School of Information Technology and
Engineering at the ADA University

School of Engineering and Applied Science
at the George Washington University

TEXT SUMMARIZATION IN AZERBAIJANI LANGUAGE

A Thesis
Presented to the Graduate Program of Computer Science and Data Analytics
of the School of Information Technology and Engineering
ADA University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Computer Science and Data Analytics
ADA University

By
Azar Aliyev

April 2023

THESIS ACCEPTANCE

This Thesis by: Azar Aliyev
Entitled: *Text Summarization in Azerbaijani Language*

has been approved as meeting the requirement for the Degree of Master of Science in Computer Science and Data Analytics of the School of Information Technology and Engineering, ADA University.

Approved:

(Adviser)	(Date)
(Program Director)	(Date)
(Dean)	(Date)

ACADEMIC INTEGRITY STATEMENT

"I affirm that this is my own work, I attributed where I used the work of others, I did not facilitate academic dishonesty for myself or others, and I used only authorized resources for my Thesis, per the ADA University Academic Integrity requirements. If I failed to comply with this statement, I understand consequences will follow my actions. Consequences may range from failing the course to expulsion from the program/university and may include a transcript notation."

(Full Name)	(Signature)	(Date: DD.MM.YY)

ABSTRACT

The research aims to answer the question of how to achieve text summarization in the Azerbaijani language. There are two types of text summarization, being extractive and abstractive. Abstractive text summarization is under the spotlight of this paper since it requires much more complex approaches to achieve the goal. Throughout the research, the problems of the models being trained primarily for English are highlighted and the difficulties of adapting them to the Azerbaijani language are discussed. Azerbaijani language contains 32 letters in its alphabet, meaning that there are extra language-specific characters compared to English. It was shown that tokenization plays a vital role in the model having a successful outcome. It was shown that changing the default normalization layer parameters of the tokenizer tuned out to be extremely helpful towards the results. Three various tokenizers were considered for this task, being WordPiece, SentencePiece Byte-Per-Encoding and Byte Level Byte-Per-Encoding tokenizers. The results of all these three tokenizers were analyzed and determined that WordPiece tokenizer gives better results and is much more space efficient due to its nature.

Apart from that, different network architectures were considered for this summarization task. Advantages and disadvantages of RNNs, LSTMs, and CNNs with the addition of attention mechanism were listed and importance of transformers compared to the three architectures was highlighted.

Azerbaijani news dataset was utilized to build the vocabulary and train the model on. BERT model was chosen to create a model for Azerbaijani text summarization. It was observed that BERT model can achieve feasible results even with moderately small amounts of data. Pre-trained multilingual models such as mBERT did not prove to be worthy since it is trained on very small amount of data.

Additionally, T5 and RoBERTa models were also tested for this task, and they did not achieve acceptable results compared to BERT itself. T5 was computationally demanding and required more training to have a result could be considered as successful. RoBERTa, however, was not suitable for summarization task at all.

Keywords: NLP, BERT, Summarization, Azerbaijani, tokenization, T5, RoBERTa, WordPiece, SentencePiece

Table of Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Explanation
NLP	Natural Language Processing
BERT	Bidirectional Encoder Representations from Transformers
OOV	Out-Of-Vocabulary
BPE	Byte Per Encoding
RNN	Recurrent Neural Network
CNN	Convolutional Neural Network
LSTM	Long Short Term Memory
MLM	Masked Language Modelling

1 INTRODUCTION

Text Summarization is about shortening a provided document and making it compact in such a way that the most important aspects of the document are not lost during the process of doing so. This allows for a text which is reduced in size and carries the key points of a document, eliminating the necessity of scanning the whole document to grasp the main ideas out of it. Throughout the last decade, the amount of information produced digitally has grown significantly in terms of its size and this trend is gaining a higher pace as of this moment. This increasing amount of produced data brings some corresponding requirements with itself, such as comprehending the data in a short time span. Automatizing this task is very important and crucial and it is constantly growing in favour due to the fact that the manual handling process of documents is a very time-inefficient and arduous task. Thus, text summarization aims to address these problems. There are two approaches to summarising a given text, being:

- *Extractive Text Summarization*
- *Abstractive Text Summarization*

Extractive Text Summarization is basically about summarizing the text by utilizing the sentences that hold the information of the highest importance. That is to say, the sentences that are used by the summarizer are extracted within the document itself.

