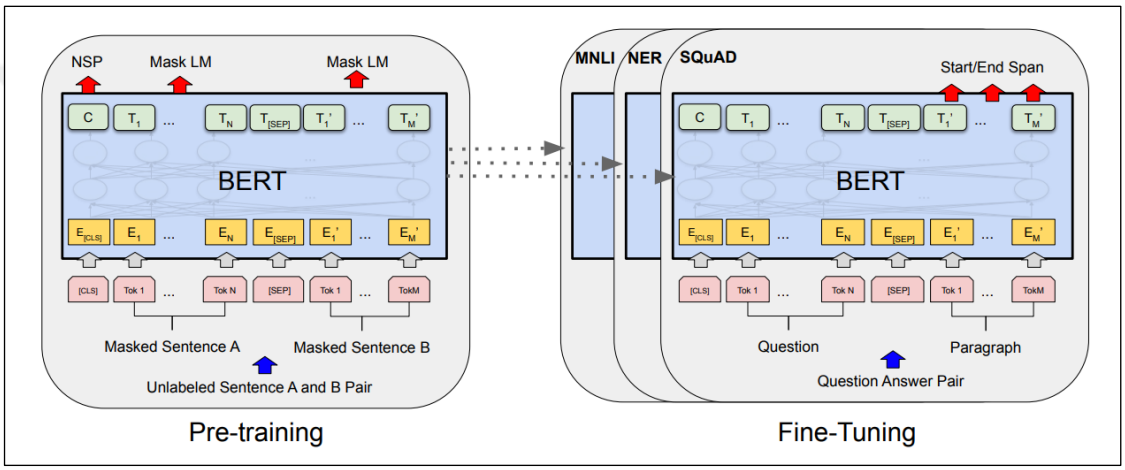

Figure 3.6. Procedures of BERT [12].

Fine-tuning is for some down-stream tasks such as MRQA, Named Entity Recognition, or text classification. In the MRQA, the system gets a data set such as SQuAD, NewsQA, or HotspotQA and improves the model to find answers to the questions in the given text. The SQuAD dataset is the most popular QA data set for studies in English and contains thousands of question & answer pairs generated from Wikipedia articles. In the training phase, the question and paragraph are the inputs of the neural network, and the answer is the output. The neural network tunes the weights to the optimum values to find the best text span that is closest to the real answer in the paragraph.

3.2.1.1. Task 1: Pre-training

BERT has technically two subtasks in a pre-training phrase: Masked Language Model (MLM) and Next Sentences Prediction (NSP), as seen in Figure 3.7. In the pre-training,

BERT trains both Masked LM and NSP together to minimize the combined loss functions. The output of the pre-training is the language model with the next sentence prediction results.

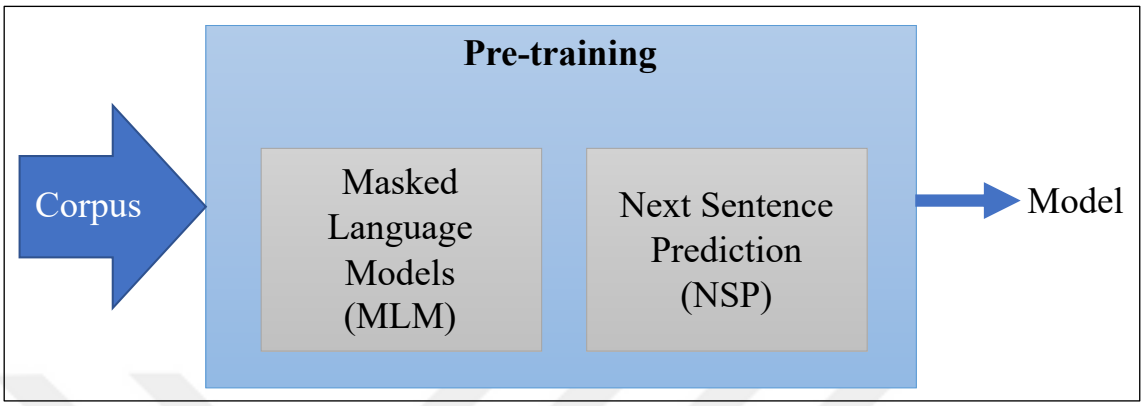

Figure 3.7. BERT pre-training subtasks.

Masked Language Modeling (MLM): The training method may be language models (LMs) or masked language models (MLMs). LM is the traditional method and generally used in unidirectional approaches. MLM is a new method that BERT recently announced and supports the bidirectional approach. MLM method masks some of the randomly selected tokens from the input and predicts the original vocabulary item of the masked word. By using a Softmax classifier, the network updates its weights by comparing the prediction with the real answer. In this way, the neural network learns the relations of the words in the language by handling the context both from left-to-right and right-to-left.

In Figure 3.8, BERT masks and guesses ‘on’ and ‘bank’. For masking a token, the general rule is as following; the selected token is 80% of the time replaced with [MASK], 10% of the time left the same, and 10% of the time replaced by a random token picked from the vocabulary (Figure 3.9). Because [MASK] tokens are not present in the fine-tuning task, usually, 15% of the tokens are masked for mitigation. This randomness improves the learning skills of the model. According to BERT’s paper, the magic of these percentages is; “If we used [MASK] 100% of the time the model wouldn’t necessarily produce good token representations for non-masked words. The non-masked tokens were still used for context, but the model was optimized for predicting masked words. If we used [MASK] 90% of the time and random words 10% of the time, this would teach the model that the observed word

is *never* correct. If we used [MASK] 90% of the time and kept the same word 10% of the time, then the model could just trivially copy the non-contextual embedding."

In training, 10 percent of 15 percent = 1.5 percent of all tokens are replaced with random words, and this is a deficient percentage to harm the general language model. More training steps are required in MLM compared with LMs since only 15 percent of tokens can be masked, and only the masked words can be predicted and learned for each step. Therefore, MLM is slower than typical LMs, but the result success rate is satisfactory. A full example of the masking is in Figure 3.10 ; for every step, BERT selects and masks a token. For training, the Transformer network is used, which is described in 3.2.2 section.

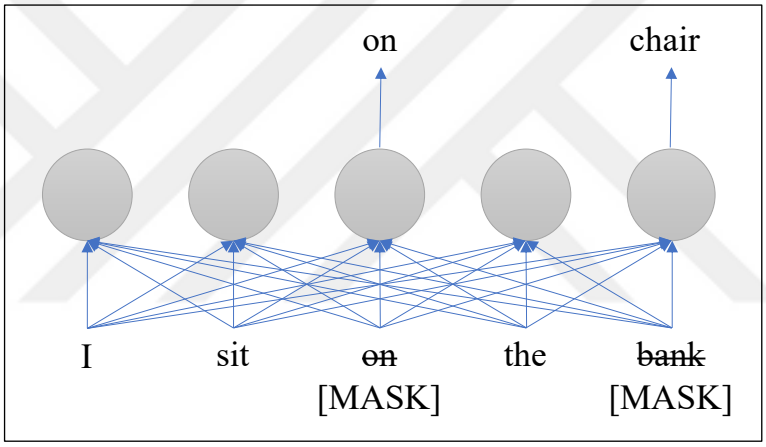

Figure 3.8. MLM masks some of the tokens and tries to predict.

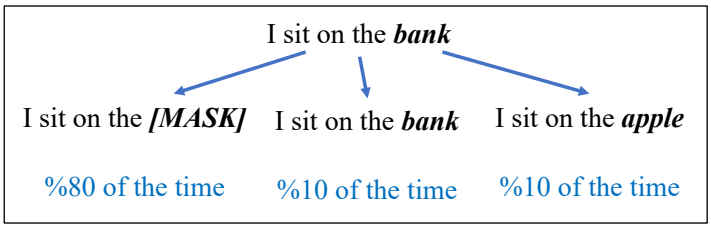

Figure 3.9. Masking example.

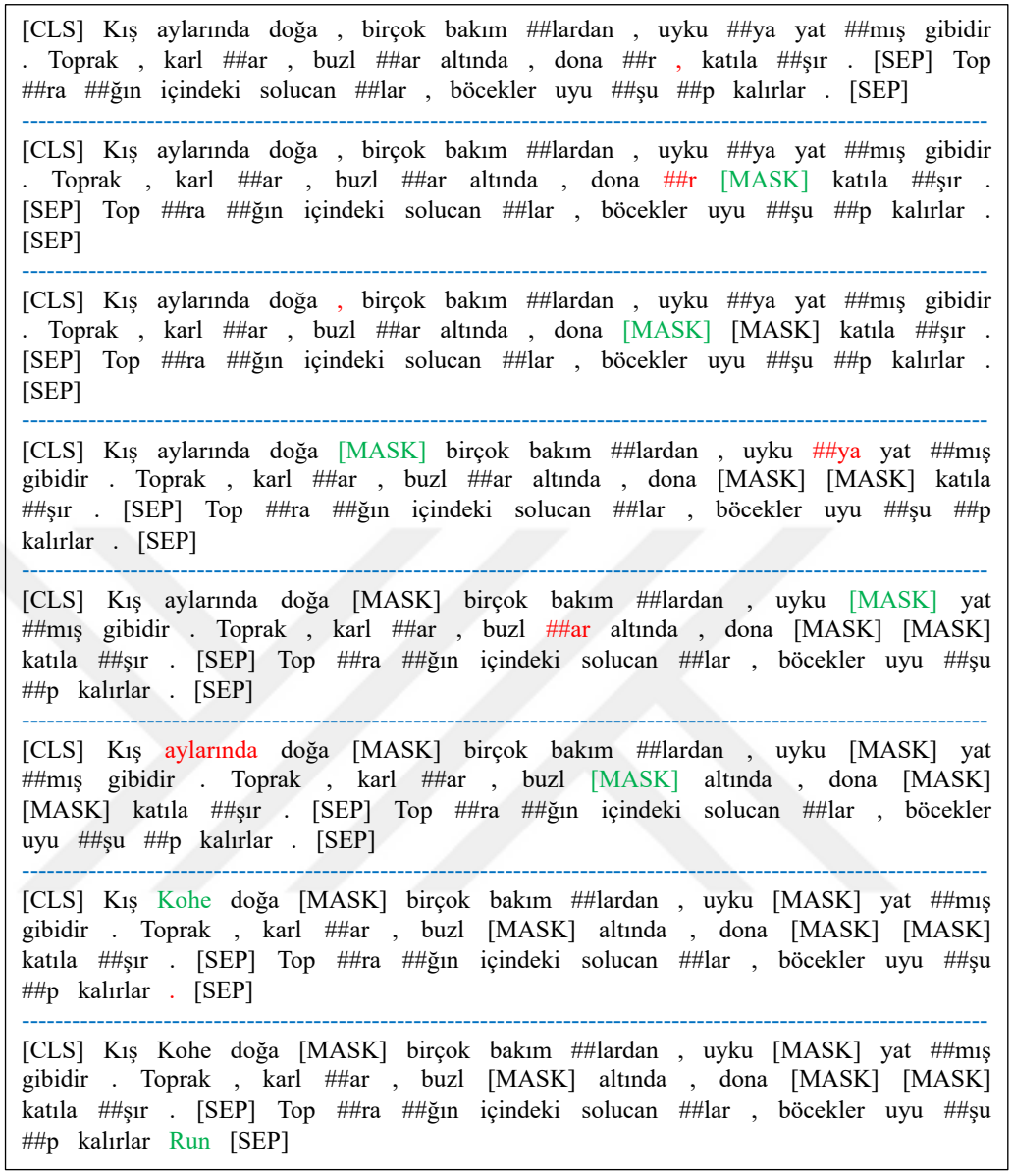

Figure 3.10. Masking journey of an input. Red items are replaced with green items.

For tokenization, BERT uses the WordPiece algorithm of Google [25]. WordPiece creates a fixed-sized vocabulary that includes words, subwords, or characters (Figure 3.11). The algorithm calculates the frequencies of tokens in the corpus and picks the most frequent ones. Vocabulary file includes both suffixes, subwords, and also words. Most probably, every word in a language can be tokenized by using the generated vocabulary file as a reference. If the vocabulary file is updated, the tokenization result of sentences also changes. For

'helping' as an example, the tokenizer may split as *'help'* and *'##ing'*, but if *'helping'* is added to the vocabulary, *'helping'* will be tokenized directly as *'helping'* without splitting.

```
[PAD]
[EOS]
[UNK]
[CLS]
[SEP]
[MASK]
ve
bir
da
##n
de
olarak
ile
##a
##e
##i
bu
için
olan
##ı
yılında
##k
```

Figure 3.11. Vocabulary example for Turkish.

There are three reserved tokens in BERT architecture:

1- **[CLS]:** Starter token of every input sequence and the classification token for Softmax classifier. It is used in both pre-training and fine-tuning.
2- **[SEP]:** Delimiter token, which splits the sequences. It is used in both pre-training and fine-tuning. In pre-training, it cuts the two sentences. In fine-tuning, it cuts the input sequence as question and paragraph.
3- **[MASK]:** Mask token used in pre-training, which indicates BERT masks this token.

In pre-training, BERT uses WordPiece tokenizer for splitting the words into pieces that may be a character, subword, or word, named *'tokens'*. WordPiece tokenizer bases on a vocabulary file, which consists of possible tokens of the language. Every vocabulary item has a unique id, and the output tokens of the tokenizer are replaced with their token-ids for generating the input sequence, named "*token embeddings"*. Although BERT is not a sequential method like RNN or LSTMs, *positional embeddings* are build using the locations of the words in the sentence, for preventing long-distance mappings with unnecessary tokens

in self-attentions. Figure 3.12 shows an example of two sentences, their tokens and embeddings which are the inputs of the Transformer network. The text-based inputs of the BERT are called "*sequences*". Each sequence starts with [CLS] and has two sentences. To highlight the owner sentence of the tokens as A or B, *sentence embe*dding is used. In the example, A represents the first sentence, B represents the second sentence. BERT adds all these embeddings together for generating the final input embedding of the deep neural network.

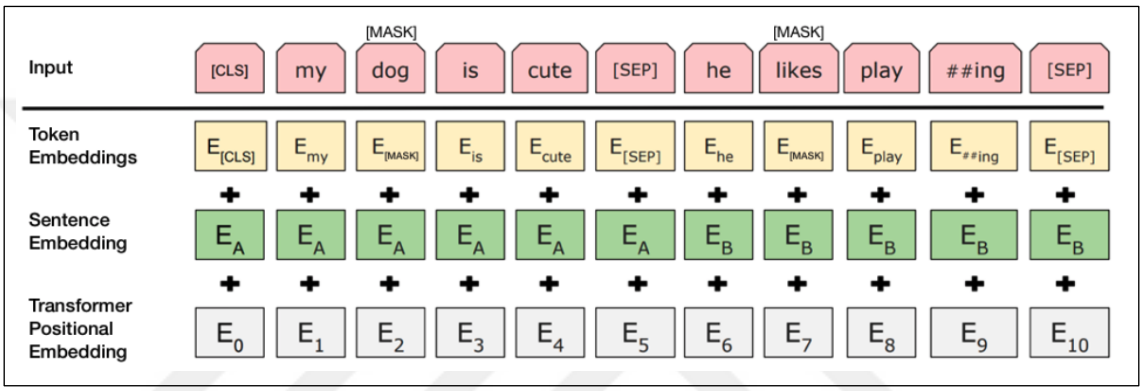

Figure 3.12. BERT input sequence example [2].

Figure 3.13 illustrates the full architecture of MLM. The system selects some tokens for masking and then converts the input *"New England Patroits … night"* to embeddings. The summed embeddings are the input of the Encoder. Using a self-attention mechanism, Encoder generates word representations, which are the input of the Decoder. Word representations form the language model, and like neural network hyperparameter weights, BERT optimizes these matrices using the standard back-propagation methods. With the Softmax loss function, the decoder compares the real and predicted values of every masked token and enhances the weights.

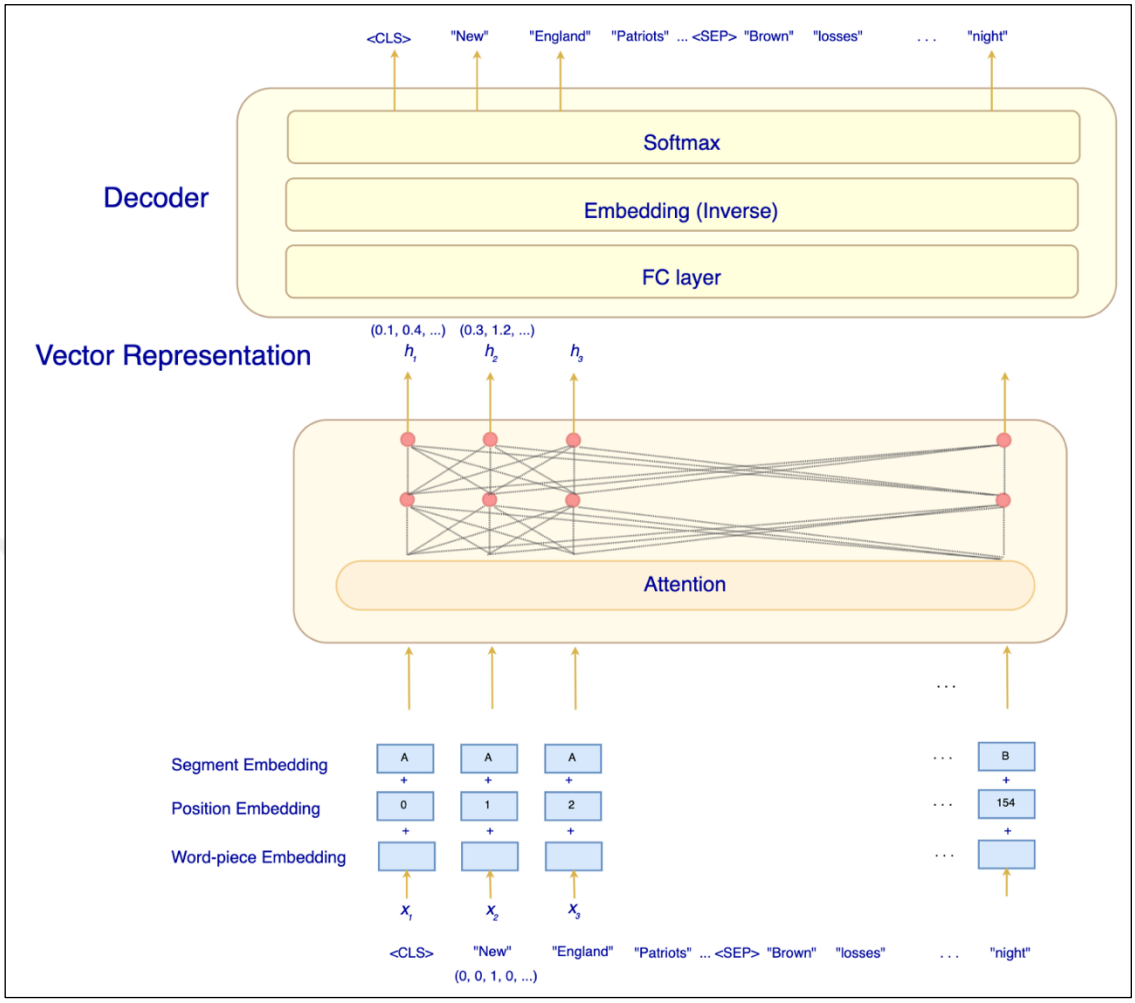

Figure 3.13. MLM architecture [26].

Next Sentence Prediction (NSP): For some NLP tasks such as Question Answering (QA) and Natural Language Extraction (NLI), understanding the relationship between the two sentences is essential. Therefore, in the pre-training task, the Next Sentence Prediction (NSP) subtask is also performed. Considering A and B sentence pairs, there is a random choice; sometimes B is left in its correct form and marked as *IsNext*, or sometimes B is replaced with a random sentence picked from the corpus and marked as *NotNext*. Figure 3.14 shows examples of B sentences; *Not Random* indicates *IsNext* and *Random* indicates *NotNext*. *IsNextSequence* probability is calculated using Sigmoid. During NSP training, BERT uses binary classification (Figure 3.15). In the figure, Classifier takes "C" as the input and using a shallow classifier predicts whether B is the next sentence of A or not.

Document 1

Kış aylarında doğa, birçok bakımlardan, uykuya yatmış gibidir.
Toprak, karlar, buzlar altında, donar, katılaşır.
Toprağın içindeki solucanlar, böcekler uyuşup kalırlar.

Not Random

A : ['Kış', 'aylarında', 'doğa', ',', 'birçok', 'bakım', '##lardan', ',', 'uyku', '##ya', 'yat', '##mış', 'gibidir', '.', 'Toprak', ',', 'karl', '##ar', ',', 'buzl', '##ar', 'altında', ',', 'dona', '##r', ',', 'katıla', '##şır', '.']
B : ['Top', '##ra', '##ğın', 'içindeki', 'solucan', '##lar', ',', 'böcekler', 'uyu', '##şu', '##p', 'kalırlar', '.']

Document 2

Koronavirüs tüm dünyada reel ekonomileri etkiledi.
Mart ve nisan aylarına ait veriler açıklandıkça virüsün ekonomiler üzerindeki etkileri de ortaya çıkmaya başladı.
Reel ekonomiye ait veriler takip edilirken, virüsün yayılmasının pik yaptığı ve ekonomi üzerindeki etkilerinin azalmaya başladığı konusunda bir görüş de oluşmaya başladı.

Random

A : ['Koro', '##nav', '##ir', '##üs', 'tüm', 'dünyada', 'reel', 'ekonomi', '##leri', 'etkiledi', '.', 'Mart', 've', 'nis', '##an', 'ayları', '##na', 'ait', 'veriler', 'açıklandı', '##kça', 'virüsü', '##n', 'ekonomi', '##ler', 'üzerindeki', 'etkileri', 'de', 'ortaya', 'çıkmaya', 'başladı', '.']
B : ['Toprak', ',', 'karl', '##ar', ',', 'buzl', '##ar', 'altında', ',', 'dona', '##r', ',', 'katıla', '##şır', '.', 'Top', '##ra', '##ğın', 'içindeki', 'solucan', '##lar', ',', 'böcekler', 'uyu', '##şu', '##p', 'kalırlar', '.'] ← From document 1

Random

A : ['Reel', 'ekonomiy', '##e', 'ait', 'veriler', 'takip', 'edilirken', ',', 'virüsü', '##n', 'yayılmasın', '##ın', 'pi', '##k', 'yaptığı', 've', 'ekonomi', 'üzerindeki', 'etkilerini', '##n', 'azalma', '##ya', 'başladığı', 'konusunda', 'bir', 'görüş', 'de', 'oluşma', '##ya', 'başladı', '.']
B : ['Kış', 'aylarında', 'doğa', ',', 'birçok', 'bakım', '##lardan', ',', 'uyku', '##ya', 'yat', '##mış', 'gibidir', '.', 'Toprak', ',', 'karl', '##ar', ',', 'buzl', '##ar', 'altında', ',', 'dona', '##r', ',', 'katıla', '##şır', '.', 'Top', '##ra', '##ğın', 'içindeki', 'solucan', '##lar', ',', 'böcekler', 'uyu', '##şu', '##p', 'kalırlar', '.'] ← From document 1

Figure 3.14. Segment generation example.

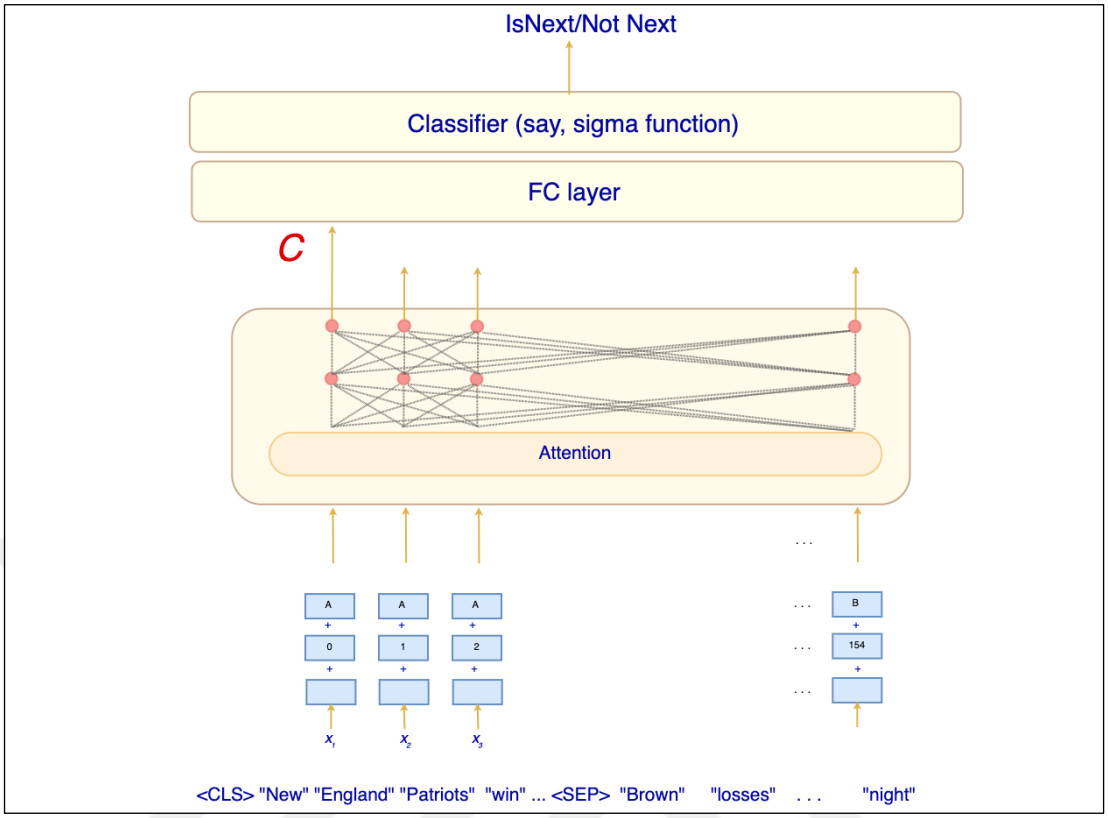

Figure 3.15. NSP architecture [26].

Parameters: The critical parameters of the pre-training task are in Table 3.1. In the pre-training task, the *maximum sequence length* shapes the input vector size of the Transformer network, and high values are needed to learn positional embeddings in long sequences. BERT selects the minimum of *maximum predictions per sequence* and *masked LM probability* multiplied by token size (in the formula below), for the number of word pieces to mask. *Do lower case* and *do whole word mask* parameters belong to tokenization and masking. In the NLP benchmarks, the published cased and whole word masked language models have higher accuracies, notably for Asian languages such as Chinese or Arabic. The current study is Turkish specific, and vocabulary file is wide enough to have most of the words already; consequently, subword or whole word masking results similarly on tokenization.

$$\text{masking threshold} = \min(\textit{maximum predictions per sequence}, \text{number of tokens} * \textit{masked LM probability}) \quad (3.1)$$

Table 3.1. Major parameters of the pre-training task [26].

Parameter	**Default value**	**Details**
max_seq_length	384	The maximum total input sequence length after WordPiece tokenization. Sequences longer than this will be truncated, and sequences shorter than this will be padded.
max_predictions_per_seq	20	Maximum number of masked LM predictions per sequence.
do_lower_case	True	Whether to lower case the input text. Should be True for uncased models and False for cased models.
do_whole_word_mask	False	Whether to use whole word masking rather than per-WordPiece masking.
masked_lm_prob	0.15	Masked LM probability.
short_seq_prob	0.1	Probability of creating sequences which are shorter than the maximum length.
dupe_factor	10	Number of times to duplicate the input data (with different masks).

Implementation Details: In the pre-training implementations, BERT has two subtasks: *create pre-training data* and *run pre-training*. *Create pre-training data* reads corpus files and vocabulary, then generates TensorFlow records which will be used in *run pre-training* task. As seen in Figure 3.16, most of the steps are based on random decisions. The task creates segments as **[CLS] Sentence A [SEP] Sentence B [SEP]**, which will be used in the Next Sentence Prediction (NSP) task. If the generated segment is larger than the *maximum sequence length* configuration parameter, system truncates the input from front or end. When the segment is ready, the system masks some tokens until the number of masked tokens reached the threshold that is calculated using configuration parameters. *Run pre-training* task reads the generated TensorFlow records and trains the system for MLM and NSP with transformers, which will be detailed on the 3.2.2 section.

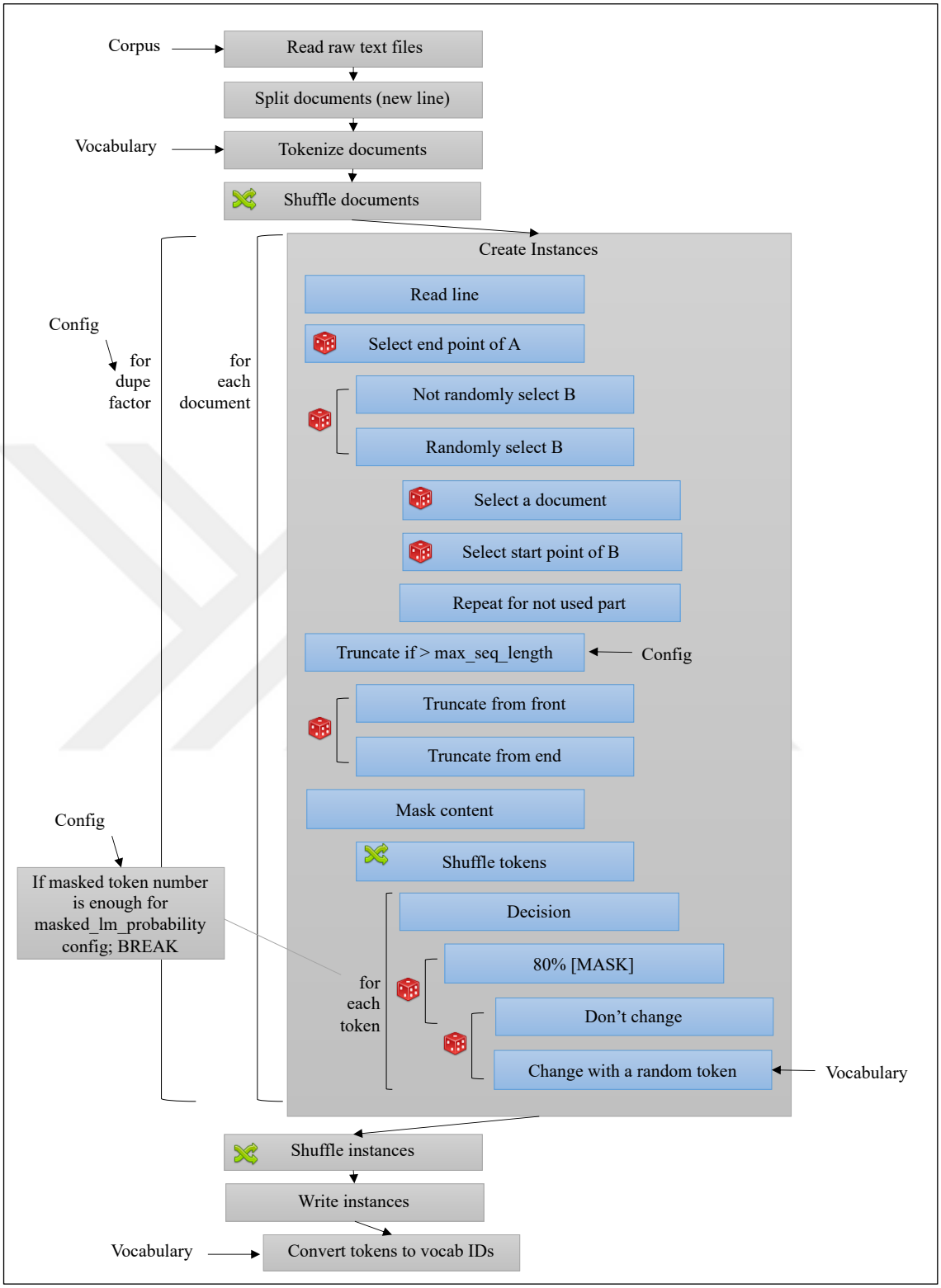

Figure 3.16. Create pre-training data steps. Dices indicate random decisions. Brackets with -for- indicate loops.

3.2.1.2. Task 2: Fine-tuning

Compared with pre-training, the fine-tuning step is speedy. For every down-stream NLP task, by adding an extra layer to the pre-trained model, BERT tunes all the task-specific parameters.

Down-stream NLP tasks can be grouped as following:

- *Text Classification*: Classification tasks, such as sentimental analysis, are sequence level subjects and BERT trains the system with a very similar approach to Next Sentence Prediction. For classification, BERT adds an extra layer at the top of the system and the output of the *[CLS]* token is the class label. In Figure 3.17, (a) and (b) are examples of text classification tasks.

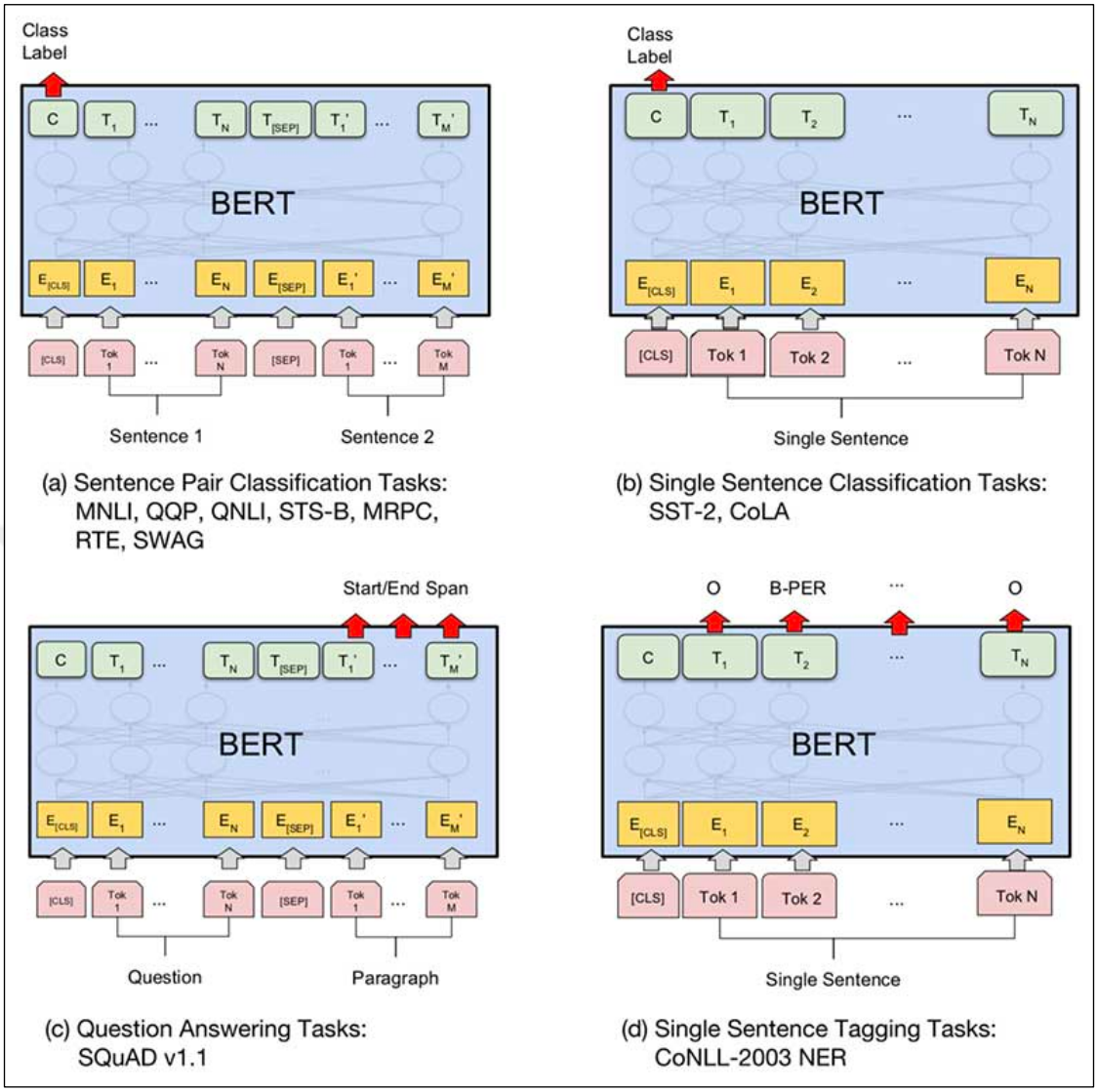

Figure 3.17. BERT down-stream tasks [2].

- *Named Entity Recognition (NER)*: The NE may be people, groups, locations or an organization. NER is a token-based task and marks all entities in the given text with their corresponding NEs. For NER, BERT adds an extra layer at the top of the system. If the token is a part of an NE, the output is the tag of the NE, and else the output is *NULL*. In Figure 3.17, (d) is an example of NER task.
- *Question Answering (QA)*: QA is a token-based task. The inputs are a question and a paragraph; the output is the start and end positions of the answer span. In Figure 3.17, (c) is an example of the QA task.

The focus of this thesis is the *Question Answering* down-stream task, which is also referred as MRQA. For MRQA training, SQuAD is the most common data set. An example SQuAD JSON file is in Figure 3.18. The SQuAD is an array of paragraphs and their question & answer pairs. Every question has an id and every answer has a start-point marker. For the answer, the reason for the *start-pointer* extra field is that the answer text may appear multiple times in the paragraph. The field is calculated by counting the number of characters from the beginning of the paragraph to the answer starting point (the indexOf function in programming). The SQuAD file has two versions: 1.1 and 2.0. In version 2.0, there are some unanswerable questions whose answers are NULL. This study focuses on SQuAD v1.1.

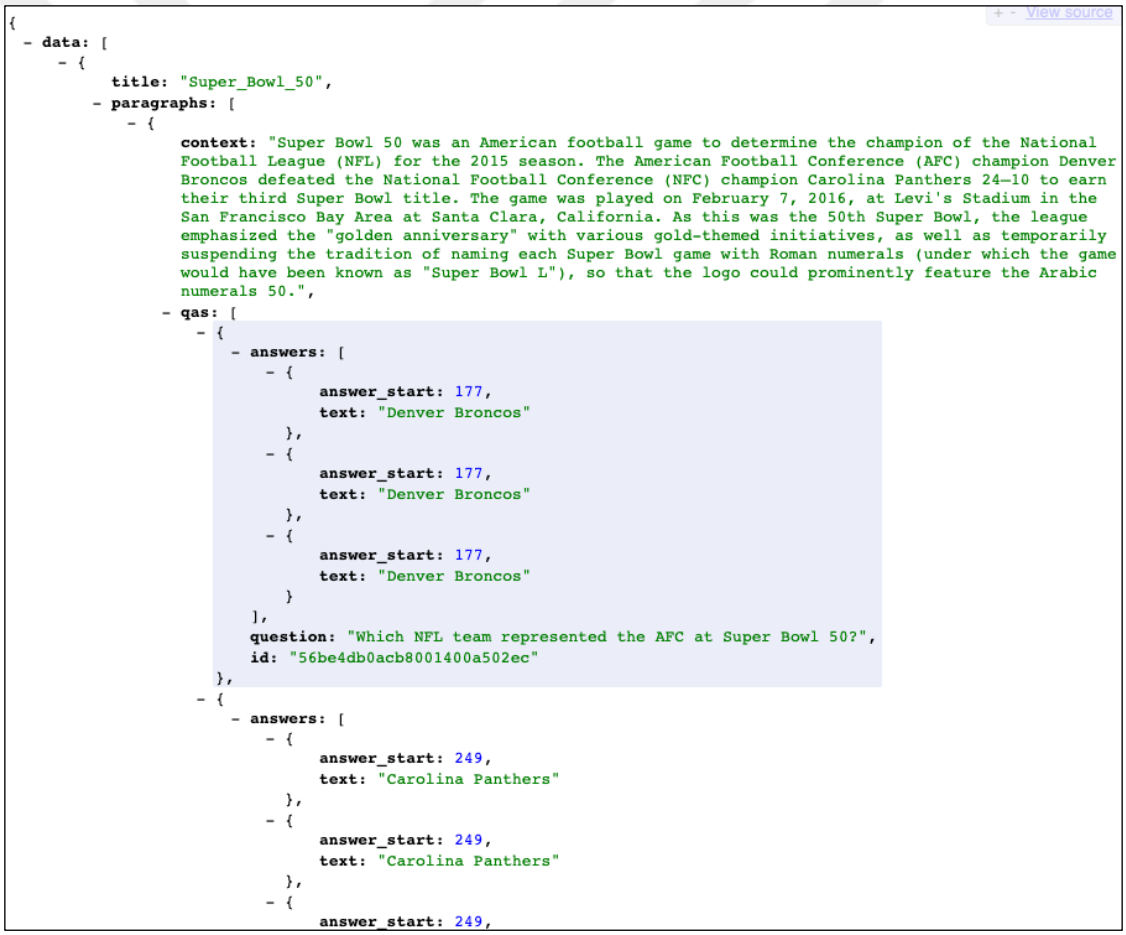

Figure 3.18. SQuAD JSON file example.

Parameters: There are crucial parameters; shown in Table 3.2. In the fine-tuning task, the input sequence of the Transformers is the union of the question and the paragraph; the output is the answer. *The maximum query length* is the number of question tokens located in the input sequence, and BERT ignores the longer part of the text. In the input sequence, a certain

number of tokens remain for the paragraph. If the paragraph is longer than the capacity, BERT splits the paragraph into chunks by a sliding window approach. It uses the *document stride* parameter to calculate the start point of the next chunk in the paragraph by adding to the previous value at every turn. On predicting the answer, the selected span from the text can be as much as *the maximum answer length* parameter, which is character-based despite all other parameters are token-based.

Table 3.2. Fine-tuning SQuAD task crucial parameters [26].

Parameter	**Default value**	**Details**
max_seq_length	384	The maximum total input sequence length after WordPiece tokenization. Sequences longer than this will be truncated, and sequences shorter than this will be padded.
max_query_length	64	The maximum number of tokens for the question. Questions longer than this will be truncated to this length.
max_answer_length	30	The maximum length of an answer that can be generated. This is needed because the start and end predictions are not conditioned on one another.
doc_stride	128	When splitting up a long document into chunks, how much stride to take between chunks.
do_lower_case	True	Whether to lower case the input text. Should be True for uncased models and False for cased models.
n_best_size	20	The total number of n-best predictions to generate in the nbest_predictions output file.
version_2_with_negative	False	If true, the SQuAD examples contain some that don't have an answer.
null_score_diff_threshold	0.0	If null-score – best_not_null is greater than the threshold predicts null.

Implementation Details: Figure 3.19 represents the step of the fine-tuning task. The system reads the SQuAD JSON file and converts them to SQuAD objects after validation. There are some rules for SQuAD samples:

- For training, each question should have exactly one answer.
- Answers should be in the documents. Because of some problems like Unicodes, if answer is not in the document, system skips the sample.

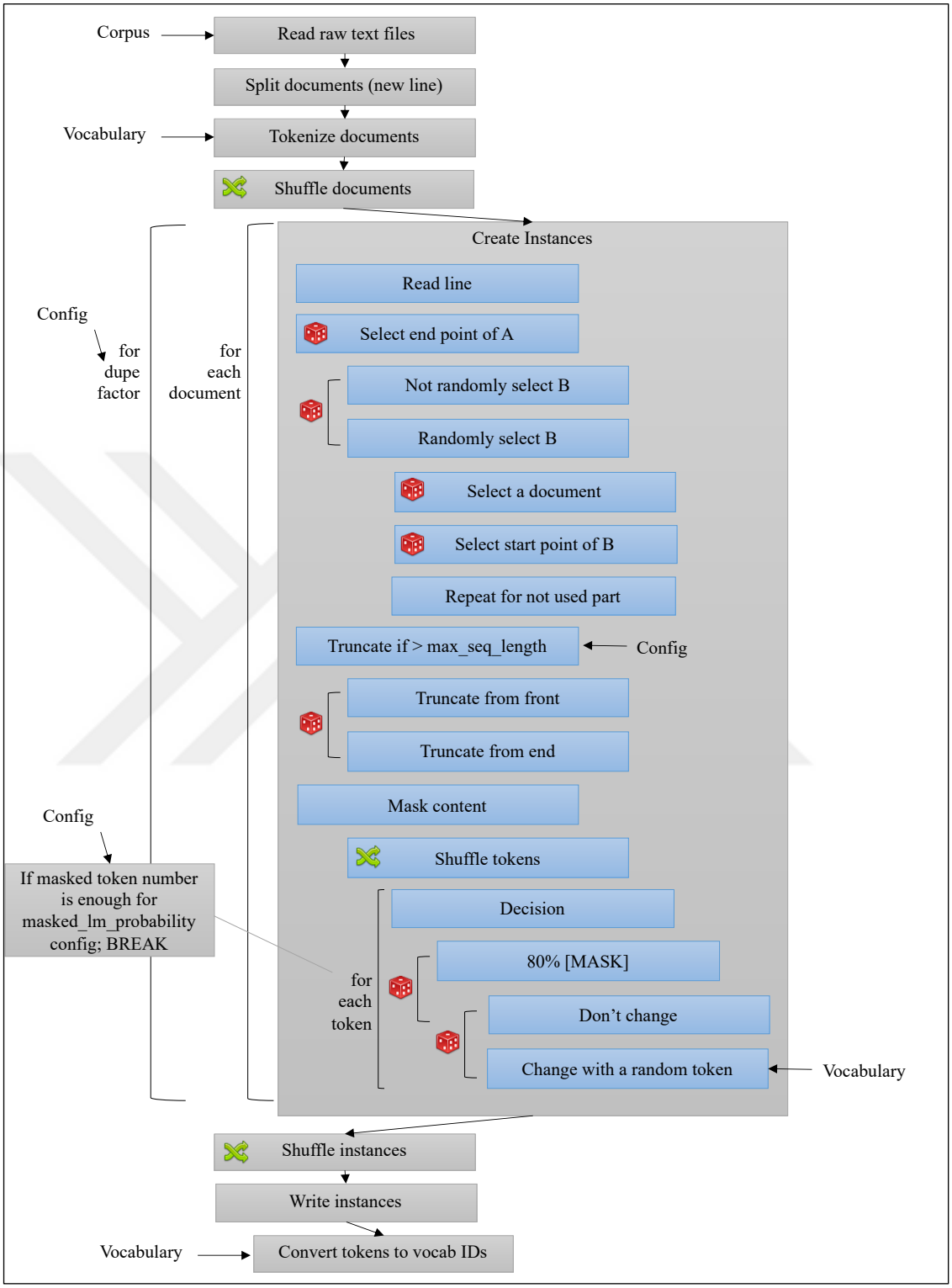

Figure 3.19. Fine-tuning steps.