Example text: "The weather is unusually hot today. It is 43 degrees Celcius and hot weather will continue for several days."
Summarized text: "The weather is 43 degrees Celcius hot today."

Abstractive Text Summarization, however, is quite different from its counterpart. Instead of referring to existing sentences inside the given document, an abstractive summarizer tries to come up with its own sentences by relying on the skills of the language model. It is what humans do when they are asked to describe a text that they have most recently read.

Summarized text: "43 degrees of Celcius has been recorded for today and this trend will continue for some time"

Therefore, the abstractive text summarization approach is considered to be a natural way of doing this process and this is the main reason behind its growing popularity.
Almost all of the existing text summarizers are focused on English texts and documents and there is no significant work on summarization for most of the other languages and Azerbaijani language is among the list of such languages. Indeed, it can be said that there has been no noteworthy contribution towards Azerbaijani Natural Language Processing methods. However, the increasing rate of data growth demands to work in this field, and there are works currently being conducted. As the support for the Azerbaijani

language in NLP increases, implementation of NLP-related problems also becomes feasible and manageable. Since the number of people working from home is growing day by day (which is also true for Azerbaijan), digital communication and data transfers are causing the capacity of digital data to grow in size. People are sending and receiving a huge amount of emails and texts constantly which indeed creates a requirement for making this process time efficient. Rather than that, online chatbots can also benefit from text summarisation as the amount of processing the user text can be reduced by summarizers and the requirements of the users can be determined much more precisely. This is also a case for Azerbaijan due to the fact that the amount of time spent online by people has increased, especially in recent years and almost all of the everyday tasks are being conducted through the internet. Therefore, the necessity of creating a text summarizer in the Azerbaijani language is higher now compared to the past.

Overall, the significance of a summarizer and its potential use cases and advantages are mentioned below as:

- *Time allocated to reading is cut down by summaries*
- *Summaries facilitate a straightforward selection of documents for specific purposes*
- *Indexing performance is improved via text summarization.*
- *Queries are much more efficiently handled and analyzed by utilizing summarizers*

However, text summarization for Azerbaijani comes with its own limitations regarding the fact that the research on NLP in the Azerbaijani language can be considered insignificant throughout the years. That is to say, the problem emerges for most of the non-English languages since all the existing models and tools created till now have mainly focused on English, leaving other languages out. English has a lot to offer digitally, which is not true for other languages. To obtain the same level of accuracy as for the English language, there is a need for more pre-processing, a little more ingenuity, and a need for more time allocation when processing a language with richer morphology and more flexible word order. A model or tool that is built on English will not have the same power for the remaining ones. Languages coming from the same language family are somehow close to each other in terms of their morphology and syntaxis. That is to say, for example, English comes from West Germanic Language and German shares the same language family with it. Therefore, a better result can be expected for these kinds of languages by utilizing English-focused tools. However, this will also not yield the best result and the requirement for modifying the structure of the models to fit other languages will still remain in play.

The morphology of the Azerbaijani language is almost completely different from English. To start with, the Azerbaijani alphabet consists of 32 letters, 6 more when it is compared to English. Hence, there exist Azerbaijani special characters ("ı", "ç", "ş", "ö", "ğ", "ə") which can become problematic for encoders and decoders, requesting additional handling. Apart from that, the lowercase form of the letter "I" being "ı" and "İ" being "i" should be taken into account during the normalization process of the text instead of "I"-"i" conversion in English. Furthermore, the Azerbaijani language does not possess grammatical gender. Gender pronouns of he and she is just "o" in the case of the

Azerbaijani language. Part of speech comprises nouns, verbs and auxiliary parts of speech, having a composition of nouns, adjectives, verbs, modal verbs, adverbial modifiers, post-positions and so on. Suffixes play a dominant role in the Azerbaijani language as they can alter the world completely (i.e. gəzmək - gəzinti, note drop of "-mək" and addition of "-inti" / English: to walk - a walk). The Azerbaijani language has six distinct general noun-related grammatical categories: case (genitive, dative, accusative, locative, ablative and nominative), possession (-ım4, -m ımız4; -ın4 -ı, -ınız4; -ı4, -sı4, -lar2+ı4), quantity(-lar, -lər) and predicative (-əm2, -ik4; -sən2, -siniz4; -dir4,, dir4+lar2). Thus, the complete difference between the Azerbaijani language from English raises the necessity for creating a language model from scratch.