A SQuAD object is in Figure 3.20: *qas_id* is the question ID, *doc_tokens* is the tokenized array form of the paragraph, *start_position* and *end_positions* are the begin & end indexes of the answer in the *doc_tokens* array.

qas_id	56bf10f43aeaaa14008c9501
question_text	Super Bowl 50 hangi ay, gün ve yıl gerçekleşti?
doc_tokens	['Super', 'Bowl', '50,', '2015', 'sezonunda', 'Ulusal', 'Futbol', 'Ligi', '(NFL)', 'şampiyonunu', 'belirlemek', 'için', 'bir', 'Amerikan', 'futbol', 'oyunuydu.', 'Amerikan', 'Futbol', 'Konferansı', '(AFC)', 'şampiyonu', 'Denver', 'Broncos,', 'üçüncü', 'Super', 'Bowl', 'şampiyonluğunu', 'kazanmak', 'için', 'Ulusal', 'Futbol', 'Konferansı', '(NFC)', 'şampiyonu', 'Carolina', 'Panthers', "24-10'u", 'yendi.', 'Oyun', **'7', 'Şubat', "2016'da",** 'Santa', 'Clara,', "California'daki", 'San', 'Francisco', 'Körfez', "Bölgesi'ndeki", "Levi's", "Stadı'nda", 'oynandı.', 'Bu', '50.', 'Süper', 'Kase', 'olduğu', 'için,', 'lig,', 'çeşitli', 'altın', 'temalı', 'girişimlerle', '"altın', 'yıldönümü"', 'nü', 'vurguladı', 've', 'her', 'bir', 'Super', 'Bowl', 'oyununu', 'Romen', 'rakamlarıyla', '(oyunun', '"olarak', 'bilinen"', 'Süper', 'Kase', 'L', '"),', 'böylece', 'logo', 'belirgin', 'şekilde', 'Arap', 'rakamlarına', '50', 'sahip', 'olabilir.’]
start_position	39
end_position	41

Figure 3.20. SQuAD object example.

The input sequence length of the neural network is configured by *max_seq_length*. In Figure 3.21, the *tokens* field is the array form of the tokenized input sequence. In *tokens_ids* array, the tokens are replaced with their vocabulary IDs. There are three separator tokens: [CLS], [SEP], [SEP]. The first part, between [CLS] and [SEP] tokens, is the question. The second part, between [SEP] and [SEP] tokens, is the paragraph. In *segment_ids*, the question is marked as 0, the paragraph is marked as 1. If the input sequence length is smaller than the *max_seq_length* parameter, it is padded with 0’s as seen in the *input_mask* field. The *start and end position* fields are the indexes of the answer in the array.

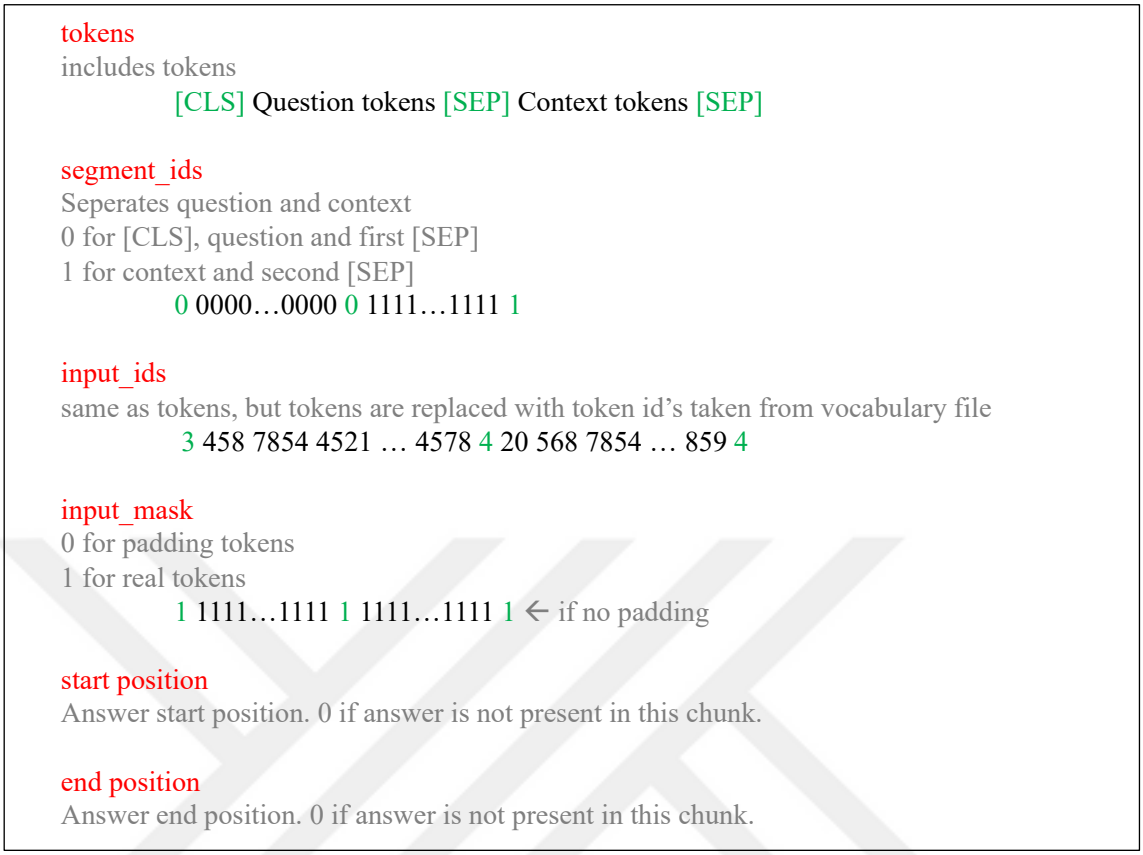

Figure 3.21. Fine-tuning chunks.

In the input sequence, the capacity reserved to the paragraph is the remaining slots from the question + three delimiter tokens. Question length is dynamic and can be up to *maximum query length*. The capacity of the document is calculated using formulas 3.2 and 3.3. The paragraph may be longer than its capacity. To deal with this problem, BERT uses a sliding window approach. The system uses chunks as dividing the paragraph up to capacity sized parts. In the paragraph, the start point of every chuck is calculated by the *doc_stride* parameter.

$$\begin{aligned} \text{maxTokensForDoc} \\ = \text{maxSeqLength} \\ - \text{len}(\text{questionTokens}) - 3 \end{aligned} \tag{3.2}$$

$$\begin{aligned} &\text{contentLength} \\ &= \min(\#\ \text{of remaning paragraph tokens}, \text{maxTokensForDoc}) \end{aligned} \tag{3.3}$$

If the *doc_stride* parameter is set to a smaller value than the capacity, there will be overlaps between chunks. As seen in Figure 3.22; if the capacity is 10, every chunk has a maximum of 10 tokens. If the *doc_stride* is 5, 5 tokens will be overlapping with previous chunks.

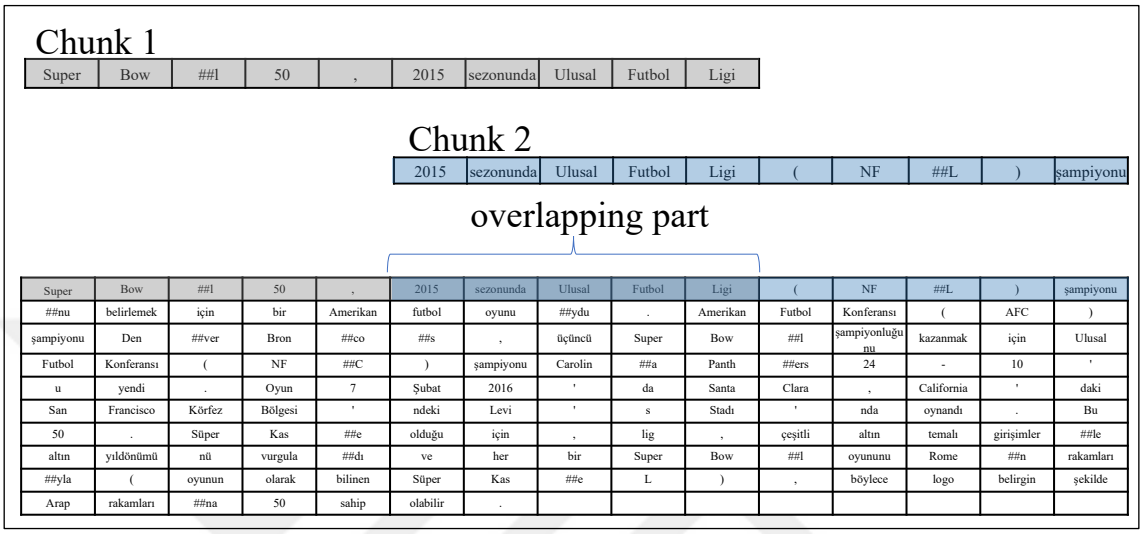

Figure 3.22. Overlapping example.

The final inputs of a fine-tuning task are shown in Figure 3.23. *Run fine-training* task trains the model with transformers, which will be detailed in the 3.2.2 section.

question
Super Bowl 50 hangi ay, gün ve yıl gerçekleşti?

answer
7 Şubat 2016

tokens
[CLS] Super Bow ##l 50 hangi ay , gün ve yıl gerçekleşti ? [SEP] Super Bow ##l 50 , 2015 sezonunda Ulusal Futbol Ligi (NF ##L) şampiyonu ##nu belirlemek için bir Amerikan futbol oyunu ##ydu . Amerikan Futbol Konferansı (AFC) şampiyonu Den ##ver Bron ##co ##s , üçüncü Super Bow ##l şampiyonluğunu kazanmak için Ulusal Futbol Konferansı (NF ##C) şampiyonu Carolin ##a Panth ##ers 24 - 10 ' u yendi . Oyun 7 Şubat 2016 ' da Santa Clara , California ' daki San Francisco Körfez Bölgesi ' ndeki Levi ' s Stadı ' nda oynandı . Bu 50 . Süper Kas ##e olduğu için , lig , çeşitli altın temalı girişimler ##le " altın yıldönümü " nü vurgula ##dı ve her bir Super Bow ##l oyununu Rome ##n rakamları ##yla (oyunun " olarak bilinen " Süper Kas ##e L ") , böylece logo belirgin şekilde Arap rakamları ##na 50 sahip olabilir . [SEP]

input_ids
3 2488 9580 76 618 2544 699 30451 351 6 121 3650 30371 4 2488 9580 76 618 30451 366 408 772 1001 604 30517
25497 597 30675 1692 835 5601 17 7 316 455 1471 3570 30475 316 1001 5973 30517 17472 30675 1692 4111 1245
10731 1144 30 30451 594 2488 9580 76 5044 7967 17 772 1001 5973 30517 25497 501 30675 1692 14994 13 28970
1452 409 30531 138 30357 375 4770 30475 2281 147 271 418 30357 8 4019 19015 30451 13261 30357 165 888 5837
15207 1373 30357 1928 10799 30357 459 17581 30357 109 5462 30475 22 618 30475 1076 3376 14 77 17 30451 1294
30451 255 1026 20059 19009 81 30671 1026 15140 30671 1792 8896 140 6 120 7 2488 9580 76 27092 14217 9 14821
111 30517 3864 30671 11 433 30671 1076 3376 14 711 30671 30675 30451 1406 20320 3203 167 1092 14821 93 618
190 837 30475 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

input_mask
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

segment_ids
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

start position
78

end position
80

Figure 3.23. Fine-tuning input example.

3.2.2. Attention & Transformers

Attention is a method that has given the most impressive results in the deep learning world in recent years, and its popularity is increasing day by day. Usage areas range from image recognition to natural language processing. The most successful practice of the attention is the machine translation problem. Attention aims to extract a summary of the input sequence

that is given to the Encoder architecture and reinterpret this summary in the target Decoder architecture. The most significant disadvantage of the fixed-length content vectors used in the traditional Encoder & Decoder architectures is that if the input sequence is long, the mechanism begins to forget some parts of the context after a while. Attention tries to solve this problem. The attention focus on finding the most critical parts of the input while summarizing the sequence, and the name 'Attention' comes from 'we pay more attention' quote.

Encoder, Decoder and Attention are the essence of Transformer architecture. For machine translation (Figure 3.24), the system converts a text from one language (e.g., English) to another (e.g., Turkish). The Encoder takes the input sequence in English and converts into an n-dimensional vector. The Decoder receives this vector and creates an output sequence, which is the output in Turkish. The n-dimensional vector is like an imaginary language, which both Encoder and Decoder know very well.

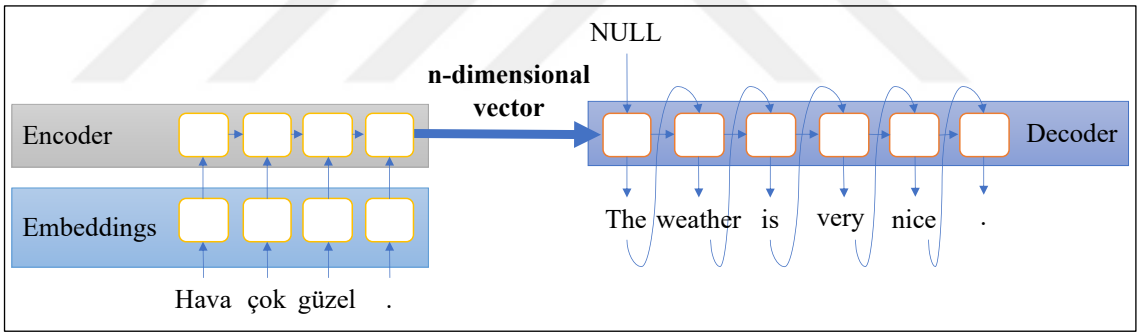

Figure 3.24. Encoder & Decoder architecture.

Attention detects the relevant parts of the given input sequence. In the example of *'She is eating a green apple.',* there is a high attention between *eating* and *apple* but low attention between *eating* and *red* (Figure 3.25). In an attention-based Encoder & Decoder architecture, there are weight matrices, which keeps the semantical relation densities of the words (Figure 3.26). By highlighting the significant parts of the sequence, these weight matrices improves Decoder's performance.

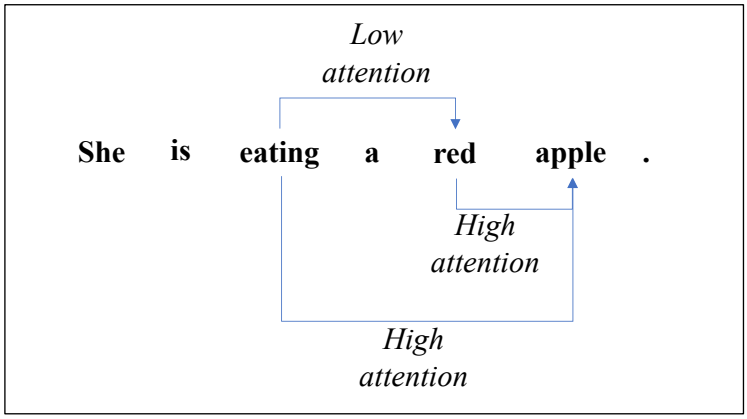

Figure 3.25. Low and high attentions in an example.

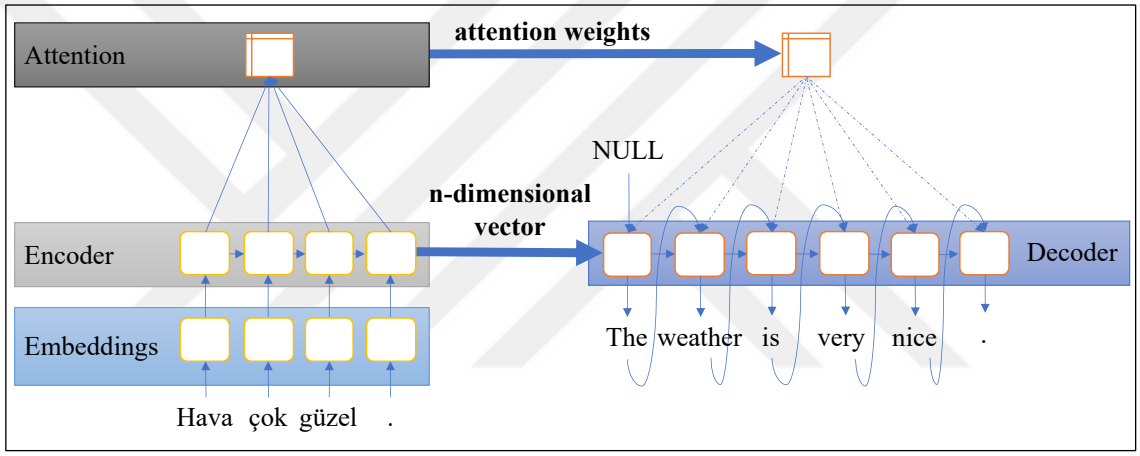

Figure 3.26. Attention-based. Encoder & Decoder architecture for machine translation.

The steps of a machine translation using attention-based Encoder & Decoder architecture is:

Step 1: The encoder gets the input sequence and generates hidden state vector; [h_1, h_2, h_3, h_4…]

Step 2: Feed Forward Neural Network generates a score vector of the hidden vector using the previous hidden state of the decoder; [s_1, s_2, s_3, s_4…]

Step 3: Softmax Layer generates the attention weights; [e_1, e_2, e_3, e_4…]

a) All the weights lie between 0 and 1, i.e., $0 \leq e_1, e_2, e_3, e_4, e_5 \leq 1$

b) All the weights sum to 1, i.e., $e_1+e_2+e_3+e_4+e_5 = 1$

Step 4: Context vector is calculated as summarizing the multiplication of h and e vectors; CV.

$$CV = e_1 * h_1 + e_2 * h_2 + e_3 * h_3 + e_4 * h_4 + e_5 * h_5 \quad (3.4)$$

Step 5: Context vector is concatenated with the output of the decoder previous steps; CV + <*NULL*> for the first step, CV + <*NULL The*> for the second step, CV + <*NULL The weather*> for the third step.

Step 6: The decoder generates the output text and also the next hidden state; *The* for the first step, *The weather* for the second step.

In a traditional RNN Seq2Seq Encoder&Decoder model, everything is crammed into a single context final state as in Figure 3.27. In an attention-based Encoder&Decoder model, there are multiple hidden states which look at everything, as in Figure 3.28.

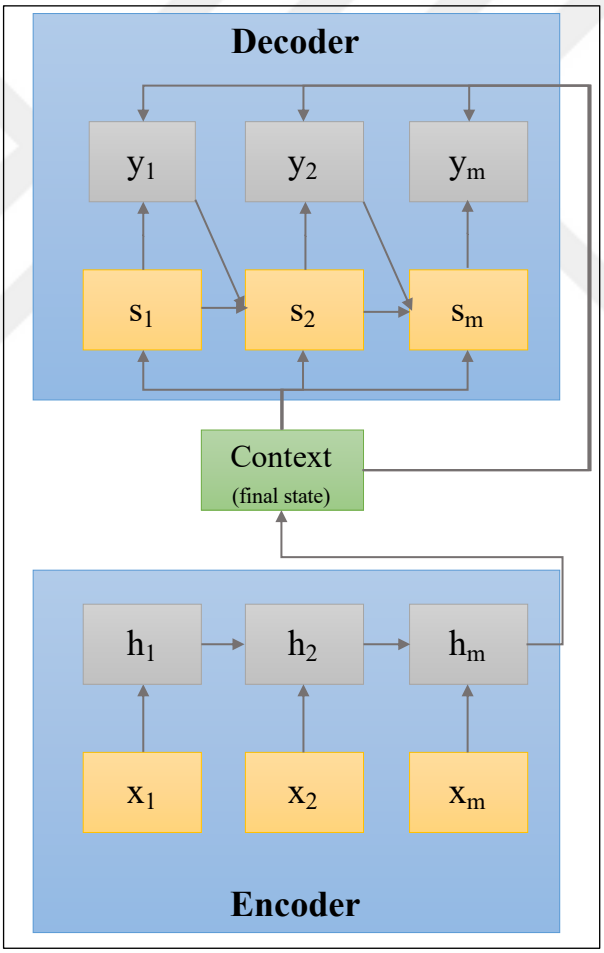

Figure 3.27. RNN Seq2Seq Encoder&Decoder example.

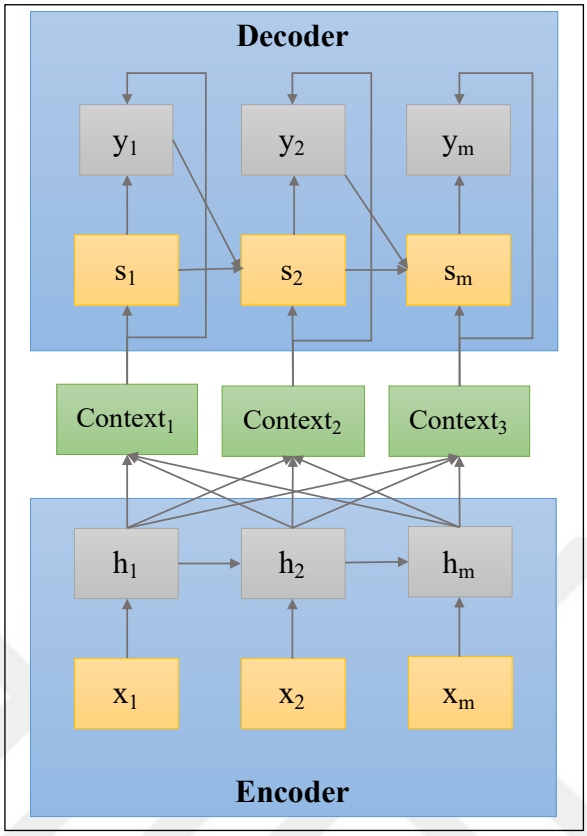

Figure 3.28. Attention-based Encoder&Decoder example.

There are three different attention types: *Hard/Local Attention* focuses on a part of input like a patch of an image; *Soft/Global Attention* focuses on entire input; *Self/Intra Attention* focuses on different positions of the input. Modern NLP architectures use a self-attention mechanism.

Instead of using recurrent network structures, Transformers use the attention mechanism. The reason is the disadvantages of existing recurrent networks, such as depending on the time series, sequential process over previous or next words. Transformer networks support parallelism because of no recurrence. Transformers facilitate long-range dependencies, and there is no gradient vanishing or explosion. There are also direct connections between components, called *residual connection*. As a result, fewer steps are enough for training.

Transformer networks are a stack of encoder and decoders which have self-attention mechanisms and Feed-Forward networks. Figure 3.30 shows an example of an encoder stack that has 12 layers and a maximum sequence length set as 512. The embedding process is only done at the bottom-most step of the encoder. The embedding vector size is a

hyperparameter, which we can fine-tune; for the examples, it is set to 512. Every encoder has two layers; a self-attention and a Feed-Forward Network. The input sequence of the encoder includes the words with their positions. In the self-attention layer, there are dependencies between the words. These dependencies are not present in the Feed Forward Network layer, and some operations can be done in parallel in the network.

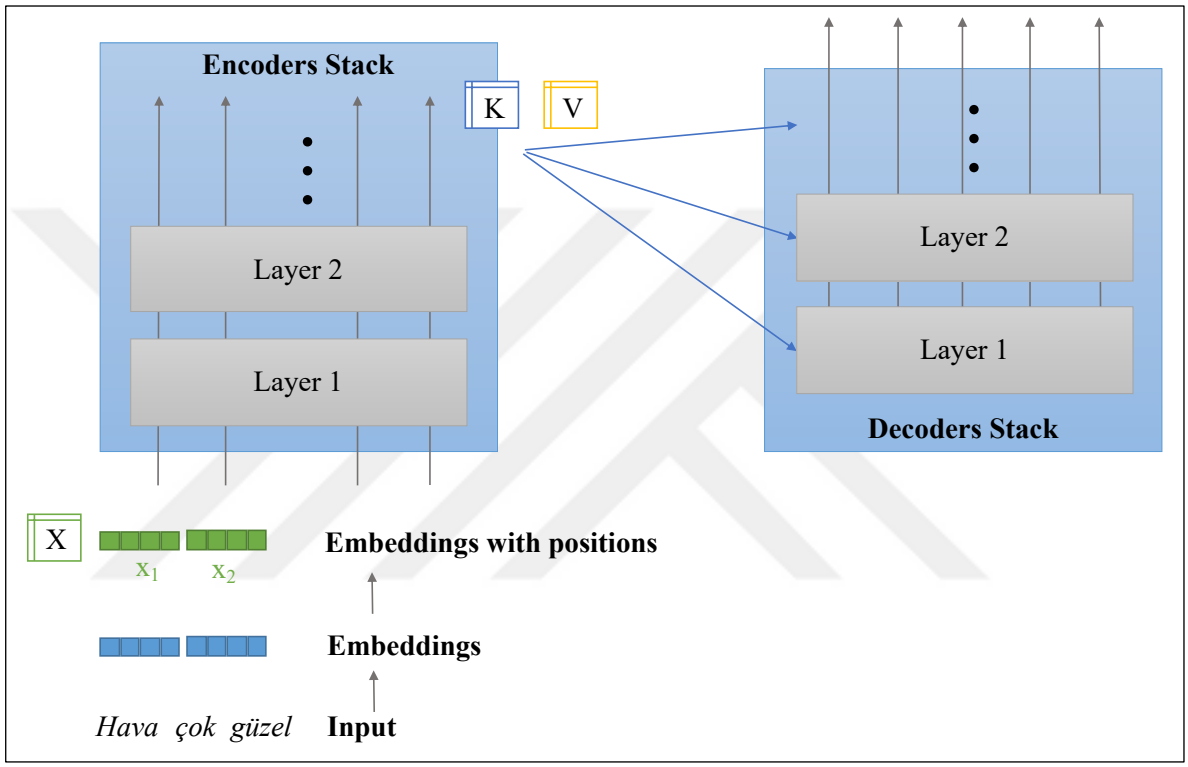

Figure 3.29. Inputs of a Transformer Network.

After generating and summing the tokens and positions embedding vectors of the input, the final vector is the input of the stack's bottom encoder, the green X_n vectors in the Figure 3.29. There is a chain approach; the output of the first encoder is the input of the second encoder, the output of the second encoder is the input of the third encoder, and the output of the last encoder is the input of all the decoders. The linear layer at the top of the decoder stack, converts the output of the decoders to logits vector that has the same size as the vocabulary file. After the Softmax layer converts the scores into probabilities, the system choses the element that has the highest probability.

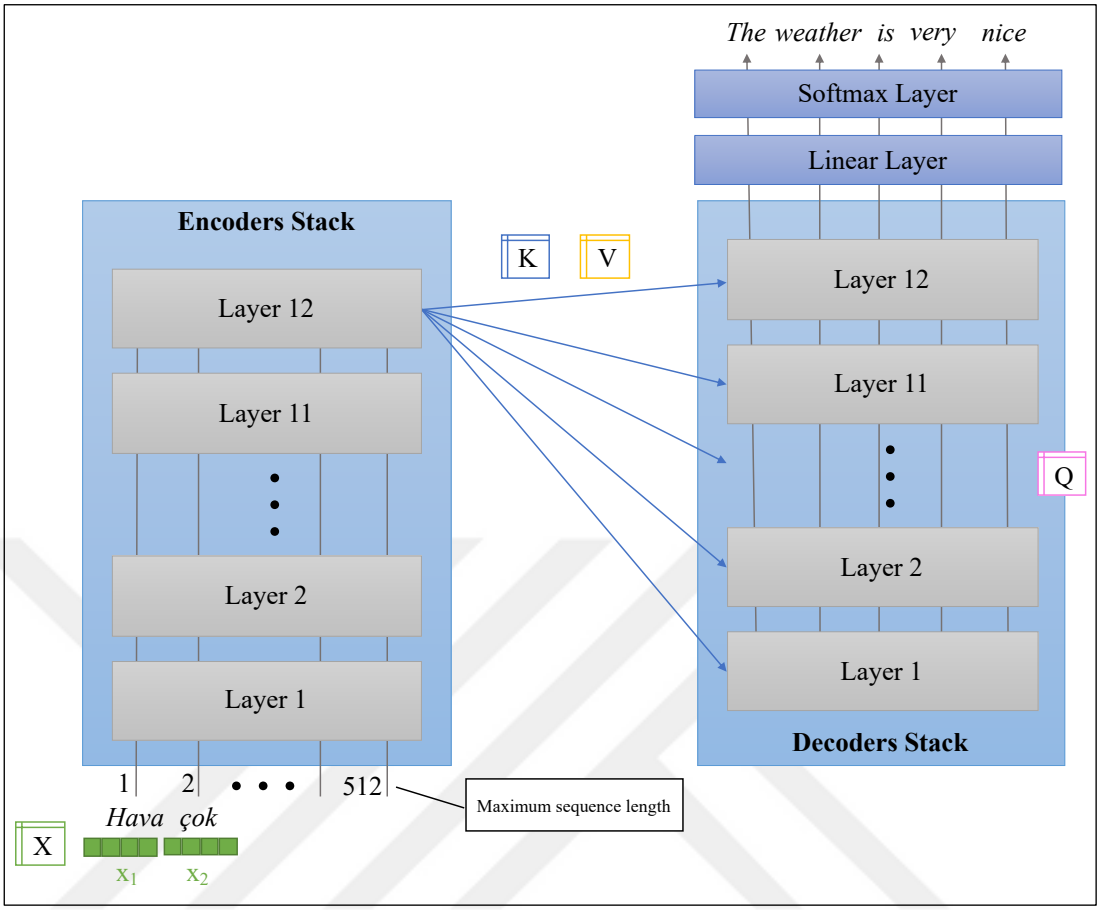

Figure 3.30. BERT encoder & decoder stacks for a machine translation example.

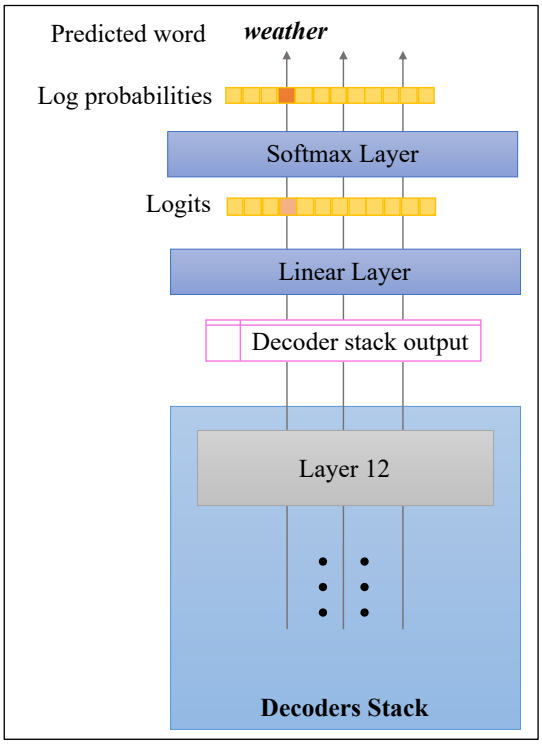

Figure 3.31. Decoder layers of a transformer network.

The subcomponents of an encoder and a decoder are similar (Figure 3.32). With *x* input vectors, self-attention computes *z* vectors. After the z vector is computed, a 10 percent dropout is applied. In order to prevent early summarization of the input, *x* vector is added to the *z* vector with residual connections shown as dotted lines in the figure. Then, to increase the stability, the mechanism normalizes the output. For reducing the number of feature maps, there is a Feed-Forward Network (FFN) after each self-attention layer in both encoders and decoders.

A decoder has a different layer from Encoder, named *Encoder-Decoder Attention*. This layer gets the outputs of the encoder (K and V matrices) and combines with Decoder self-attention results (Q matrices). This operation inhibits Decoder to look further positions in the sequence when generating the output (Table 3.3). If that is an English to Turkish machine translation example; the task of Decoder is generating the next word. In every timestamp t, the Decoder gets the output of the Softmax layer of timestamp t-1. The Softmax layer at the top of the Decoder generates the next token, like a typical Encoder & Decoder architecture.

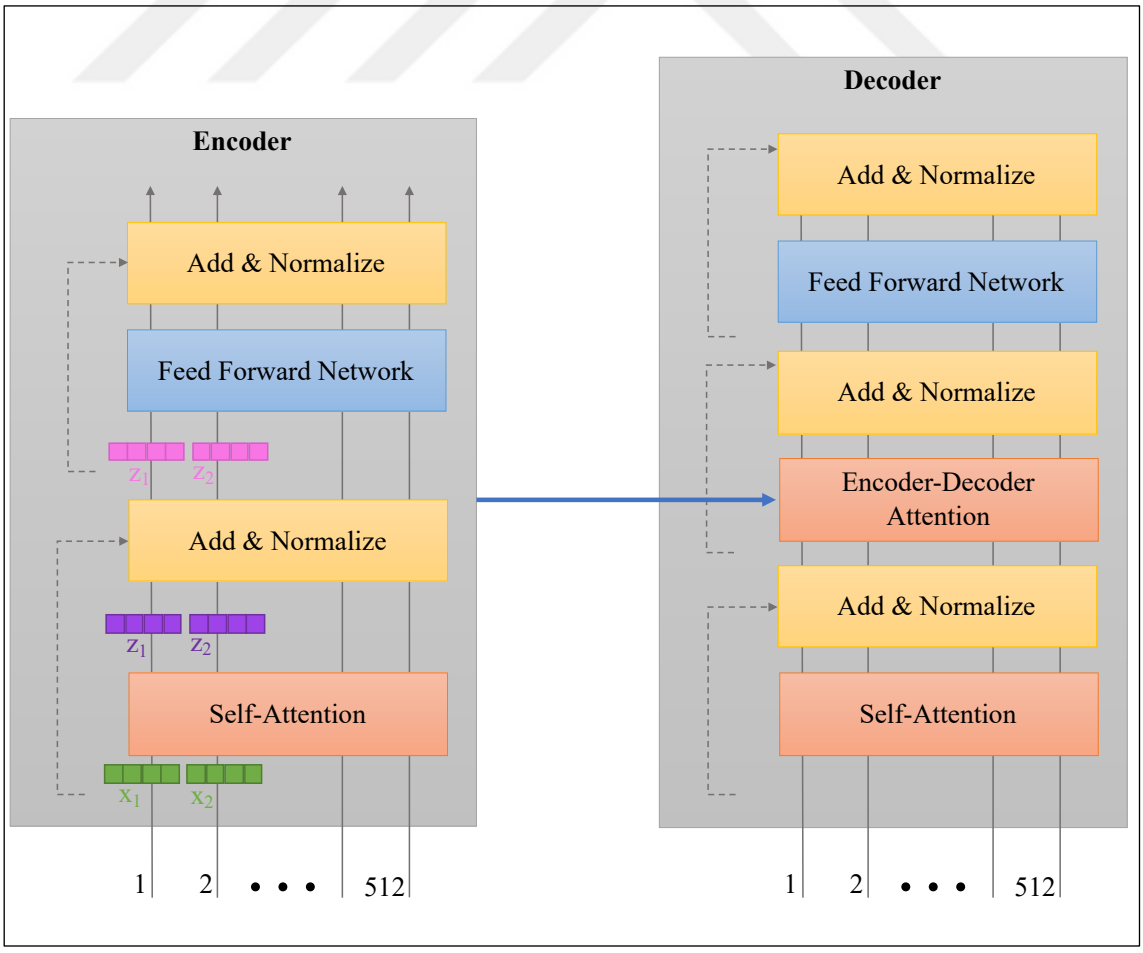

Figure 3.32. Post operations after self-attention.

Table 3.3. Machine translation decoder example.

Step	Input of the decoder			Output of the decoder		
1	<CLS>			Düşünen		
2	<CLS>	Düşünen		Düşünen	makineler	
3	<CLS>	Düşünen	makineler	Düşünen	makineler	…

For each token in the input sequence, three vectors (q, k, v) are created by multiplexing three weight matrices (W^Q, W^K, W^V) with the embedding vector (x_1, x_2, …, x_n).

- **Query vector (q):** The vector of the source token which is paid attention.
- **Key vector (k):** The vector of the target token with which the attention will be calculated.
- **Value vector (v):** Result vector.

The calculation steps of the Self-attention are based on the equation 3.5.

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.5}$$

Step 1: System multiplies embeddings vectors (x_1, x_2, …, x_n) with W^Q, W^K and W^V matrices to calculate Query (q_1, q_2, …, q_n), Key (k_1, k_2, …, k_n) and Value (v_1, v_2, …, v_n) vectors. Training process optimizes W^Q, W^K and W^V matrices like NN hyperparameter weights. Figure 3.33 shows the details. The vector dimensionalities are configurable. In the example, the input and output vectors have dimensionality of 512; q, k, and v vectors have dimensionality of 64.

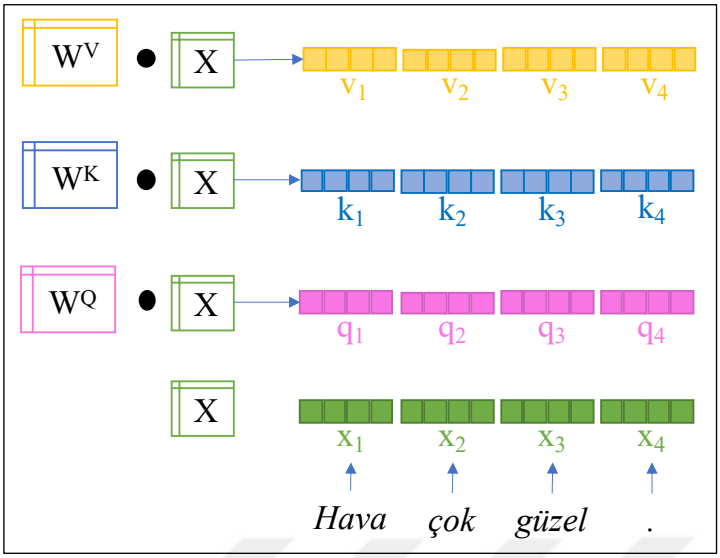

Figure 3.33. Self-attention steps (1).

Step 2: For each source word (i), system calculates scores of each target word (j) in the input sequence. The formula of the score is; dot product of q_i and k_j vectors of each source word i and each target word j.

$$Score = QK^T \tag{3.6}$$

For the first word '*Thinking'*, the mechanism calculates scores for all words. For the second word '*Machines'*, the mechanism again calculates scores for all words (Figure 3.34).

Step 3: For normalization, system divides the scores by the square root of vector dimensionality $d_k = 64$ (Figure 3.34).

$$Divide\ by\ 8\ = \frac{QK^T}{\sqrt{d_k}} \tag{3.7}$$

Step 4: System applies the Softmax function. Softmax result shows the effect of the target word to the selected position (Figure 3.34). In the example, to the first word, the effect of *'Thinking'* is 0,88, but *'Machines'* is 0,12. The Softmax results in the figure are only for the first word, and the system repeats same operations for other words.

$$Softmax = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) \tag{3.8}$$

Step 5: For eliminating unnecessary words, system multiplies the Softmax result with Value vectors (Figure 3.34). As a result of the Softmax score of the first-word is higher than the second word, its value vector is more visible in order to indicate its importance factor in the figure.

$$Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.9}$$

Step 6: System sums finals vectors for attention (Figure 3.34). In z_1, v_1 has more density than v_2.

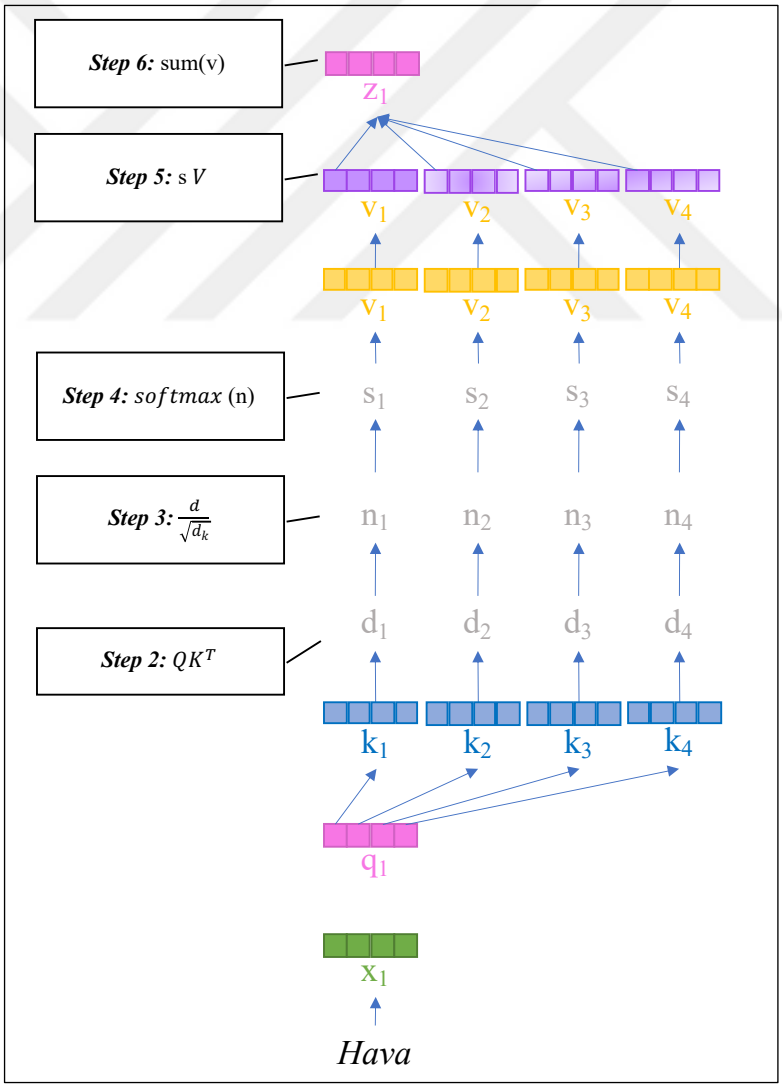

Figure 3.34. Self-attention steps (2-6) for '*Thinking*'.

BERT uses a multi-layered bidirectional Transformer encoder-based architecture. $BERT_{BASE}$ configuration has 12 Transformers layers with 12 attention-heads. Let's assume that, the input is '*Yeşil elmayı seviyorum. Daha lezzetliler. - I like green apples. They are more delicious.*'. Figure 3.35 shows the output of Transformer-layer = 0 and Attention-head = 0. The highlighted connection indicates a strong relationship between tokens: *yeşil (green)* and *elma (apple)*. The colored box bar represents the 12 attention heads. The visualization changes when the selected token or attention head is changed.

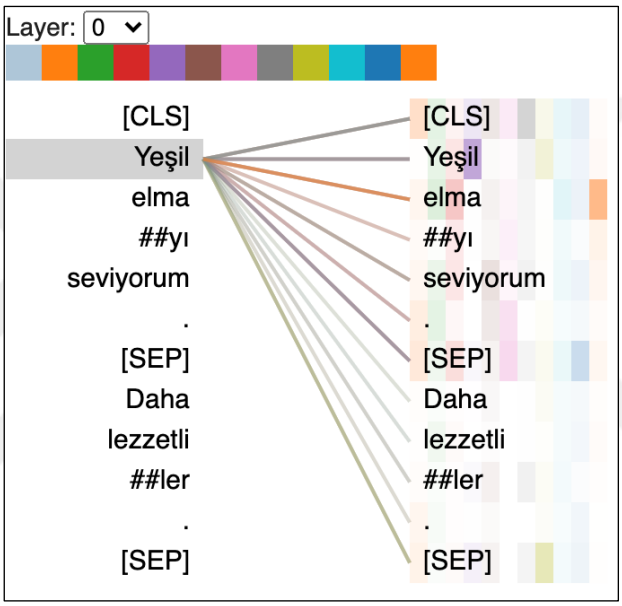

Figure 3.35. Transformer-layer = 0 and Attention-head = 0 output.

In the lower layers of the transformers, there connections are usually with previous or next words. In the higher-level layers, the semantic relations become more visible. But, there isn't a similar hierarchy between the attention heads in the same layer. Attention heads analyze the same input from different perspectives within the scope of the given layer. Figure 3.36 shows Transformer-layer = 4 and Attention-head = 0 output: there is a connection between *yeşil (green)* and *lezzetli (delicious)*. Figure 3.37 shows Transformer-layer = 11 and Attention-head = 6 output: there is a connection between *yeşil (green)* and *seviyorum (like)*.

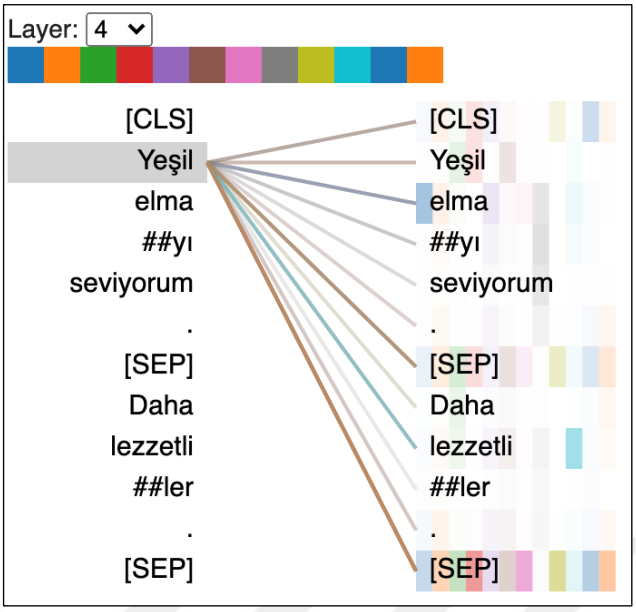

Figure 3.36. Transformer-layer = 4 and Attention-head = 0 output example.