The development of the Azerbaijani language model itself is not a straightforward task indeed. Performance characteristics of the model strictly and heavily depend on the size of the corpus that would be utilized for the establishment of the language model. In other words, the higher the volume of the corpus the better the performance of the model. However, a large corpus volume basically demands a higher computation power and this can become really problematic depending on the availability of computational resources. A trade-off between performance and time may be necessary to handle the situation to some extent. For the record, the language representation model called BERT (Bidirectional Encoder Representations from Transformers) is a very powerful tool for the pre-training of the language model. Its noteworthy performance level of it comes from the existence of the massive-sized corpus and the availability of computational resources that are enough to handle this size. The corpus for the pre-training process comprised the dataset BookCorpus with 800 million words (English) joined with English Wikipedia having 2500 million words [5]. Two training sessions were carried out on this dataset: one for the BERT base model and one for BERT large model. The completion time of the training processes was as follows:

- *The base model of the BERT was trained on 4 Cloud TPUs, having 16 TPU chips in total. The completion time of the training process was roughly 4 days [5].*
- *The large model of the BERT was trained on 16 Cloud TPUs, having 64 TPU chips in total. The completion time of the training process was roughly 4 days as well [5].*

Therefore, by observing the above-mentioned figures, creating a language model with acceptable performance standards is a time-consuming and power-demanding task. It should be noted that this procedure is required for extractive text summarization and there is no concern about computational power availability and gathering a huge dataset for developing an adequate language model in the case of abstractive text summarization. Abstractive text summarization methods are mainly based on word frequency approaches and do not require the implementation of deep learning methods to have a working concept. To further explain, the task of the abstractive text summarizers is to distinguish and find out the words that carry the information with the utmost importance compared to the remaining words that exist inside the document. One straightforward of achieving this is to consider that the most important words would probably be the ones that come across much more frequently than the others, thus high-frequency words. However, several problems also appear in this case. For example, there may be a word that is used

less frequently but carries important information that should be used inside the summarized output text. The chances of missing this kind of words are extremely high for frequency-based methods. A workaround for that would be to go with graph-based methods. Graph-based methods are based on the Page Rank algorithm and establish a connection between the sentences of the document. Sentences become nodes of the graph and weighted connections between the nodes aid to extract the most important sentences inside the document. Another problem may come with Out-of-Vocabulary (OOV) words since the corpus is unlikely to have all of such words. Abstractive text summarizers may fail to use such words during the summarization process.

The evaluation process of the final model may be considered another problem. It is very difficult to measure the performance of the summarizer with numerical indicators and manual (human) evaluation might be necessary. However, this can be considered a subjective matter as a summarized text might be acceptable for one person and insufficient for another one. An averaged evaluation method has to be thought of to tackle this issue to some degree.

It should be noted that the multilingual BERT model is trained for 104 languages and includes the Azerbaijani language also. However, the corpus utilized to perform the training process for the Azerbaijani language was significantly small in size to offer a performance relatively close to the English model. Therefore, using multilingual models is not going to yield a result that can be regarded as promising. The study on Turkish Text Summarization, which is examined later in this paper, also highlights this issue in a much more detailed way.

2 LITERATURE REVIEW

Many types of research have been conducted related to text summarization in the scope of its both modes (extractive and abstractive) over the years and they possess some useful insights that are extremely important to consider while developing the pipeline for the Azerbaijani language model. Most of the papers focus on the implementation of English models for other languages and experimenting with the performance of multilingual models on non-English languages. Some of them are mentioned and examined below by extracting necessary and crucial findings from the papers. Main questions to address is about determining the proper tokenization method for the Azerbaijani language, finding out the performance of multilingual deep learning models on the languages other than English and learning about which architecture is feasible for this purpose.