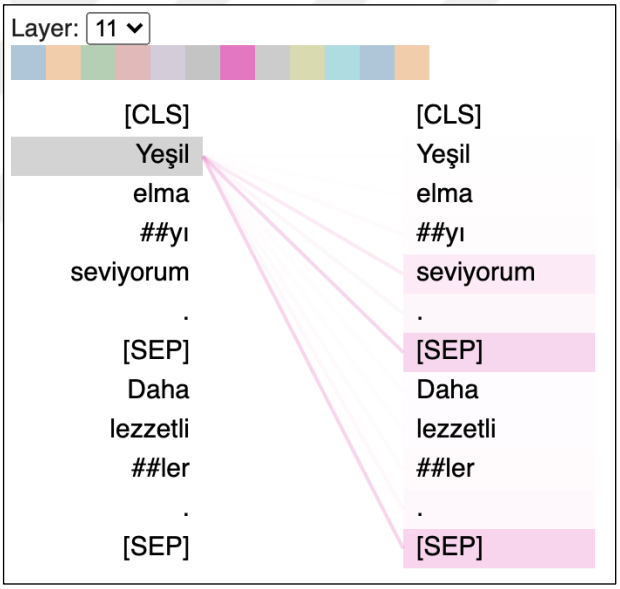

Figure 3.37. Transformer-layer = 11 and Attention-head = 6 output example.

When using multi-headed attention, all weight matrices and vectors of the single-headed attention have a new dimension representing the attention-heads:

- The q, k, and v vectors are converted to matrices.
- Q, K, and V weight matrices are converted to a set of matrices that have a matrix for every attention-head.

Every word (i) has a z_i vector, and the final output is a matrix. When multi-headed attention is used, there are *# of attention heads * z* matrices. Different from single-headed attention,

for generating the final Z matrix output, the mechanism concatenates all z matrices and multiples with the W^0 weight matrix.

Firstly, there were two pre-trained models published by Google: $BERT_{BASE}$ and $BERT_{LARGE}$ (Table 3.4). Later, their numbers are increased to 6 (Table 3.5). $BERT_{BASE}$ has a similar model size as OpenAI GPT. $BERT_{BASE}$ and $BERT_{LARGE}$ use the same architecture and only their parameter counts are different.

Table 3.4. BERT pre-trained models (earlier).

Model	**Layers**	**Hidden layer nodes**	**Attention heads**	**Total parameters**
$BERT_{Base}$	12	768	12	110M
$BERT_{Large}$	24	1024	16	340M

Table 3.5. BERT pre-trained models (now).

Counts	**Hidden layer nodes=128**	**Hidden layer nodes=256**	**Hidden layer nodes=512**	**Hidden layer nodes=768**
Layers=2	$BERT_{Tiny}$			
Layers=4		$BERT_{Mini}$	$BERT_{Small}$	
Layers=6				
Layers=8			$BERT_{Medium}$	
Layers=10				
Layers=12				$BERT_{Base}$

4. ANALYSIS AND DESIGN

This section gives information about the approaches that we've used in the study and the design of the system. Section 4.1 summarizes BERT, and Section 4.2 describes how BERT solves some morphological problems of Turkish. Section 4.3 presents the data sets used in the study. Section 4.4 defines the training procedures, and Section 4.5 lists the training parameters. Last, Section 4.6 gives some information about our evaluation metrics.

4.1. BERT

Bidirectional Encoder Representations from Transformers (BERT) was proposed by Google in 2018. First, system generates a context-sensitive language model, called *"pre-training task"*. Then, the system can perform a series of NLP tasks, called *"fine-tuning task"*. Figure 4.1 presents the architecture of BERT. In this thesis, we trained a BERT model that has 110 million parameters, 12 transformer layers with 12 attention heads.

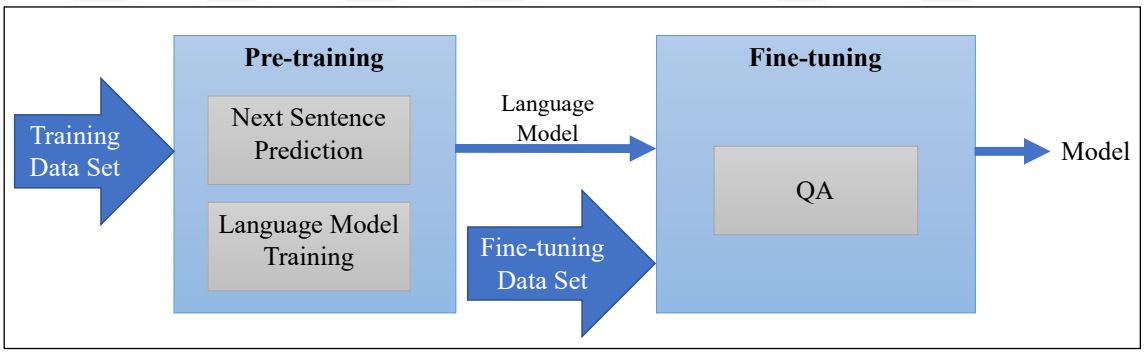

Figure 4.1. BERT Architecture.

Pre-training: BERT is an unsupervised deep learning method that builds bidirectional transformer-based language models based on Contextualized Word Embeddings (CWE). A language model predicts the probability of a word in a given context. After tokenization of the input text using the WordPiece algorithm, BERT masks some tokens randomly. The recommendation is masking 15 percent of the content. Then, the Transformer network updates the word representation weight matrices during the prediction of each masked token with a Softmax classifier. The Softmax loss function only counts the predictions of the masked values and ignores the unmasked words' predictions. This training operation based

on masking is named as Masked Language Model (MLM) training. The real success of BERT comes from the MLM start-of-art idea. In parallel to MLM training, BERT determines the relationships between the sentences using sentence labels in the training data. This is a binarized task called Next Sentence Prediction (NSP). Training of MLM and NSP is called *pre-training*, and the output model is operable with a simple *fine-tuning* for many different NLP tasks.

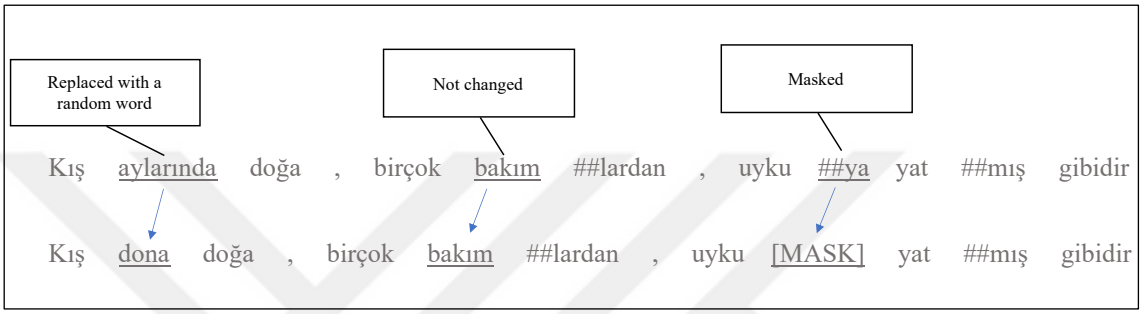

Figure 4.2. Token replacement examples.

Instead of always masking, BERT replaces the mask candidate word; 80 percent of the time with [MASK], 10 percent of the time with a random subword picked from vocabulary, and the remaining 10 percent of the time lefts as unchanged. This mitigation prevents mismatch between pre-training and fine-tuning tasks; because [MASK] only performs in pre-training.

The Transformer is a stack of encoders that have multi-head self-attentions, therefore unlike traditional sequential approaches that only care unidirectional as the previous or next words, the self-attention evaluates the input as a whole and discovers which parts of the input are more relevant with the masked word. Because of multi-heads, every attention head separately computes attention with different weight matrices and then concatenates the results together.

Traditional word embeddings, also called shallow representations, only focus on lower layers of the model; hence they cannot capture higher-level information such as long-term dependencies, negation, or anaphora from scratch data. In BERT, layers are in a chained stack logic; the input of the lowest layer is directly the embedding, and the output of each layer is the input of the next layer. On language model training, the transformer layers are trained all together with massive data; consequently, the lower-levels are more concerned

with the syntactic tasks such as connections of words, and the higher-levels are focused on more semantic relations of the context. In transformers layers, for preventing early summarization of the data, there are residual connections from inputs to outputs.

Fine-tuning: When the language model is generated, BERT can do a particular NLP task by using a supplementary data set. During the fine-tuning procedure, the weighs of the BERT network are slightly modified.

Question Answering is one of the down-stream tasks of fine-tuning. Like pre-training, BERT again uses transformers for training. In the preparation of MLM, the input is a tokenized text which has a certain amount of masked subwords, and the output is the prediction of the masked words. In QA training, the input is a tokenized sequence, which is the union of question and paragraph, and the output is the prediction of the answer. In fine-tuning, this time, BERT optimizes the weights to the appropriate values for finding the most trustworthy answers to the given questions and paragraphs.

In this study, because of its state-of-art solutions and highly accurate results in English and Chinese benchmarks, we decided to train a Turkish language model then fine-tune it for a question answering system using BERT and evaluate with a banking sector QA data set, as seen in Figure 4.1. We have trained a model that can solve the MRQA problem and give high success rates for Turkish, by taking the examples of the studies made for English so far. BERT, which is the most used in current MRQA studies and a reference model in similar studies in other languages, was taken as the basis of this study. The datasets will be detailed in the 4.3 section.

4.2. WORD SENTENCE DISAMBIGUATION (WSD) IN BERT

In Turkish, many of the difficulties arise from being an agglutinative language and having a rich but complex morphology, including a comprehensive set of possible suffix tags and diversity of constituent orders in inverted sentences. Part-of-speech (POS) tagging is a crucial preprocess for most NLP tasks, which is the process of analyzing the text morphologically and dividing it into parts. Compared to Turkish, it is easier for morphologically simpler and limited languages such as English. Morphological

disambiguation is the most challenging Turkish language problem that is essential for NLP applications like Word Sense Disambiguation (WSD), syntactic parsing or spelling correction. Some rule-based methods partially solve morphological disambiguation for Turkish, such as Zemberek. *'Çekoslovakyalılaştıramadıklarımızdan mısınız?'* is one of the longest word in Turkish and Zemberek output is in Figure 4.3.

```
çekoslovakyalılaştıramadıklarımızdan  [Çekoslovakyalı:Noun,Prop]
        çekoslovakyalı:Noun+A3sg|laş:Become
        →Verb|tır:Caus
        →Verb+ama:Unable|dık:PastPart
        →Noun+lar:A3pl+ımız:P1pl+dan:Abl
mısınız [mı:Ques]
        mı:Ques+Pres+sınız:A2pl
?       [?:Punc]
        ?:Punc
```

Figure 4.3. Morphological Disambiguation result of *'Çekoslovakyalılaştıramadıklarımızdan mısınız?'* using Zemberek.

Word embeddings have become famous, as they enable the input text to be converted into a numerical form and easily inserted into mathematical operations for neural networks. After major innovations in NLP, state-of-art solutions such as BERT provide Contextualized Word Embeddings (CWE), which includes semantic vector representations of the words depending on their context. CWE captures the polysemy, in which although both are the same word, the embedding vector differentiates in different contexts. The word embeddings of the *'bank'* should be different for a finance office and a seat in the park. BERT trains a language model, which is simply a set of CWEs.

For some NLP tasks such as question answering, machine translation or text classification, Word Sense Disambiguation (WSD) is sufficient rather than fully solving the morphological disambiguation problem. In modern NLP, Contextualized Word Embeddings solve the Word Sense Disambiguation problem [29]. In considering all these stated circumstances; BERT is a solution for MRQA by overcoming Word Sense Disambiguation problem using Contextualized Word Embeddings. For generating a BERT language model that includes CWEs, the following approaches are combined;

- A subword-based embedding system: WordPiece.

- Masked Language Model (MLM) & Next Sentence Prediction (NSP) trainings.
- Bidirectional Transformers and Self-Attention.

A subword-based embedding system: WordPiece is a new generation word segmentation algorithm that builds a language's subwords, which can be a word, a syllable, or a single character. The algorithm divides the text into characters and then systematically brings them together in order to create combinations, which are the candidate subwords. The candidate subwords are applied one-by-one to the training data, followed by calculating the likelihood of the system. Which of them most increases the likelihood is put into the vocabulary. Vocabulary size is a parameter, and the subword selection process is repeated until reaching the given configuration. Although WordPiece is not a morphologically perfect POS tagger for Turkish, it is satisfactory for the tokenization of the input sequences. Figure 4.3 shows the tokenization result of *'Çekoslovakyalılaştıramadıklarımızdan mısınız?'* using WordPiece algorithm. In the figure, ## indicates the subword is a suffix. For covering the unknown words that the tokenizer cannot handle, there is also an Out-Of-Vocabulary (OOV) subword.

Çek	##os	##lovak	##yalı	##laştır	##amadı	##k	##larımızdan
mısınız	?						

Figure 4.4. WordPiece Tokenizer result of 'Çekoslovakyalılaştıramadıklarımızdan mısınız?'.

Masked Language Model (MLM) & Next Sentence Prediction (NSP) trainings: After tokenization of the input text with the WordPiece algorithm, BERT randomly masks some tokens, followed by predicting the original value of these masked tokens. While the prediction of each masked token, the transformers network optimizes the word representation weight matrices with a Softmax classifier that compares the predicted and the original tokens. Word representation weight matrices are the base of the Masked Language Model (MLM). The word representation weight matrices are Contextualized Word Embeddings (CWE), which have semantical dynamic information for the language and support polysemy. In parallel to MLM training, BERT determines the relationships between the sentences using sentence labels in the training data. The Next Sentence Prediction (NSP) is a binarized task and uses a sigmoid classifier.

Bidirectional Transformers and Self-Attention: The transformers network is a stack of multiple layers. There is a logical hierarchy between the layers; each layer's output is the input of the next layer. The lower-levels are more concerned with the syntactic tasks such as connections of words, and the higher-levels are focused on more semantic relations of the context. Each transformer layer has encoders based on multi-head self-attentions. Unlike traditional sequential approaches that only care unidirectional as the previous or next words, the self-attention evaluates the input as a whole and discovers which parts of the input sequence are more relevant with the masked word. Using multi-heads, every attention head separately computes attention with different weight matrices and then concatenates them together for a larger perspective.

Figure 4.5 shows the transformers architecture of BERT pre-training task. The inputs are two masked sentences labeled as “B is the next sentence of A” or “not”. The output of the system is a masked language model that also has information about sentence relations.

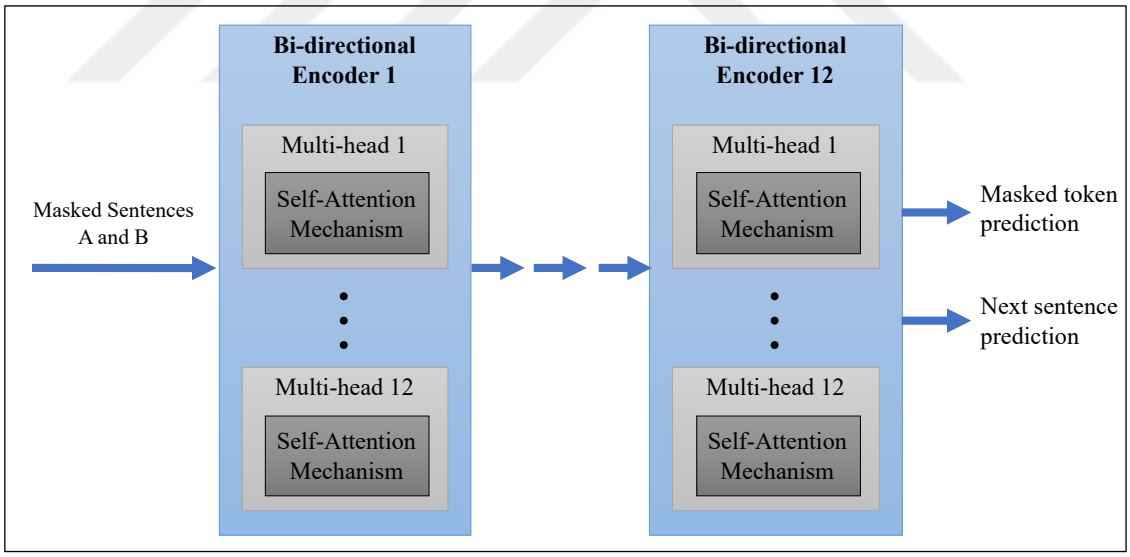

Figure 4.5. BERT pre-training architecture.

4.3. DATA SETS

The concerning problem of building a question answering system with deep learning methods is the preparation and adaptation of the training data to the chosen training procedure. Especially for NLP tasks, the quality of the training data has a significant impact

on the system's outcome. Regarding this matter, one of the most time-consuming parts of this thesis study was the generation and customization of the training data. The training data can be defined under two main headings: the corpus set used to train the language model, and the data sets used to fine-tune the system for the QA task.

Language Model Data Sets: Table 4.1 represents the corpus set used in building the language model. Wikipedia Corpus is the dump of Turkish Wikipedia articles published on Wikimedia publicly [30]. News Corpus is a vast collection of online Turkish newspapers. Economy Corpus is the smallest one specific to the banking domain, collected from several economy blog websites. All corpora are in Turkish.

Table 4.1. Language model data sets.

Name	**Size**	**Content**	**Information**
Wikipedia Corpus (Tr)	456,5 MB	~4,5 M sentences	Turkish Wikipedia dump 922335 pages (dump date: 08/2019)
News Corpus (Tr)	2,5 GB	~20 M sentences	News articles collection in Turkish
Economy Corpus(Tr)	15,5 MB	~270K sentences	Turkish economy blogs from Web

Fine-tuning Data Sets: Table 4.2 represents the data sets used in fine-tuning the model for question answering. The Stanford Question Answer Data Set (SQuAD), published in 2016, is a fundamental resource for MRQA researches. The SQuAD is an array of paragraphs and their question & answer pairs. It is available in 2 versions: in version 1.1, the answers to each question are apparent, and in version 2.0, some questions have no answer. In this study, SQuAD v1.1 is the main format used, and we converted all data sets to this form as described in 5.2 section.

SQuAD (Tr) is the Turkish translated from of original SQuAD, which is in English. The SQuAD data set is generated by Stanford University, which contains more than 100,000 real question-answer pairs created by humans over 536 Wikipedia passages. NewsQA (Tr) is the Turkish translated from of original NewsQA, which is in English. NewsQA data set is generated by Microsoft Research, which contains 120K question-answer pairs created by humans over CNN news articles. Banking Sector QA (Tr) is the data set created by a private

Turkish bank's employers, supervised by the authors of this study. The documents used in the data generation are the tutorials and legislation archives of the bank.

Table 4.2. Fine-tuning data sets.

Name	Size	Content	Information
SQuAD (Tr)	24,42 MB	490 documents 20963 paragraphs 45872 questions 56117 answers	Q&A from paragraphs from Wikipedia articles. (Machine translation from English to Turkish)
NewsQA (Tr)	19,66 MB	8379 documents 8343 paragraphs 21270 questions 21270 answers	Q&A from articles from CNN news. (Machine translation from English to Turkish)
Banking Sector QA (Tr)	5 MB	679 documents 1637 paragraphs 17708 questions 17708 answers	Q&A from documents from the banking sector. (in Turkish)

SQuAD and NewsQA datasets are machine translated datasets, and translation process harms some of the contexts as expected. The cleanup procedure of the dirty data is described in the 5.1 section. Table 4.3 compares the numbers of the elements before and after the machine translation operations; nearly half of the data had been lost. In the table, the train set indicates the data used for training the model, and the development set shows the data used for validating the model.

Table 4.3. Translation details of Fine-tuning data sets.

Number of	Documents	Paragraphs	Questions	Answers
SQuAD (En, Tr)				
English (Original)				
Train Set	442	18896	87599	87599
Development Set	48	2067	10570	34726
Turkish (Translated)				
Train Set	442	18896	40014	40014
Development Set	48	2067	5858	16103
NewsQA (En, Tr)				
English (Original)				
Train Set	11428	11428	74160	74160
Development Set	638	638	4212	4212
Turkish (Translated)				
Train Set	7917	7917	20147	20147

Development Set	426	426	1123	1123

Content lengths are critical on the strategy of selecting the training parameters of fine-tuning tasks. The maximum and average lengths of the paragraphs, questions, and answers are detailed in Table 4.4 and Table 4.5. The counting procedure is described in section 5.4. Although the measurement unit of paragraphs and questions is token, the answer length are character based. An extended open-domain data set is generated from the union of SQuAD and NewsQA data sets, named *SQuAD (Tr) + NewsQA (Tr)*. The process is described in section 5.3.

Table 4.4. Average content lengths of Fine-tuning data sets.

Average lengths	**Paragraph (token)**	**Question (token)**	**Answer (character)**
SQuAD (Tr)			
Train Set	145,45	11,64	14,38
Development Set	152,90	11,74	13,48
NewsQA (Tr)			
Train Set	372,17	7,67	14,06
Development Set	364,92	7,61	13,34
SQuAD (Tr) + NewsQA (Tr)			
Train Set	212,39	10,31	14,27
Development Set	189,13	11,08	13,47
Banking Sector QA			
Train Set	186,87	11,30	36,83
Test Set	186,98	11,30	36,71

Table 4.5. Maximum content lengths of Fine-tuning data sets.

Maximum lengths	**Paragraph (token)**	**Question (token)**	**Answer (character)**
SQuAD (Tr)			
Train Set	845	46	169
Development Set	783	36	149
NewsQA (Tr)			
Train Set	1074	56	317
Development Set	942	20	162
SQuAD (Tr) + NewsQA (Tr)			
Train Set	1074	56	317
Development Set	942	36	162

Banking Sector QA			
Train Set	784	49	256
Test Set	784	49	256

4.4. TRAINING PROCEDURE

The training procedure can be defined under three main headings: the language model generation for Turkish, preprocessing of fine-tuning data sets, and fine-tuning the model for the question answering task.

Language Model Generation: For training a language model, a corpus and vocabulary file is required. The corpus list, which is used in the pre-training, is detailed in the Table 4.1. Subwords are built base on the corpora using the WordPiece algorithm. The system puts the subwords to the vocabulary file, which most increased the likelihood of the training. 32000 subwords sized vocabulary file is used, recommended by BERT. WordPiece algorithm built approximately 30000 of these subwords from the corpora. Then, we manually added 2000 items as abbreviations and words that are most commonly used in the economy and banking domains, but not already present in the vocabulary file. BERT tokenize corpus using WordPiece tokenizer. The pre-training input sequence has two parts, which may be consecutive sentences or randomly selected pairs from the documents. Using these sentence pairs and their label, Next Sentence Prediction (NSP) training learns the relations of the sentences as they are pertinent or not. BERT masks some of the tokens of the input. The prediction process of these masked tokens updates some weight vectors called *word representations* that are the building blocks of the language model. Technical details of BERT is in 3.2.1 and 4.1 sections.