The initial investigation was around multilingual language models and their performances in order to be considered for Azerbaijani language. Papers related to abstractive text summarization in Turkish language by utilizing multilingual language models were referred since Azerbaijani and Turkish languages are were similar to each other in terms of their structure. These works firstly emphasizes the recent advancements in the deep learning field by mentioning the improvements on pre-trained models, improving upon issues like semantics and fluency that prevent the creation of summaries of greater quality. Subsequently, the existence of monolingual and

multilingual models of the same model type is also underlined and throughout the research, it was shown that there is a significant performance gap between monolingual and multilingual models, the former one demonstrating better performance [3]. The testing of the monolingual model was carried out with the help of two large-scale Turkish datasets during most of the works (TR-News and MLSum) [3]. This again points out that the availability of a huge corpus is extremely essential to acquire acceptable efficiency. Apart from that, papers also mention and compare various tokenization methods which are used during the pre-processing of the NLP model. Problems similar in the Azerbaijani language were also discussed like the lowercase conversion of the Turkish letter "I" [13] [3]. Some encoding problems were encountered while utilizing the multilingual models [8] as a result of the existence of letters "ğ", "ü", "ç", "ş", "ö" [3]. However, it was indicated that the m-BERT-cased model did not have significant issues regarding the aforementioned problem [3]. Because the tokenization process directly determines the input space and output space of the pre-trained models, the vocabulary and its quantity are crucial. This becomes much more crucial when the input is in a language with a large vocabulary, such as Turkish or Azerbaijani, or when the input is in a morphologically complex language. In the end, the results demonstrate that for the Turkish language, the pre-trained encoder-decoder networks outperform the RNN-based technique (pointer-generator network) [3]. The benefit of merging the two datasets is another discovery for all the pre-trained encoder-decoder models (Combined-TR). The effectiveness of all the models appears to be significantly improved when the number of training samples is increased for the summary-generating task. Several considered models for Turkish summarization are compared with each other (Figure 2.1).

	TR-News			MLSum (TR)			Combined-TR		
Models	R1	R2	RL	R1	R2	RL	R1	R2	RL
LEAD-2	31.37	17.91	26.92	36.32	23.18	31.39	33.61	20.31	28.95
LEAD-3	28.64	16.21	24.07	34.88	22.20	29.45	31.47	18.93	26.51
Pointer generator (See *et al*. 2017)	31.61	18.55	29.57	38.04	25.01	35.70	35.23	22.03	33.04
(Scialom *et al*. 2020)	–	–	–	36.90	21.77	32.60	–	–	–
mT5	**41.13**	**25.75**	37.60	**42.26**	**27.81**	**37.96**	42.49	27.58	38.67
mBART	40.52	25.22	36.80	40.47	26.17	36.22	41.97	26.95	38.08
BERTurk-uncased	40.50	25.24	37.23	41.47	27.31	37.52	42.51	27.62	38.86
BERTurk-cased	41.06	25.60	**37.69**	41.48	27.23	37.66	**42.75**	**27.83**	**39.08**
mBERT-uncased	33.04	14.94	30.42	33.59	15.98	30.51	34.13	15.95	31.20
mBERT-cased	39.73	24.51	36.37	40.27	26.22	36.40	41.20	26.35	37.50

Figure 2.1. Results of various models for Turkish summarization [3]