Preprocessing of Fine-tuning Data Sets: The data sets, which are used in the fine-tuning, is detailed in the Table 4.2. SQuAD and NewsQA data sets have been converted from English to Turkish by machine translation. Naturally, some data mismatches occurred during the process. Hence, the translated data sets had to be controlled and cleaned. BERT automatically supports the SQuAD data set format. According to the SQuAD, we reformatted NewsQA (Tr) and Banking Sector QA data sets. Also, a new data set is

generated: the extension of SQuAD (Tr) and NewsQA (Tr) data sets. The implementation details of the processes are in Section 5.

Fine-tuning: In this study, fine-tuning has two phases. In the first phase, the system increases its question-answering skills in an open-domain, meaning as general purpose. The second phase is the customization of the model in a closed-domain. In other words, using SQuAD (Tr) and NewsQA (Tr) datasets system firstly trains the language model, which is the output of pre-training. Then, using the Banking Sector QA data set, system fine-tunes the model in accordance with the content or question and answer patterns of the banking industry. The MRQA training steps are illustrated in Figure 4.6.

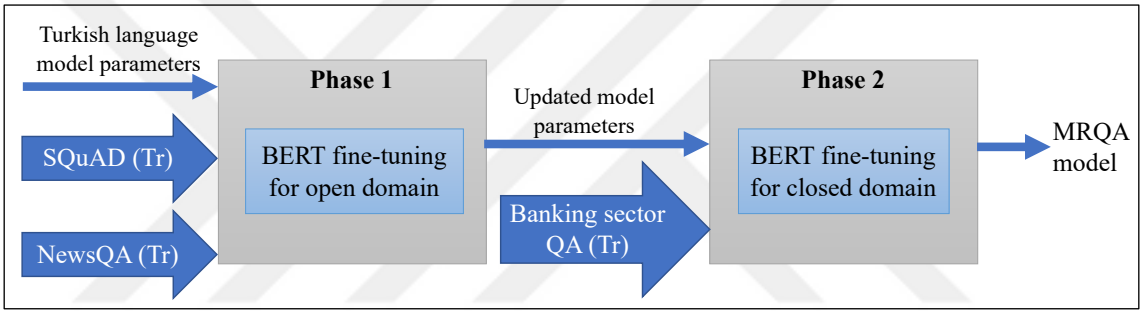

Figure 4.6. Fine-tuning phases.

4.5. TRAINING PARAMETERS

In the pre-training task, the *maximum sequence length* shapes the input vector size of the Transformer network, and high values are needed to learn positional embeddings in long sequences. On calculating the number of word pieces to mask, BERT selects the minimum of *maximum predictions per sequence* and *masked LM probability* multiplied by token size. *Do lower case* and *do whole word mask* parameters belong to tokenization and masking. In the NLP benchmarks, the published *cased* and *whole word masked* language models have higher accuracies, notably for Asian languages such as Chinese or Arabic. The current study is Turkish specific, and vocabulary file is wide enough to already have most of the words. Consequently, *subword* or *whole word masking* tokenizers give similar results.

In the fine-tuning task, the input sequence of the Transformers is the union of the question and the paragraph; the output is the answer. The *maximum query length* is the number of

question tokens located in the input sequence, and BERT ignores the rest if a longer text. In the input sequence, a certain number of tokens left for the paragraph. If the paragraph is longer than its capacity, BERT splits the paragraph into chunks by a sliding window approach. For every turn to calculate the start point of the next chunk, the *document stride* parameter is added to the last pointer in the paragraph. On predicting the answer process, the selected span from the paragraph can be as long as the *maximum answer length* parameter. It is a character-based parameter, although all other parameters are token-based.

4.6. EVALUATION METRICS

Similar to other machine reading comprehension and SQuAD studies, Exact Match (EM) and F-Score will be used as the evaluation metrics. Exact match counts the prediction only if it is same as the real answer, the F-Score counts the predictions that are overlapping with the real answer. EM is stiffer than F-Score.

5. IMPLEMENTATION

In the study, although we didn't directly change the BERT codes, this section gives information about some implementations we have done for the preparation of the data. Section 5.1 summarizes the translation process of the data sets. Section 5.2 describes the formatting procedures of the data sets. Section 5.3 presents how to generate the extended data set used in the study. Section 5.4 defines the procedures of the collecting statistics of the data sets, and Section 5.5 gives information about how to build a vocabulary file used in BERT training. The BERT codes are present in Google's GitHub repository [26].

5.1. TRANSLATING DATA SETS

We translated SQuAD and NewsQA data sets from English to Turkish using paid Google Translate API (v3), served by Google Cloud Platform. Due to the Google Translate API's multi-process prevention protections, operations had been done sequentially; paragraphs, questions, and answers had been converted one by one. Translation of every data set to Turkish nearly took two days. When using the API, it is essential to manually set the source language to English. For unique names, API overwise detects incorrect languages such as German or Spanish and causes erroneous translations.

Since some parts of the post-translation data become contrary to the MRQA rules, a rework is done:

1. The answers, which are no longer present in paragraphs, are deleted.
2. Since some answers are deleted in the previous step, we remove the questions that have no answers. Approximately 50 percent of the data has been lost after machine translation.
3. In some of the MRQA data sets, the 'start point' is marked where the answers begin in the paragraph. Since the location of these marks would have changed as a result of translation, they are recalculated. If the answer appears in more than one place in the paragraph, we manually check and mark the correct index.

5.2. FORMATTING THE DATA SETS

The format of the SQuAD dataset is the most minimal in the QA datasets and only contains the required fields for MRQA studies. BERT official codes natively support the SQuAD files for the MRQA task. An example of SQuAD v1.1 data set is in Figure 5.1. SQuAD file form is simple, as seen in Figure 5.2. *Data* field includes the documents, there are *paragraphs* and a *title* in every *data*, there are *questions* in every *paragraph*, there are *answers* of every *question*, and every *answer* has a *text* and an *answer_start* field which points the starting location of the text in the paragraph (indexOf function and zero-index included).

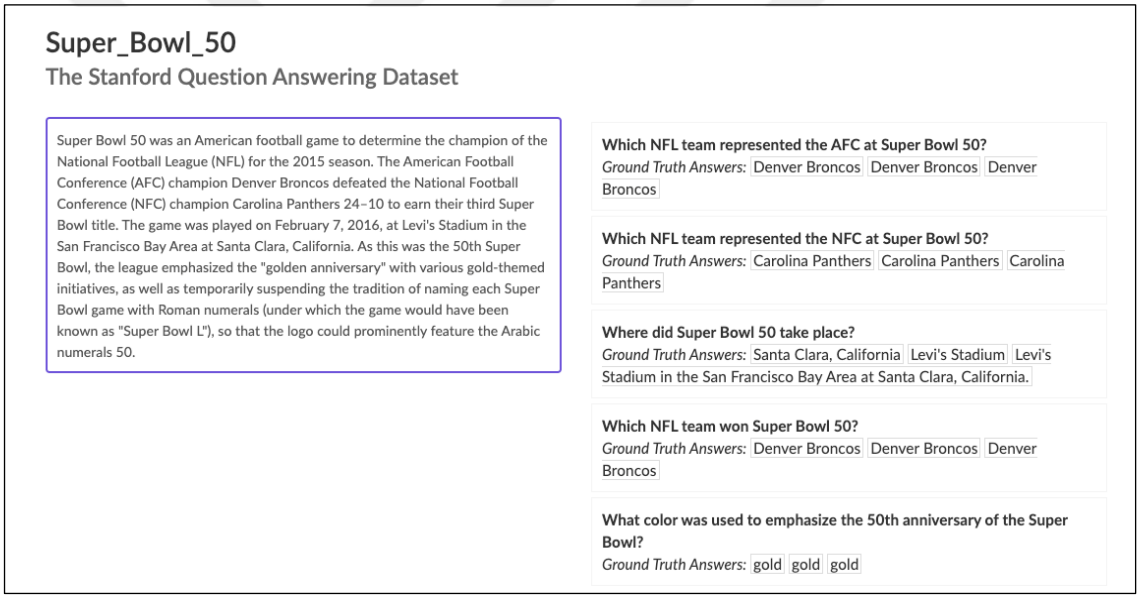

Figure 5.1. A SQuAD data set example.

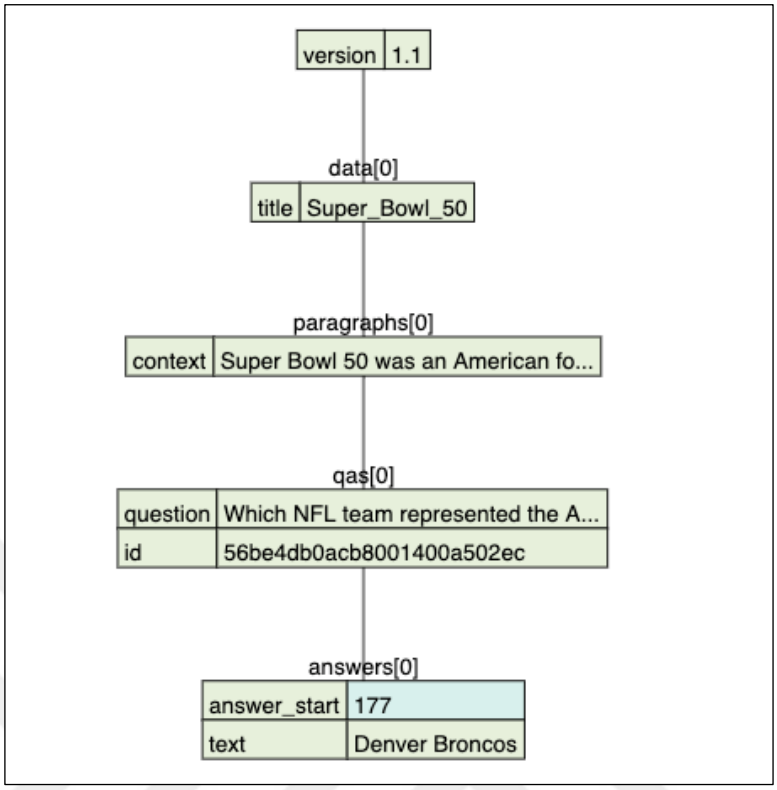

Figure 5.2. SQuAD file format example.

The NewsQA data set has some additional fields for other NLP tasks; *question tokens*, *answer spans*, and *context tokens*. An example of the NewsQA data set is in Figure 5.3. Since the context tokens have hundreds of items, for fitting the captured image to this page, we manually deleted other question and context tokens. These tokens and spans are not necessary for the MRQA task and pruned from the NewsQA (Tr) data set. The rest of the fields are easily converted to the SQuAD form, as they are logically in the same manner.

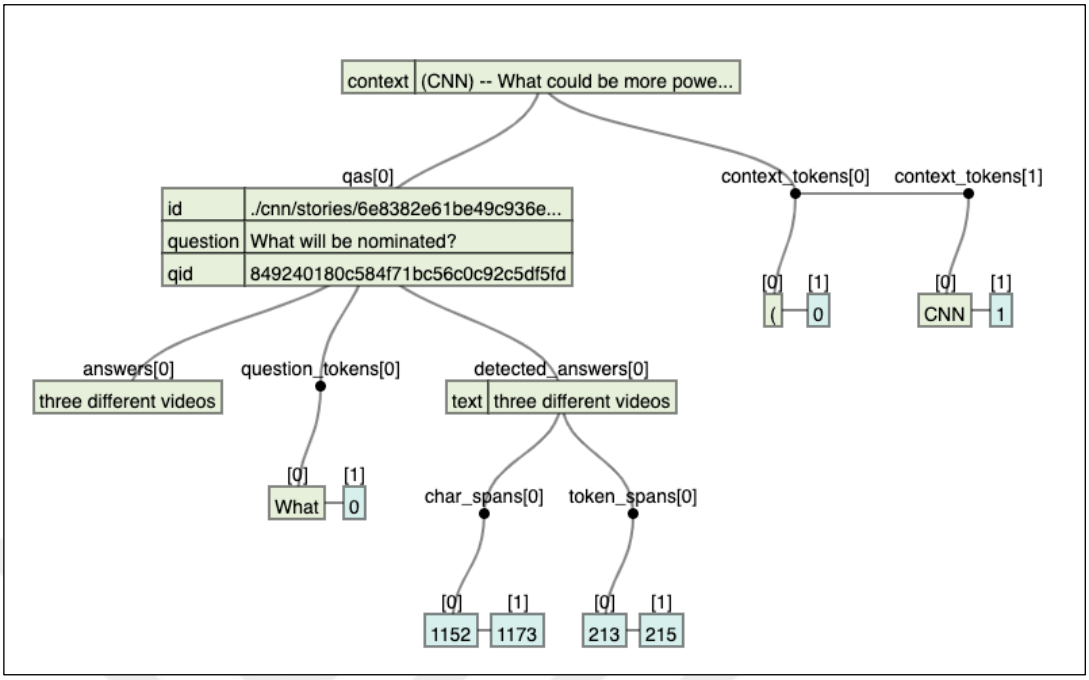

Figure 5.3. NewsQA file format example.

5.3. GENERATING THE EXTENDED DATA SET

After SQuAD and NewsQA data sets were converted to the same format as described in 5.2 section, we combined them. To increase the success of training, we also shuffled the elements of the result data set for randomness.

5.4. COLLECTING DATA SET STATISTICS

For selecting the correct parameter values of BERT, it is necessary to know the lengths of the contents in the data set. Paragraph and question content lengths are measured by token, and the answer length is measured by characters. It is easy to count the length of a text in *character units*, but the *token unit* requires some additional operations. The tokenizer used by BERT is the WordPiece algorithm, and the first and foremost part of the tokenizer is the vocabulary file. WordPiece divides the text into pieces using this vocabulary file. The tokenizer output of different vocabulary files is also different. For these reasons, the vocabulary file must be created before tokenizing the paragraph and the questions of the data set. The steps of creating the Vocabulary file is described in next section.

5.5. CREATING THE VOCABULARY FILE

BERT has a rule: in the corpus file, every line should contain only one sentence. Using Zemberek tokenizer, the raw corpus files are converted as one sentence per line. To create the vocabulary file for Turkish, we used the codes in the [31] repository. The most important two variables are the *corpus* and *minimum count* parameter. Two variables are interdependent; the vocabulary file always be different when the content of the *corpus* is changed or when the *minimum count* parameter is changed. The algorithm extracts subwords from the corpus and counts their occurrences. WordPiece puts the subwords to the vocabulary file, which are occurred in the corpora more than the given *minimum count* parameter. BERT recommends the vocabulary file size as 32K subwords. We trained the system with various sized vocabulary files by changing the *minimum count parameter*. Other parameters were used with their default values, and for each time, *five iterations* were done.

6. TEST AND EVALUATION

This section gives the details of our experiments in the study. Section 6.1 presents the pre-training and Section 6.2 presents the fine-tuning training steps and their test results in details. Section 6.3 analyzes the errors in the results. Section 6.4 compares the model with previous Turkish QA solutions. Section 6.5 compares the SQuAD data set evaluation results of the model with the BERT models of other languages, which are already trained for a QA system using translated SQuAD data set. Last, Section 6.6 compares the results of the pre-training BERT models that supports Turkish.

For splitting the banking sector QA data set to training and test sets, we used k-fold cross-validation. 80 percent of the data is selected for training and 20 percent of the data is selected for testing. The process is repeated five times; as a result, every question or answer is at once in the test set. The average score of these 5 experiments is accepted as the final accuracy.

6.1. PRE-TRAINING EXPERIMENTS

In this study, we used three different corpus set, as described in the 4.3 section. At the beginning of the study, we did not have all the corpus set together. Therefore, the corpuses have been added to the system when they were available. Figure 6.1 shows the experiments of the pre-training process.

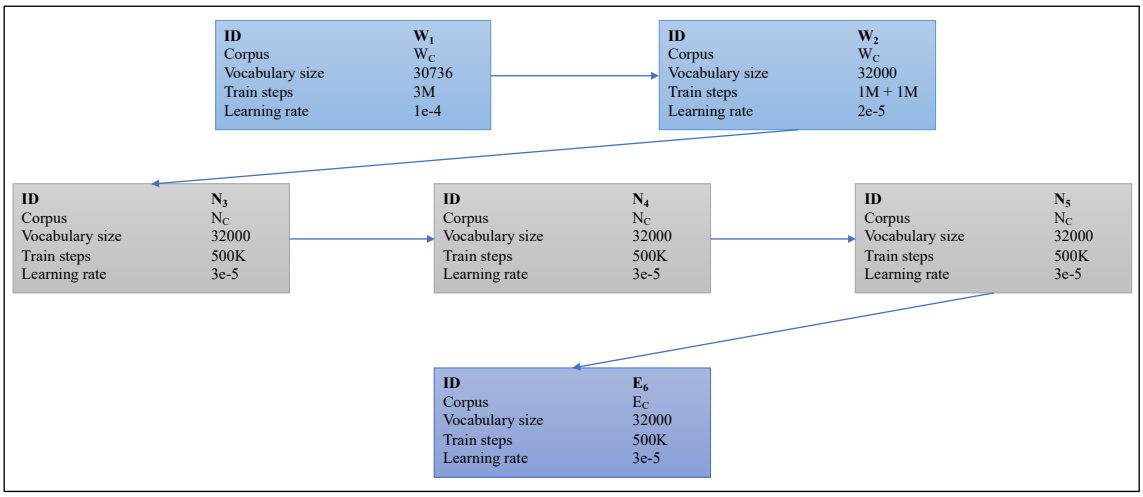

Figure 6.1. Pre-training experiments.

W_1: Firstly, we trained the system using Wikipedia corpus, which is medium-sized data. Three million training steps were applied with a low learning rate (1e-4). The vocabulary file had 30736 subwords generated using Wikipedia corpus. Evaluation data of W_1 model is the Wikipedia corpus data. It took 10 days on a TPU server.

W_2: Then, we extended the vocabulary file to 32000 subwords by manually adding economy/finance domain common words such as ATM, OTP, IBAN, arbitaj. The output of the W_1 was fine-tuned two million more training steps with a medium learning rate (2e-5). Evaluation data of W_2 model is the Wikipedia corpus data. It took 5 days on a TPU server.

N_3: When the News Corpus was available, it has been added to the system. News corpus is larger than Wikipedia. The output of the W_2 was fine-tuned five hundred training steps with a high learning rate (3e-5). Evaluation data of N_3 model is the News corpus data. It took 2 days on a TPU server.

N_4: The output of the N_3 was fine-tuned five hundred training steps with a high learning rate (3e-5). Evaluation data of N_4 model is the News corpus data. It took 2 days on a TPU server.

N_5: The output of the N_4 was fine-tuned five hundred training steps with a high learning rate (3e-5). Evaluation data of N_5 model is News corpus data. It took 2 days on a TPU server.

E_6: Then, to observe the effect of the economy data in the corpus, an economy corpus had been crawled from Web. When the Economy Corpus was available, it has been added to the system. Economy corpus is very smaller than the other corpuses. The output of the N_5 was fine-tuned five hundred training steps with a high learning rate (3e-5). Evaluation data of E_6 model is the Economy corpus data. It took 1 day on a TPU server.

It is an experimental result to use low learning rates for initial trainings and higher learning rates for fine-tunings. The BERT configuration values are in Table 6.1. During fine-tuning a present model, it is forbidden to change the vocabulary size. Therefore, in W_1 although the vocabulary size 30736, it was set to 32000 in the configuration. All the experiments had been done on a paid Google Cloud TPU (Tensor Processing Unit) server - v2. It is possible to train shorter sequence lengths on GPU, but 512 requires a TPU hardware.

Table 6.1. BERT parameters and values used in the experiments.

Parameter	**Parameter value**
BERT configuration	BASE
attention_probs_dropout_prob	0.1
hidden_act	Gelu
hidden_dropout_prob	0.1
hidden_size	768
initializer_range	0.02
intermediate_size	3072
max_position_embeddings	512
num_attention_heads	12
num_hidden_layers	12
type_vocab_size	2
vocab_size	32000
max_seq_length	512
do_lower_case	False
max_predictions_per_seq	75
masked_lm_prob	0.15
dupe_factor	5

The evaluation results of W_1, W_2, N_3, N_4, N_5 and E_6 are in Table 6.2. BERT takes the evaluation data of both *masked LM* and *next sentence prediction* from the corpus that already used in the training. For W_1 and W_2, the test data is from the Wikipedia corpus. For N_3, N_4 and N_5, the test data is from the News corpus. For E_6, the test data is from the Economy corpus. Predictably, comparing the test results of different corpus is an invalid test. If the corpus is larger, its accuracy will be lower.

Table 6.2. Evaluation results of different corpus sets.

Training ID	**Masked LM accuracy**	**Next sentence accuracy**
W_1	81.22%	100.00%
W_2	76.92%	99.50%
N_3	72.42%	98.25%
N_4	71.67%	96.63%
N_5	74.43%	98.88%
E_6	100.00%	100.00%

In the experiments, it is seen that;

- Although both W_1 and W_2 are evaluated using Wikipedia corpus, there is a loss on W_2 result. Economy words, added to the vocabulary, have a negative effect on the results. The reason is, the evaluation data is from Wikipedia, which is a general purpose corpus.
- Compared with W_2, the accuracy of N_3 is lower. The reason is, the News Corpus is larger.
- More training steps have a positive effect for N_5 and N_3.
- The accuracy of E_6 is the highest. The reason is, the Economy Corpus is the smallest corpus.

The language model accuracies are not directly consistent with the MRQA task accuracies. The evaluation results of these models on MRQA task are listed in Section 6.2.

In the experiments, different vocabulary sizes had been tested. Figure 6.2 shows the vocabulary sizes for different *minimum occurrence threshold* values. For Wikipedia corpus, if minimum occurrence is set to 100, the generated vocabulary size is 96593 and if it is set to 500, the generated vocabulary size is 40786. In the tests, W_1 training scenario had been repeated with different vocabulary sizes as seen in the Figure 6.3. It is validated that, 30-32K is optimal for the vocabulary size.

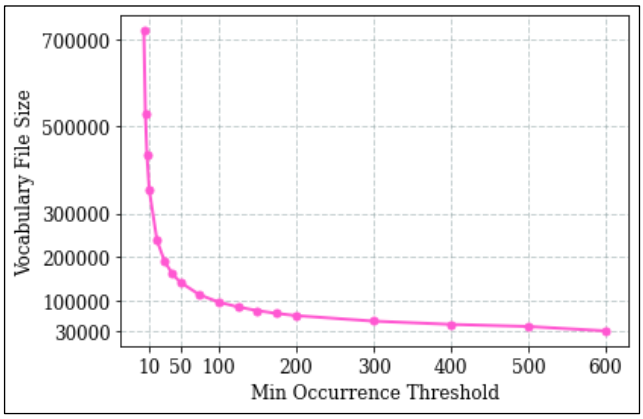

Figure 6.2. Minimum occurrence threshold parameter and vocabulary sizes.

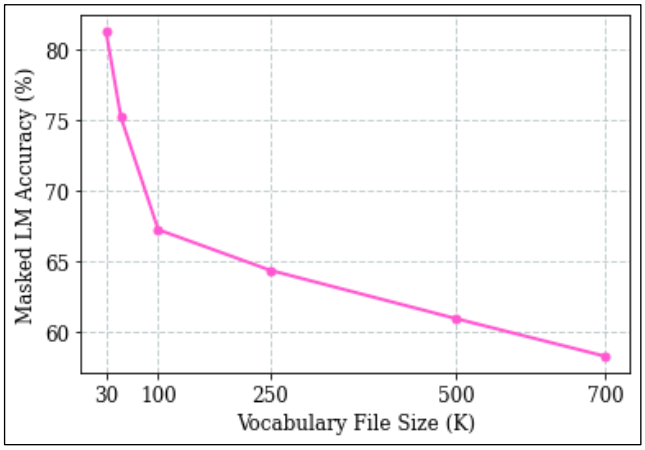

Figure 6.3. Vocabulary file size and results.

In the experiments, different *max_seq_length* values had been used. As proposed by BERT, 512 has given better results compared with 64, 128, 256 and 512.

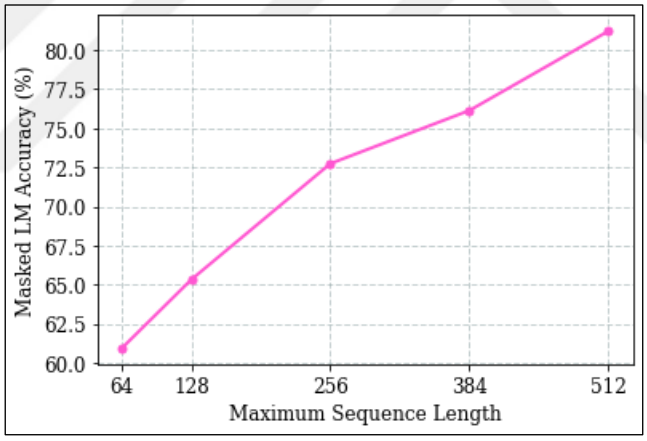

Figure 6.4. Maximum sequence length parameter and results.

In the experiments, the effects of the number of training steps had been measured. In Figure 6.5, MLM is the Masked LM accuracy and NSP is the next sentence prediction accuracy. It is seen that, using one million or more training steps gives better results.

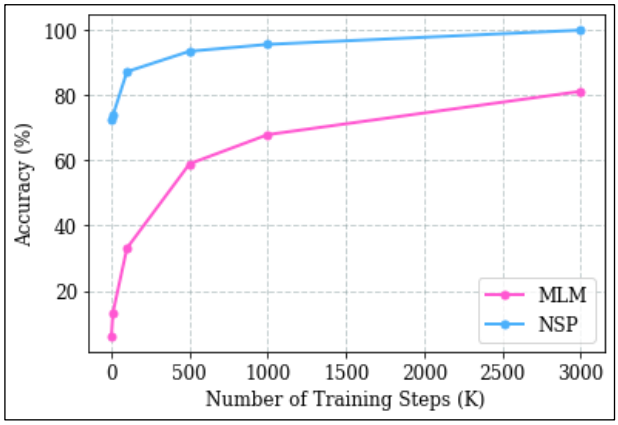

Figure 6.5. Number of training steps and results.

6.2. FINE-TUNING EXPERIMENTS

The pre-training experiments have 6 output models, as detailed in Section 6.1: W_1, W_2, N_3, N_4, N_5, E_6. In order to find the pre-training model, which gives the highest Exact Match (EM) and F-Scores for fine-tuning the Banking Sector QA Dataset (DS_B), all pre-training models had been experienced. Figure 6.6 shows the results of the fine-tuning DS_B with all the pre-training models. The numbers in the legend are the subscripts of the pre-training models. N_5 and E_6 have the highest F1-Scores but, the Exact Match result of E_6 is lower. Therefore, for further experiments, N_5 had been used. As expected, W_1 had the worst result in the set.

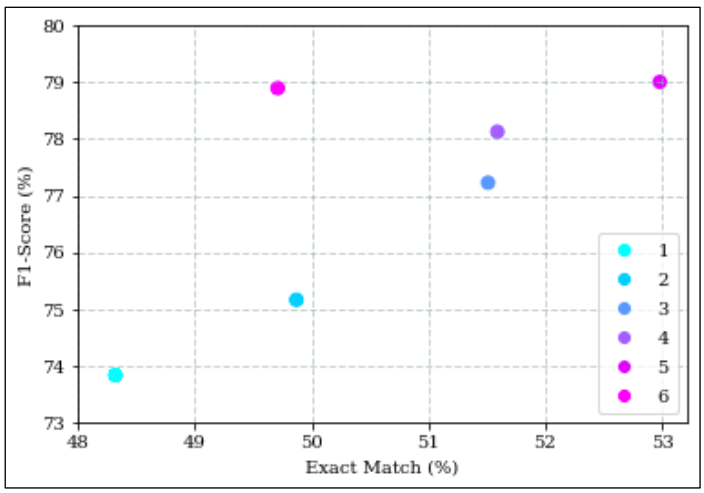

Figure 6.6. The pre-training models and their fine-tuning results.

Cased model accuracy is 2-3 percent higher than the *uncased* model for Turkish. Therefore, the cased model had been used for all experiments.

Longer maximum sequence lengths had a positive effect on training. Increasing the maximum sequence length (64 to 128, 128 to 256, 256 to 384 and 384 to 512) raised the masked LM accuracy ~3 percent for each step (Figure 6.7). Therefore, for further experiments, 512 has been used. But the attention is quadratic to the sequence length. Hence, increasing the sequence length took more training time.

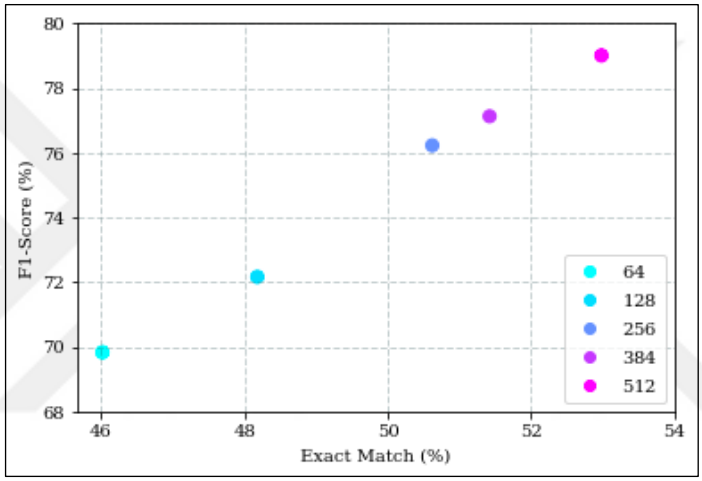

Figure 6.7. Different max_seq_length parameters and results.

To determine the success of the system, we examined different training parameters. When choosing these parameters, we based on our computations of the paragraph, question, and answer lengths in the data sets. We decided on some values and analyzed all their combinations. The best and worst parameter values for the Banking Sector QA data set are in Table 6.3. Figure 6.8 presents different combinations and accuracy results in terms of EM and F-Score. In (a) and (b) subfigures, 64 - 512 are the token counts, and in (c) 30 - 128 are the character length.

Table 6.3. Best and worst parameter values for the Banking Sector QA data set.

Maximum sequence length (token)	**Document stride (token)**	**Maximum query length (token)**	**Maximum answer length (character)**	**EM**	**F-Score**
512	256	64	64	54,09	79,01
128	64	64	30	44,38	70,11

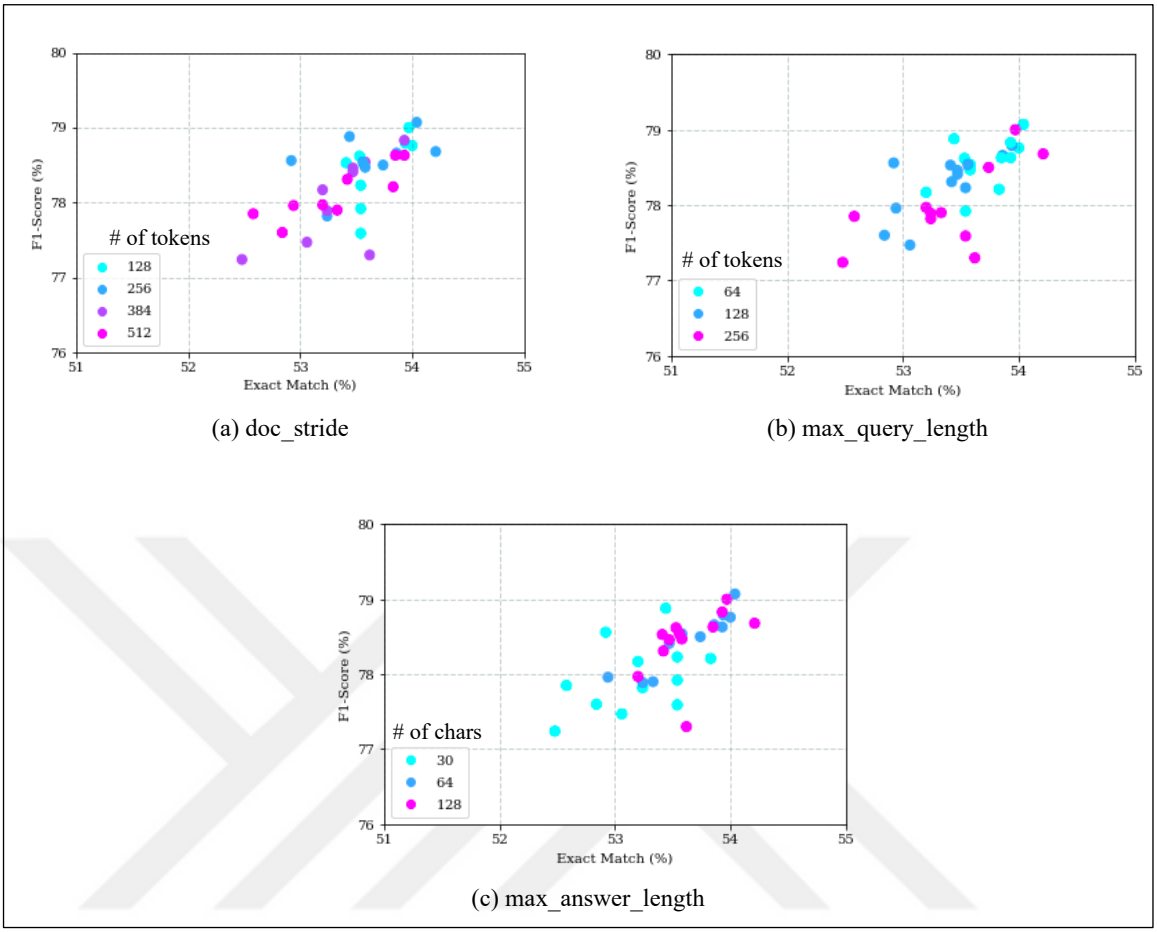

Figure 6.8. Different parameters and results.

Multiple data sets are available for training; NewsQA (Tr), SQuAD (Tr), and Banking Sector QA (Tr) data sets. The main goal is achieving a high score for the evaluation of the Banking Sector QA data set. The SQuAD (Tr) and NewsQA (Tr) are open-domain and large data sets. We think that doing fine-tuning more than one phase as: first, fine-tuning with open-domain data sets and then fine-tuning with the Banking Sector QA data set will increase the success of the system. Because of multiple open-domain data sets (SQuAD and NewsQA), different training scenarios are possible, as seen in Table 6.4 and Figure 6.9.

Table 6.4. Different phase combinations of the data sets.

Choice	**Phase 1 Data Set (Fine-tuning)**	**Phase 2 Data Set (Fine-tuning)**	**Phase 3 Data Set (Fine-tuning)**
1	SQuAD (Tr)	Banking Sector QA (Tr)	-
2	NewsQA (Tr)	Banking Sector QA (Tr)	-
3	SQuAD (Tr)	NewsQA (Tr)	Banking Sector QA (Tr)

4	NewsQA (Tr)	SQuAD (Tr)	Banking Sector QA (Tr)
5	SQuAD + NewsQA union data set	Banking Sector QA (Tr)	-

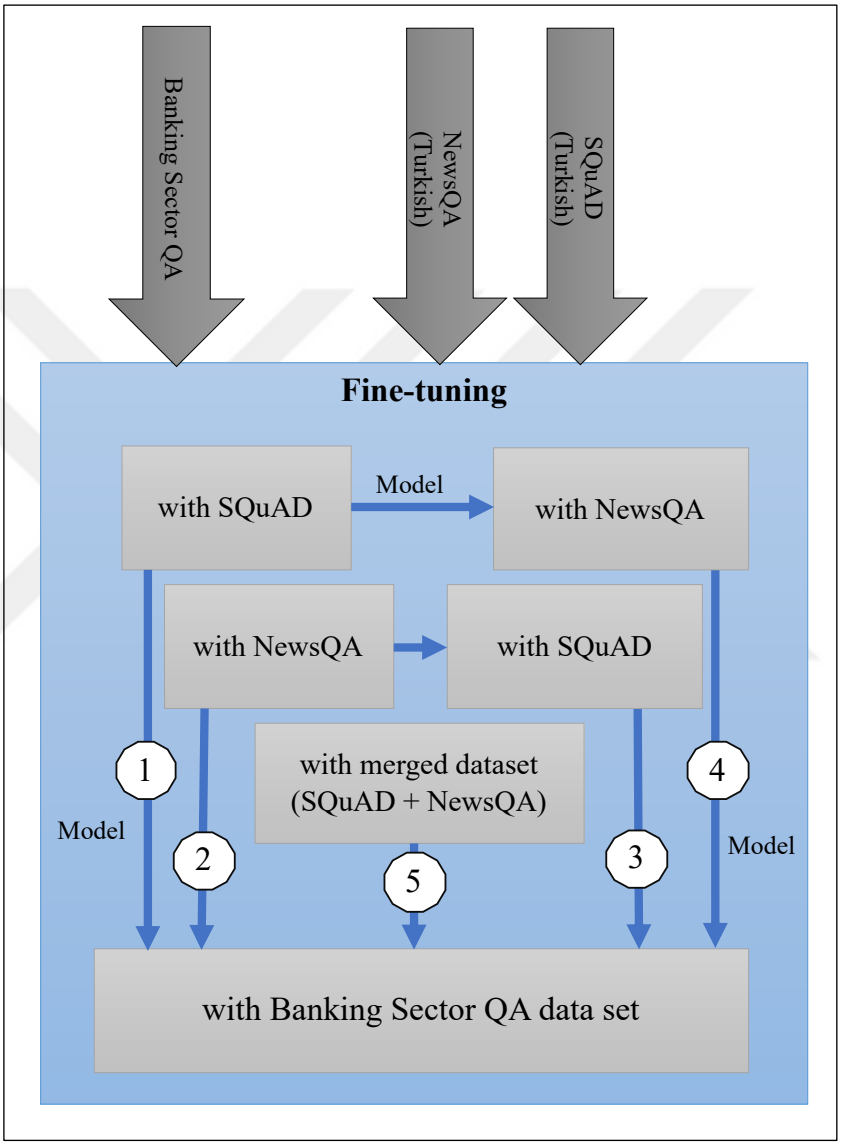

Figure 6.9. Training scenarios of fine-tuning data sets.

For finding the scenario which has the highest accuracy, we studied several different data set combinations. In Figure 6.10;

- DS_S indicates SQuAD Data Set,
- DS_N indicates NewsQA Data Set,
- DS_E indicates SQuAD + NewsQA Data Sets (extended union data set),

- DS_B indicates Banking Sector QA Data Set.

In the figure, every row has multiple data sets. For the training process, there is a chain logic. Using the given data set, the output of every training box is the input of the next one. Every model is evaluated with the last data set item of the chain. For example, M_5 is first trained with DS_S and then fine-tuned with DS_B and evaluated using the DS_B data set.

In the experiments, it is seen that;

- Fine-tuning with the *SQuAD (Tr)* data set, before training of the *NewsQA (Tr)* data set, has a positive effect on the *NewsQA (Tr)* data set evaluation results (Figure 6.11 - a).
- Fine-tuning with the *NewsQA (Tr)* data set, before training of the *SQuAD (Tr)* data set, has a negative effect on the *SQuAD (Tr)* data set evaluation results (Figure 6.11 - b).
- Fine-tuning with *SQuAD (Tr) + NewsQA (Tr)* (Extended) data set, before training of the *Banking Sector QA* data set, has a positive effect on the *Banking Sector QA* data set evaluation results (Figure 6.11 - c).

Figure 6.10. Different training scenarios and data sets.

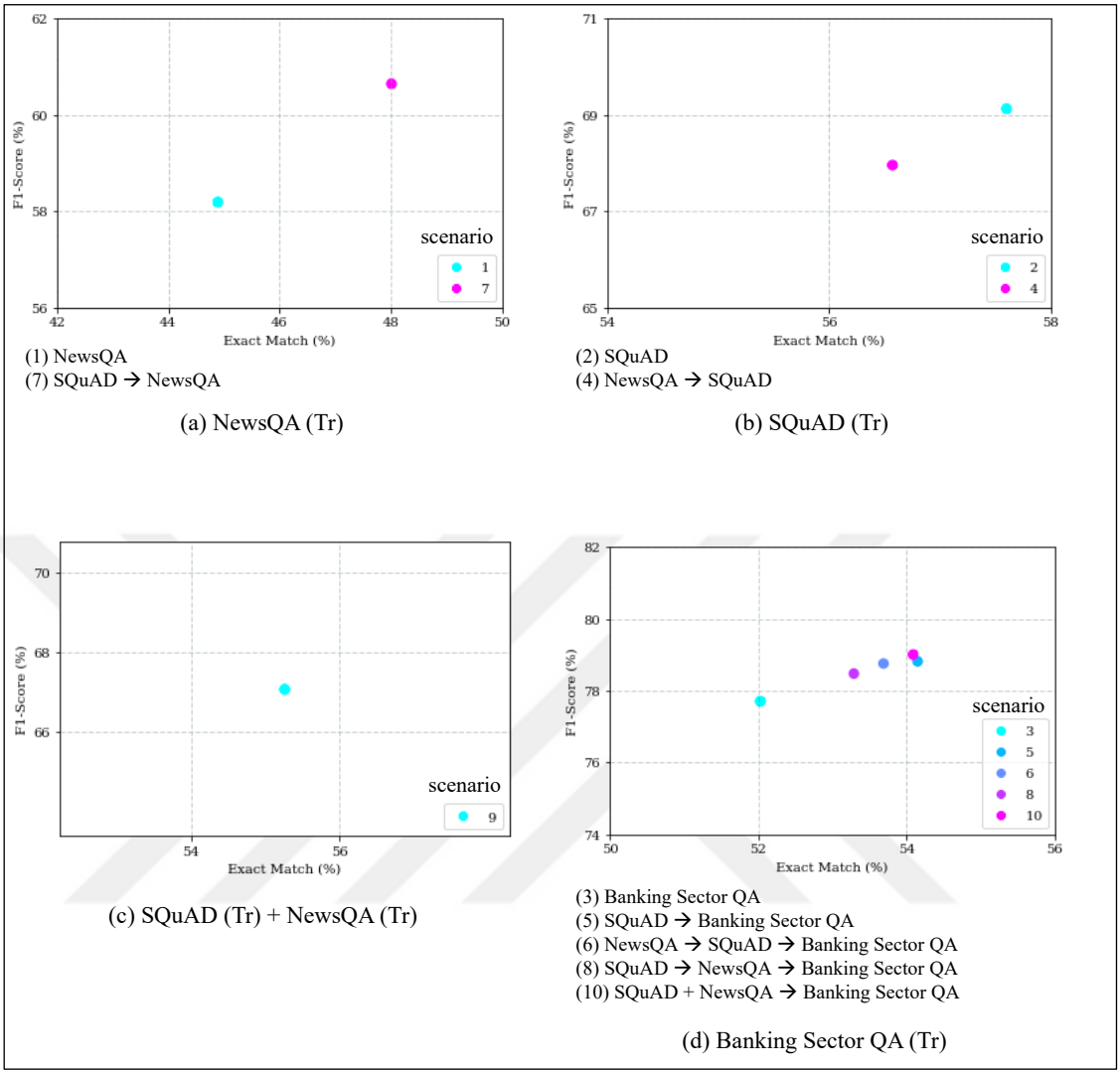

Figure 6.11. Different scenarios and data set evaluation results.

Five epochs count was ideal for training MRQA with BERT. Two epochs were insufficient, and 10 epochs were not notably different than 5 epochs.

6.3. ERROR TYPES IN QUESTIONS & ANSWERS

To evaluate the system's real-world performance, we asked the team, who prepared the questions, to think freely during the process. When writing the items, we wanted them not to abide by any question patterns and write different forms of questions if possible. Therefore, there were many kinds of question types in both training and test data sets. The system gives correct answers to majority of common question types with a form or questions that have distinct answers in the text. When we analyze the results, we observed that Exact

Matches (EM) remained quite low compared to F-Scores. For this reason, we found the results with an EM of 0; 572 out of 17708 answers or 3.2 percent. Table 6.5 lists the error types of wrong answered questions and Table 6.6 gives some examples of these errors. The errors can be categorized as:

1- Multiple possible answers: Some questions may have more than one correct answer. In Table 6.6 for Error ID = 1, both the real and the predicted answer can be considered as correct. 30 percent of the samples, which have zero EM in the test set, are in this category.

2- Questions requiring interpretation: In order to answer some questions, it is mandatory to know about the whole subject. These questions require interpretation, and it is not easy to identify their answers. They are the most challenging types of questions, and no solution has yet been found for such questions in other languages, including English. In Table 6.6 for Error ID = 2, the predicted answer is the direct but incorrect answer of the question.

3- Conditional answer: The answer to some questions depends on the state. There are cases when the general answer is not valid, or on the contrary, the answer is only correct in certain situations. In Table 6.6 for Error ID = 3, the answer is *correct* except a type of customer who has a given brand's cell phone.

4- Questions requiring a list of elements: The answer to some questions consists of a list. Instead of giving a general answer to such questions, it is sometimes necessary to return the detailed list. In Table 6.6 for Error ID = 4, the hours in the answer is crucial.

5- Answers with syntax variations: Because of being an agglutinative language and having a comprehensive set of possible suffix tags, it isn't easy to generate the exact match of the real answer. In Table 6.6 for Error ID = 5, “it is” is the correct for the answer but causes a zero EM.

6- Incorrect question: After the team prepared the questions, no corrections were made to the written questions in order to practice the real-world. For this reason, some items in the data set have spelling mistakes, grammatical errors, or typos.

7- Incorrect answers: The system could not answer 25 percent of the questions properly. The reason for this is that, some questions are in an uncommon form which the system hasn't seen in the training set, and some questions or answers are cut by the system because they are longer than the given maximum length parameters.

Table 6.5. Description of wrong answers with zero EM (3.2%).