Transfer of models towards other languages rather than English to fine-tune the model for evaluation in another language is under the spotlight of recent researches [2]. Here, task-specific annotations in one language are employed [4]. A large number of probing experiments were presented to explain this phenomenon. These experiments demonstrate that transfer is possible even in languages written in different scripts, that transfer is most effective between languages that have a similar morphological and topological structure, that monolingual corpora can be used to train code-switching models, and that the model can identify translation pairs [2]. High lexical overlap among languages aids transfer, however, multilingual models can also transfer between languages written in various scripts, having zero lexical overlaps, demonstrating that it captures multilingual representations [4]. It is also demonstrated that transfer is most successful for languages that are typologically similar, indicating that although model's representation is capable of mapping learned structures onto new vocabularies, it does not appear to learn systematically altering those structures to accommodate a target language with different word order. These findings lead the authors of the paper [4] to the conclusion that multilingual models indeed produces multilingual representations, but that some language pairs are consistently underrepresented in these representations. However, works on Turkish text summarization proved that the performance of the multilingual models is not as good as the monolingual model counterparts of the same model. In other words, although multilingual model is capable of language transfer between non-similar languages, the results will not be significant in order to state that the performance of the summarizer is acceptable, especially for the Azerbaijani language.

A reasonable approach to developing the Azerbaijani text summarizer might emerge which is basically training the model on Turkish corpus and fine-tuning the trained model on Azerbaijani corpus. However, the multilingual model produces better results with languages that are similar to each other topologically [4] and it can be argued that Turkish is the most related language to Azerbaijani since both of these languages are coming from the same language family. Despite the fact that these languages are very similar to each other, the differences are also quite a lot. For example, the letter "ə" in the Azerbaijani alphabet does not exist in the Turkish one. Apart from that, some suffixes are separated from the words in the Turkish language which is not true for Azerbaijani, where suffixes are never separated from the words. This can lead to some issues related to the tokenization process of the language model if it is not taken care of. All of these mentioned things stress the importance of developing a language model for the Azerbaijani language from scratch once more in order to acquire better results [13].

Pretraining of language models and the power of transformer architectures is also the factor and point of utmost importance when it comes to abstractive text summarization. Most of the works here [5] claim that the capacity of pre-trained representations is allegedly significantly constrained by current techniques, particularly for fine-tuning methods. The main drawback is that pre-training architectural options are limited due to the unidirectional nature of standard language models, further explained by the fact that such limitations are not ideal for activities at the sentence level and could be disastrous when applied to token-level tasks where it is essential to take into

account context in both directions [5]. The BERT was introduced for the purpose of enhancing existing methods that are based on fine-tuning by suggesting a "masked language model" (MLM) which is a novel pre-training target that BERT suggests in order to alleviate the aforementioned unidirectional restrictions [5]. The goal of the masked language model, which randomly selects some tokens from the input to mask, is to predict a word's original lexical identity solely from its context. Pre-training a deep bidirectional Transformer using the MLM objective, which unlike the left-to-right language model pre-training allows the representation to integrate the left and the right context. A "next sentence prediction" task is also introduced in addition to the masked language model, which together pre-trains text-pair representations. This feature can be manipulated to achieve abstractive text summarization objectives. Overall, the paper demonstrates the following points:

- *The significance of pre-training in both directions for language representations. For pre-training, BERT uses masked language models instead of unidirectional language models to enable deep bidirectional representations [5].*
- *Many highly developed task-specific architectures are no longer necessary thanks to pre-trained representations. BERT outperforms several systems with task-specific designs and is the first fine-tuning-based representation model to attain significant performance on a broad range of tasks, being sentence-level and token-level [5].*
- *BERT proposes new possibilities for myriad NLP tasks thanks to its main feature which is bi-directionality [5].*

Tokenization is another factor that directly affects the performance of the language model as it was mentioned before. To simply restate, tokenization is a pre-processing stage in creating a language model which helps to create a vocabulary for the language. Therefore, a single mistake in this stage can lead to the creation of words for the language that makes no sense in terms of their meaning. Again, the main analysis of works in this area was solely based on the languages that share similar structure, both morphologically and syntactic, with the Azerbaijani language. Since Turkish and Azerbaijani languages are similar to each other to some extent, the claims of the papers related to Turkish can be integrated into the language model creation process for the Azerbaijani language. The main questions that are tried to be answered are as follows:

- *How do various methods of tokenization affect Turkish language modelling in terms of its performance characteristics? [1]*
- *While the vocabulary size is adjusted to balance model size and performance, how does the model's performance alter when using different tokenization techniques? [1]*

By trying to answer the above question, several conclusions can be drawn. Firstly, morphological level tokenizers are compared with factual ones and found to be off somehow acceptable performance. Apart from that, a performance boost is observed in the case of morphological and word-level tokenizers compared to the factual ones. By using 16.6k tokens on RoBERTa-TR-medium the performance variations are measured [1]. WordPiece and BPE are concluded to yield the best results among other tokenization techniques on most of the NLP tasks. That is to say, WordPiece even demonstrates better

performance than BPE except for the news classification part, in which the difference is not that significant to say BPE suits better than WordPiece [1]. The word-level tokenizer performs badly when compared to the BPE, WordPiece, and morphological-level tokenizers, probably because the vocabulary that is provided is not used to its full potential, hence insufficient vocabulary capacity [1]. The Turkish character-level tokenizer performs the worst in the majority of assignments. The explanation could be that larger language models are better able to represent character relationships than medium models, which might not be able to achieve the same result. Performance enhancement with increased vocabulary volume is pointed out for all of the tokenizers [1]. However, the speed of this improvement saturates much more quickly when it comes to BPE and WordPiece since they do not gain from having a broader vocabulary [1]. 16k tokens are enough for them to outperform other tokenizers and this difference shrinks with the increased vocabulary size [1]. Considering the fact that higher vocabulary volume requires much more computational time, whether to use morphological and word-level tokenizers over BPE and WordPiece becomes questionable. The performance enhancement may be impossible after the vocabulary size exceeds 66k tokens due to the rising computational difficulty. As an alternative, the processing power might be utilized to further expand the availability of other resources, such as the number of parameters in the Transformer blocks.

3 RESEARCH METHODOLOGY

As it was aforementioned, there is no significant research that has been carried out regarding text summarization topics in the Azerbaijani language. Based on the previous research that has been conducted on text summarization in non-English languages, similar methodologies followed by the authors of those works are utilized in order to achieve the goal of the task for the Azerbaijani language. That is to say, languages that share similar morphological and syntactic structures with the Azerbaijani language are extremely helpful in terms of defining the main methodology of this research. The main methodology of this research is clearly explained starting from this point of the paper.

The overall goal of the model architecture is to be able to process input sequences and generate acceptable responses which are also in the form of sequential data. Thus, the text summarizer model should accept long sequential inputs and generate a comparatively shorter version of the input that is also sequential. The working principle of this is very similar to language translation models since they also process sequential inputs. However, the main difference here is that in language translation tasks the output may or may not have the same length as the input: it can be longer, shorter or even the same depending on the language. Therefore, the main objective is to modify language translation models and make them suitable for the summarization task.

The research is both experimental and developmental. Experimentation is carried out by referring to various methods and models that can be considered for this task. Apart from that, since there is no significant model and development have been performed in this field for the Azerbaijani language, the research is also considered

developmental due to the fact that a new model is trained from scratch as one of the results of this research.

There are an array of methods that can be utilized to achieve the task related to text summarization in the Azerbaijani language. These methods are examined one by one throughout the paper and their advantages and disadvantages are clearly listed to emphasize the choice of architecture that is going to be used.

3.1 BENEFITS OF TRANSFORMERS

A key component of human intelligence is the capacity to change and manage sequences. All the human learns about the world comes as sequential patterns of sensory input, and everything to interact with a particular thing calls for responsive sequences of thoughts and actions, which can be verbal, physical, cognitive or emotional. That is to say, humans are able to process complex sequences of actions and produce logical and related reactions (or outputs in terms of machine language) - an innate ability. There is an array of factors that influence this process and yet some are even not clear regarding their working principle. Typical backstage processes that are performed during a casual conversation between two persons involve instant interpretation of words and sentences, context recognition, recalling past related conversations and also making a connection between sentences in an ongoing conversation. The list may grow, however, these would be the most typical actions that are performed during a conversation without even realizing it and all these things happen in a blink of an eye. To replicate these qualities from the side of machine intelligence is an extremely difficult task indeed.