Error ID	Description	Counts	Percentage
1	Multiple possible answers	180	31.46%
2	Questions requiring interpretation	85	14.86%
3	Conditional answers	80	13.98%
4	Questions requiring a list of elements	5	0.8%
5	Answers with syntax variations	34	5.94%
6	Incorrect question	49	8.56%
7	Incorrect answers	139	24.30%
31,46	**Total**	**572**	**100%**

Table 6.6. Examples for error types 1-5.

Error ID	Question	Real answer	Predicted answer
1	What's IBAN?	Used for transferring money.	International Bank Account Number
2	How is the credit card objection made?	You should fill the form and fax.	From our website.
3	Does the application support all phone models?	Only except X brand	supports
4	When is the job executed?	00:00, 01:00, 02:00	in the night
5	Is it possible to pay the tax online?	possible	it is possible

6.4. COMPARISON WITH OTHER TURKISH QA SYSTEMS

The output model of this study has been compared with earlier QA systems developed for Turkish. Some of the existing systems are only trained for specific closed-domains, while some have yielded results in open-domains. Open-domain models find answers to different types of questions in a wide range of domains, rather than particular question types or a closed-domain. Some of the previous systems support only a small set of specific question types. Also, evaluation metrics of the published results are different; MRR, Precision, or EM with F-Score. In the light of all these circumstances, it is impossible to compare the results directly. Table 6.7 lists the accuracies of the systems. The model of this study had been tested in both open and closed-domains. In the open-domain, we trained the system for general purposes with the extended dataset, a combination of SQuAD (Tr) and NewsQA (Tr) data sets. In the closed-domain, we fine-tuned the open-domain model using the Banking Sector

QA (Tr) data set in order to enhance the performance in the banking field. Even if the metrics are different, we have noticed that the current system's output is either equal or higher in terms of skills and success rates compared to other systems.

In the table, TREC-9, TREC-10, SQuAD (Tr), and NewsQA (Tr) are data sets translated to Turkish. MRR is Mean Reciprocal Rank, which considers the rank of the first correct answer in the list of possible answers. The specific questions in the second row are: *who, where, when,* and *what*. Specific factoid questions in the third row are *Author, Capital, Date of Birth, Date of Death, Language of Country, Place of Birth, Place of Death*.
(*) is Phase 1: Fine-tuning the model using merged SQuAD (Tr) and NewsQA (Tr) data sets.
(**) is Phase 2: Fine-tuning the model, which is already trained with Step 1, using Banking Sector QA (Tr) data set.

Table 6.7. Comparison of Turkish QA Systems.

Study	**Data set**	**Domain**	**Metric**	**Results**
BayBilmiş	TREC-9 and TREC-10	Open	MRR	0,313
Automatic QA for Turkish with Pattern Matching	Only Specific Questions	Closed	Precision	0,79 (Average)
A Factoid QA System Using Answer Pattern Matching	Only Specific Factoid Questions	Closed	MRR	0,73
Current Study	SQuAD (Tr) and NewsQA (Tr) *	Open	EM F-Score	55,26 67,07
	Banking Sector QA (Tr) **	Closed	EM F-Score	52,98 79,01

6.5. COMPARISON WITH OTHER BERT LANGUAGE MODELS

To figure out our QA data sets, our model has been compared with the BERT models of other languages. The SQuAD is the best known MRQA data set. In this study, we translated SQuAD into Turkish and included in the training. A similar translation has also been done for Arabic, and Arabic Reading Comprehension Dataset (ARCD) is available. There are several QA data sets prepared for Chinese. Harbin Institute of Technology's joint laboratory has created the CMRC 2018 machine reading comprehension data set. CMRC 2018 resembles SQuAD format; the system extracts answers from chapters for the given question.

Delta Research Institute in Taiwan has built the DRCD data set. DRCD resembles SQuAD format, and it is in Traditional Chinese. Harbin Institute of Technology's Xunfei Joint Laboratory has released the CJRC machine reading comprehension data set for the judicial field. Table 6.8 list the results of some QA models based on BERT in English, Arabic and Chinese. Although some BERT models have developed for French, Persian, and Korean, these models do not yet have a success rate published for the question-answering task. In the results, it has been seen that our model's performance is better than the Arabic model, which is also trained with machine-translated SQuAD data set. Compared with Google's Chinese model, although our model has worse results than DRCD, it is mostly better than CMRC and CJRC data set performances. The main reason for the SQuAD results score differences between English and other languages might be the SQuAD data set's official language is English. There is a data loss for both the counts of the examples and their semantical meanings during the machine translation process to other languages.

In the table, SQuAD (Tr) and NewsQA (Tr) are data sets translated to Turkish. *mBERT* is the multilingual and *BERT-Chinese* is the Chinese models, published by Google. *ARCD* (Arabic Reading Comprehension Dataset) is the Arabic SQuAD data set that is already translated from the original English SQuAD. (*) is Phase 1: Fine-tuning the model using merged SQuAD (Tr) and NewsQA (Tr) data sets.

Table 6.8. Comparison of BERT models with other languages.

Language	**Model**	**Data sets**		**Domain**	**EM**	**F-Score**
		Training	**Evaluation**			
English	BERT (Single Model)	SQuAD (En - Original)	SQuAD	Open	85,08	91,83
Arabic	mBERT	ARCD	ARCD		34,2	61,3
	AraBERT	ARCD	ARCD		30,6	62,7
Chinese	BERT-Chinese	CMRC	CMRC		18,6	43,3
		DRCD	DRCD		82,2	89,2
		CJRC	CJRC		55,1	75,2
Turkish	Current Study	SQuAD (Tr)	SQuAD (Tr)		57,60	68,34
		NewsQA (Tr)	NewsQA (Tr)		48,01	59,86
		SQuAD (Tr) + NewsQA (Tr) *	SQuAD (Tr) + NewsQA (Tr) *		55,26	67,07

6.6. COMPARISON WITH OTHER BERT TURKISH MODELS

Apart from our model, two more models are available in Turkish. One of them is mBERT, the multilingual model that Google published for all languages. The other is BERTurk [32], which has been specially trained for Turkish. We compared the success of our model with mBERT and BERTurk in Turkish. Table 6.9 lists the results of the models. In the results, it has been seen that our model's performance is better than the mBERT model and a bit worse (~2 percent) than BERTurk. The number and size of the corpora used in the training of BERTurk is larger than ours. This might be the reason of the 2 percent differences in the accuracies.

In the table, SQuAD (Tr) and NewsQA (Tr) are data sets translated to Turkish. *mBERT* is the multilingual model published by Google. (*) is Phase 1: Fine-tuning the model using merged SQuAD (Tr) and NewsQA (Tr) data sets. (**) is Phase 2: Fine-tuning the model, which is already trained with Step 1, using Banking Sector QA (Tr) data set.

Table 6.9. Comparison of Turkish base models.

Data sets		**Model**	**Domain**	**EM**	**F-Score**
Training	**Evaluation**				
SQuAD (Tr) and NewsQA (Tr) *	SQuAD (Tr) and NewsQA (Tr) *	BERTurk	Open	57,43	69,36
		Current Study		55,26	67,07
		mBERT		54,52	65,74
Banking Sector QA (Tr) **	Banking Sector QA (Tr) **	BERTurk	Closed	55,89	80,87
		Current Study		54,09	79,01
		mBERT		50,74	77,03

7. CONCLUSION

In this thesis, we trained a question-answering system using deep learning methods. The output model of the study has been trained and tested both in open and closed-domains. Used deep learning method builds a language model consists of contextual word representations and tackles with polysemy. Our most important findings are;

In Turkish question answering systems, deep learning models give better results in both open and closed-domains. Compared with traditional approaches, deep learning methods have constant improvement in the skills of answering a wide range of questions. Statistical or rule-based purposes have succeeded only in a limited subject or predefined question types. This inference is not only specific to NLP, and also valid for other artificial intelligence problems.

The data sets used in the training of the system have an important role in the performance of the system. The corpora used in the pre-training directly affects the quality of the generated language model. The QA datasets used in the fine-tuning directly affect the system's ability to question-answering. Considering the comparison of our training model with BERTurk, the only difference is that BERTurk has been pre-trained with a larger corpus. The performance of the language model also increased its QA success rates. For this reason, it is important to pay attention to preparation of the data.

The fine-tuning datasets we translated into Turkish contributed to success. Even though we used automatic machine translation and the process caused a loss in the data sets, the SQuAD and NewsQA datasets have a tremendous effect on the system's performance for QA task. A similar approach can be applied to other languages that suffer from massive data shortages for applying deep learning methods.

Data sets prepared for English can be translated into any language using machine translation, and open-domain QA systems can be trained using these translated data sets. These general-purpose models can be fine-tuned with a proper data set for a closed-domain system later on, if needed. Although the creation of QA data sets is time-consuming and requires human resources, it is still an inevitable requirement to build a successful system. In recent years,

the researches on the NLP domain have considerably increased the number of modern approaches, such as the BERT method used in this thesis.

To summarize, in this study, we proposed an MRQA system for Turkish in the banking domain. To the question, the system generates the best answer, which is the most correct and the shortest span in a given text. Applying the BERT deep learning technique, we trained a language model for Turkish using massive corpus collections followed by a fine-tuning process for the MRQA task using large QA datasets. To enhance the MRQA skills of the systems, we also translated some open-domain QA datasets from English to Turkish. At the end of the experiments, it is seen that the system's accuracy is higher than other QA solutions for Turkish. Additionally, the proposed method is not specific to Turkish and applicable for numerous NLP tasks of other limited languages.

It will be important that future research investigates the performance problems of BERT. Although state-of-art scores are announced using the BERT model, it is still slow in performance and requires TPU hardware for better results. Some newer models are recently become available such as; ELECTRA, T5, or GPT-3. It may be useful to have the same experiments with these methods and compare their evaluation results with BERT.

REFERENCES

1. Rajpurkar P, Zhang J, Lopyrev K, Liang P. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*. 2016;2383-92.

2. Devlin J, Chang M-W, Lee K, Toutanova K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*. 2018.

3. Amasyalı MF, Diri B. Bir soru cevaplama sistemi: Baybilmiş. *Türkiye Bilişim Vakfı Bilgisayar Bilimleri ve Mühendisliği Dergisi*. 2005;1(1):1-1.

4. Biricik G, Solmaz S, Özdemir E, Amasyalı MF. A Turkish Automatic Question Answering System with Question Multiplexing: Ben Bilirim. *International Journal of Research in Information Technology (IJRIT)*. 2013;1(6):46-51.

5. Çelebi E, Günel B, Şen B. Automatic question answering for Turkish with pattern parsing. *2011 International Symposium on Innovations in Intelligent Systems and Applications*. 2011:389-93.

6. Er NP, Cicekli I. A factoid question answering system using answer pattern matching. *Proceedings of the Sixth International Joint Conference on Natural Language Processing*. 2013:854-8.

7. Zheng Z. AnswerBus question answering system. *Proceedings of the Second International Conference on Human Language Technology Research*. 2002:399-404.

8. Akın AA, Akın MD. Zemberek, an open source nlp framework for Turkic languages. *Structure*. 2007;10:1-5.

9. Eryiğit G, Oflazer K. Statistical dependency parsing for turkish. *11th Conference of the European Chapter of the Association for Computational Linguistics*. 2006.

10. Oflazer K, Say B, Hakkani-Tür DZ, Tür G. Building a Turkish Treebank. *Treebanks*. 2003:261-77.

11. Derici C, Çelik K, Kutbay E, Aydın Y, Güngör T, Özgür A, et al. Question analysis for a closed domain question answering system. *International Conference on Intelligent Text Processing and Computational Linguistics*. 2015:468-82.

12. Sahin M, Sulubacak U, Eryigit G. Redefinition of Turkish morphology using flag diacritics. *Proceedings of The Tenth Symposium on Natural Language Processing (SNLP-2013), Phuket, Thailand, October*. 2013.

13. Trischler A, Wang T, Yuan X, Harris J, Sordoni A, Bachman P, et al. Newsqa: A machine comprehension dataset. *arXiv preprint arXiv:1611.09830*. 2016.

14. Yang Z, Qi P, Zhang S, Bengio Y, Cohen WW, Salakhutdinov R, et al. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*. 2018.

15. Seo M, Kembhavi A, Farhadi A, Hajishirzi H. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*. 2016.

16. Peters ME, Neumann M, Iyyer M, Gardner M, Clark C, Lee K, et al. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*. 2018.

17. Radford A, Salimans T. Improving language understanding by generative pre-training. 2018.

18. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, et al. Attention is all you need. *Advances in neural information processing systems*. 2017:5998-6008.

19. Fisch A, Talmor A, Jia R, Seo M, Choi E, Chen D. MRQA 2019 shared task: Evaluating generalization in reading comprehension. *arXiv preprint arXiv:1910.09753*. 2019.

20. Antoun W, Baly F, Hajj H. AraBERT: Transformer-based model for Arabic language understanding. *arXiv preprint arXiv:2003.00104*. 2020.

21. Cui Y, Che W, Liu T, Qin B, Yang Z, Wang S, et al. Pre-training with whole word masking for chinese bert. *arXiv preprint arXiv:1906.08101*. 2019.

22. Martin L, Muller B, Suárez PJO, Dupont Y, Romary L, de la Clergerie ÉV, et al. Camembert: a tasty french language model. *arXiv preprint arXiv:1911.03894*. 2019.

23. Korean BERT pre-trained cased (KoBERT) Github page 2020 [cited 2020 1 June]. Available from: https://github.com/SKTBrain/KoBERT.

24. Farahani M, Gharachorloo M, Farahani M, Manthouri M. ParsBERT: Transformer-based Model for Persian Language Understanding. *arXiv preprint arXiv:2005.12515*. 2020.

25. Wu Y, Schuster M, Chen Z, Le Q v., Norouzi M, Macherey W, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*. 2016.

26. Hui J., NLP - BERT & Transformer Medium homepage 2020 [cited 2020 1 June]. Available from: https://medium.com/@jonathan_hui/nlp-bert-transformer-7f0ac397f524.

27. Google BERT Github page 2020 [cited 2020 1 June]. Available from: https://github.com/google-research/bert.

28. Wiedemann G, Remus S, Chawla A, Biemann C. Does BERT make any sense? Interpretable word sense disambiguation with contextualized embeddings. *arXiv preprint arXiv:1909.10430*. 2019.

29. Wikimedia homepage [cited 2020 1 June]. Available from: https://dumps.wikimedia.org/backup-index.html.

30. Vocabulary builder for BERT Github page 2020 [cited 2020 1 June]. Available from: https://github.com/kwonmha/bert-vocab-builder.

31. Schweter S. BERTurk - BERT models for Turkish: Zenodo; [cited 2020 1 June]. Available from: https://doi.org/10.5281/zenodo.3770924.

APPENDIX A: BANKING SECTOR QA EXAMPLES

You should take permission from the author to use or copy these data sets.

Table A.1. Example document 1.

Example Document	1
Title	Müşterilerimiz ATM'lerden işlem yaparken güvenlik açısından nelere dikkat etmeli?
Context	Kart şifrenizi Bankamız çalışanları dahil kimsenin öğrenmesine izin vermeyiniz. Şifrenizi belirlerken doğum tarihiniz, telefon numaranız, kart numaranız gibi kolay elde edilebilecek bilgilere dayanarak oluşturmayınız. İşlemleriniz sırasında yardım tekliflerini kabul etmeyiniz. Kartınızı ATM'den geri almadan, ATM'nin yanından ayrılmayınız. Kartınızı kaybetmeniz/çaldırmanız durumunda ya da diğer her türlü soru ve sorunlarınız için vakit kaybetmeden 0850 200 0 666 TEB Telefon Şubesi'ni arayınız.
Question-1	Kart kaybedilirse ne yapılmalıdır?
Answer-1	0850 200 0 666 TEB Telefon Şubesi'ni arayınız
Question-2	Şifre belirlerken nelere dikkat etmeliyiz?
Answer-2	doğum tarihiniz, telefon numaranız, kart numaranız gibi kolay elde edilebilecek bilgilere dayanarak oluşturmayınız
Question-3	İşlem sırasında yardım almalı mıyız?
Answer-3	yardım tekliflerini kabul etmeyiniz

Table A.2. Example document 2.

Example Document	2
Title	IBAN (ULUSLARARASI BANKA HESAP NUMARASI)
Context	Avrupa Birliği düzenlemeleri çerçevesinde, ülkeler arasında gerçekleştirilen para transferlerinin hızı ile kalitesini artırmak ve maliyetlerini düşürmek amacıyla International Bank Account Number-IBAN adı verilen Uluslararası banka hesap numarası standardı geliştirilmiştir. IBAN 26 haneden oluşmaktadır. Halk Bankası IBAN numaraları harf içerebilmektedir. TR38 0003 2000 1250 0000 0111 01. Müşterilerimiz hesaplarına ait İBAN numaralarını IVR üzerinden öğrenebiliyor ve sistem otomatik olarak bu bilgiyi SMS olarak gönderiyor. Pusulada hesap işlemleri altına eklenen "IBAN Bilgisini SMS Gönder" menüsü ile müşterilerimize IBAN bilgisini SMS olarak gönderebiliyoruz. İşlemler Inbound aramalarda yapılabilecek, güvenlik seviyesi 1. Düzey+SMS OTP (SMS OTP telefonundan arıyorsa SMS OTP tuşlamayacak) olacak şekilde düzenlendi. Müşterinin cep telefonlarını listeleyeceğiz, seçim kısmından kayıtlı hangi telefonuna isterse gönderebileceğiz. Cep telefonuna yapılan gönderimlerde şube kodu, hesap no, IBAN bilgileri olacaktır. IBAN oluşan tüm hesap türleri üzerinden menü açılacak ve IBAN bilgisi gönderilebilecektir. Yapılan işlemler irtibat gözleme kayıt atacaktır. Menü Bireysel, Ticari, Tüzel müşterilerimiz için açılabilecektir. IBAN Bilgilerinin SMS'le Gönderilmesi Hesap işlemleri üzerinden IBAN Bilgisini SMS Gönder menüsünden işlem yapıyoruz. Onay alarak işlemi tamamlıyoruz.

Question-1	Menü kimler için açılabilecektir?
Answer-1	Bireysel, Ticari, Tüzel müşterilerimiz için
Question-2	Müşterilere IBAN bilgisini nasıl gönderiyoruz?
Answer-2	SMS olarak
Question-3	Müşteriler IBAN numaralarını nereden öğrenebilir?
Answer-3	IVR üzerinde
Question-4	IBAN kaç haneden oluşmaktadır?
Answer-4	26
Question-5	IBAN neden geliştirilmiştir?
Answer-5	Avrupa Birliği düzenlemeleri çerçevesinde, ülkeler arasında gerçekleştirilen para transferlerinin hızı ile kalitesini artırmak ve maliyetlerini düşürmek amacıyla
Question-6	Cep telefonuna yapılan gönderimlerde ne olacaktır?
Answer-6	şube kodu, hesap no, IBAN bilgileri
Question-7	IBAN nedir?
Answer-7	International Bank Account Number

Table A.3. Example document 3.

Example Document	3
Title	TİCARİ KMH / Hesap İşletim Ücreti İçin KMH Kullanımı
Context	Yürütme durdurma kararına göre gerçek kişi bireysel müşterilerden hesap işletim ücreti alınmamaktadır. Ancak ticari müşteriler bu kapsam dışındadır; dolayısıyla hesap işletim ücreti alınmaya devam etmektedir. Kmh hesaplarından hesap işletim ücreti alınmasına dair koşullar aşağıda belirtilmiştir. KOBİ, KOBİ (+), Tarım ve İşletme işkolları için geçerli olmak üzere, Hesap bakiyesi yeterli ise KMH limitlerinden tahsilat yapılmamaktadır. KMH limiti ve artı bakiye kullanılarak tahsilat yapılabilmektedir. Sadece KMH limiti kullanılarak tahsilat yapılabilmektedir. KMH limitinden kısmi tahsilat yapılabilmektedir. KMH limitinden fazla tahsilat ve limit aşımı yapılmamaktadır. Kredi durumu ' İzleme ' ve ' Normal ' haricinde olan müşterilerin KMH limitlerinden tahsilat yapılmamaktadır. Gerçek kişi müşterilerin KMH limitlerinden tahsilat yapılmamaktadır. Ticari müşterilerde hesap işletim ücreti ile ilgili herhangi bir muafiyet tanımlanamamaktadır. Konu ile ilgili mesaj (şikayet) kaydı açılmamalıdır.
Question-1	KMH limitinden kısmi tahsilat yapılabilir mi?
Answer-1	yapılabilmektedir
Question-2	Kmh hesaplarından hesap işletim ücreti alınmasına dair koşullar nerede belirtilmiştir?
Answer-2	aşağıda
Question-3	Gerçek kişi bireysel müşterilerden hesap işletim ücreti alınmakta mıdır?
Answer-3	alınmamaktadır
Question-4	Kimlerden hesap ücreti alınmamaktadır?
Answer-4	gerçek kişi bireysel müşterilerden
Question-5	KMH limitinden fazla tahsilat yapılabilir mi?
Answer-5	yapılmamaktadır

Question-6	Hesap bakiyesi yeterli ise KMH limitlerinden tahsilat yapılır mı?
Answer-6	yapılmamaktadır
Question-7	Gerçek kişi müşterilerin KMH limitlerinden tahsilat yapılır mı?
Answer-7	yapılmamaktadır
Question-8	Sadece KMH limiti kullanarak tahsilat yapılabilir mi?
Answer-8	yapılabilmektedir
Question-9	KMH limiti ve artı bakiye kullanarak tahsilat yapılabilir mi?
Answer-9	yapılabilmektedir
Question-10	Hesap bakiyesi yeterli ise ne olur?
Answer-10	KMH limitlerinden tahsilat yapılmamaktadır
Question-11	Hesap bakiyesi yeterli ise nereden tahsilat yapılmamaktadır?
Answer-11	KMH limitlerinden
Question-12	Ticari müşterilerde hesap işletim ücreti ile ilgili muafiyet tanımlanabilir mi?
Answer-12	Tanımlanamamaktadır
Question-13	Kredi durumu ' İzleme ' ve ' Normal ' haricinde olan müşterilerin KMH limitlerinden tahsilat yapılır mı?
Answer-13	yapılmamaktadır
Question-14	Kimler kapsam dışıdır?
Answer-14	ticari müşteriler
Question-15	Gerçek kişi müşterilerin neyinden tahsilat yapılmamaktadır?
Answer-15	KMH limitlerinden
Question-16	KMH limitinden fazla limit aşımı yapılabilir mi?
Answer-16	yapılmamaktadır
Question-17	Ticari müşterilerden hesap işletim ücreti alınıyor mu?
Answer-17	alınmaya devam etmektedir
Question-18	Neye dair koşullar aşağıda belirtilmiştir?
Answer-18	Kmh hesaplarından hesap işletim ücreti alınmasına dair
Question-19	Koni ile ilgili şikayet kaydı açılmalı mıdır?
Answer-19	açılmamalıdır
Question-20	Kredi durumu ' İzleme ' ve ' Normal ' haricinde olan müşterilerin neyinden tahsilat yapılmamaktadır?
Answer-20	limitlerinden