3.2 SEQUENCE TRANSDUCTION

Most of the tasks related to machine learning can be interpreted as transforming a given input sequence into an acceptable output sequence. Speech recognition, language translation and text summarization are some examples of that. In terms of speech recognition, the input is a sequence of sounds or sound waves and the output is a textual sequence. When it comes to language translation and text summarization, input and output are both made up of textual sequences. However, the output differs from the input due to its form which will be broken down later on. The process of transforming an input sequence into an output sequence is called sequence transduction. It entails translating an input sequence to an output sequence, which may be longer or have different formatting. Achieving sequence transduction comes with its own challenges since output sequences should be resilient to the distortions and slight changes in input sequences. Other problems that are related to sequence transduction are but not limited to:

- ***Sequence length**: Long sequences can make it challenging to fully grasp the relationships between the input and output sequences and all pertinent information.*
- ***Vocabulary size**: Modeling and processing all the potential variants might be difficult due to the vast number of inputs and outputs that can be produced by a sequence.*

- ***Non-linear relationships**: It might be challenging for models to learn and generalize because the relationships between the input and output sequences can be complex and also non-linear.*
- ***Lack of labelled data**: Tasks involving sequence transduction frequently call for significant amounts of labelled data, which can be expensive and time-consuming to acquire.*
- ***Inference time**: Because of the high computational complexity, inference time might be a barrier in sequence transduction, especially for longer sequences.*

In order for a model to be able to achieve sequence transduction there is a necessity for some kind of artificial memory. The following example clarifies the last statement to some extent:

"Hogwarts Legacy is a new game about the wizarding world. Events in the game take place in the 1800s."

Looking at the example sentence given above, it can be observed that "the game" refers to "Hogwarts Legacy" which is introduced in the first sentence. Thus, while reading the second sentence, the human brain catches the connection between these two parts and it is essential for machines to capture that kind of dependencies and connections between the different parts of the input. Recurrent Neural Network (RNN), Long-Short Term Memory (LSTM) and Convolutional Neural Network (CNN) models are able to achieve this task. However, each comes with its own problems which are discussed further on.

3.3 RECURRENT NEURAL NETWORKS

RNNs possess loops throughout the model network, enabling the persistence of the information. The processing nodes inside the RNN architecture are connected to each other like a sequence, indicating that an input like a sentence can be fed to the input nodes of such a network word-by-word.

Words are processed one by one and each input node receives information about the previous word which is already processed by the network at a particular timestep (Figure 3.1). This type of network is also referred to as a sequence-to-sequence network. The output is generated by passing the information coming from the hidden states to the decoder stage of the network.

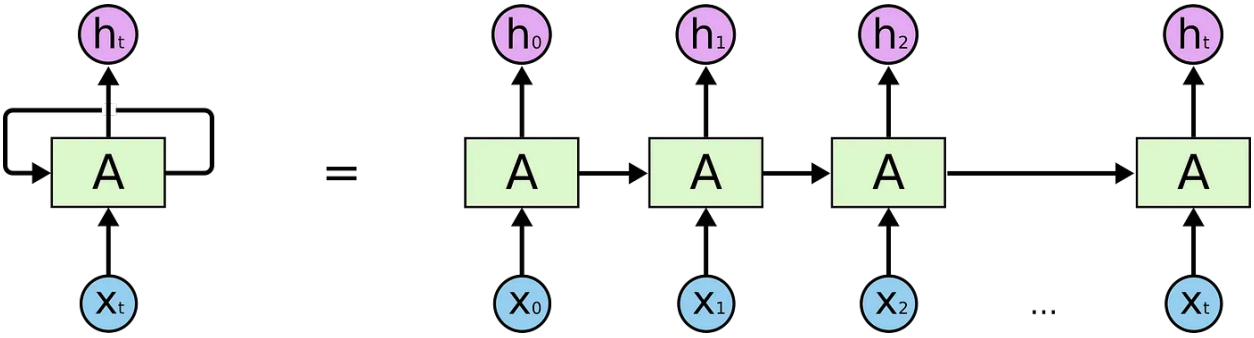


Figure 3.1. A Recurrent Neural Network

Despite the fact that RNNs are suitable for sequential inputs [12], they face a problem while trying to establish and capture long-term dependencies. The problem becomes obvious in the case of next-word prediction. In a sentence like "The moon behind the…" it would be pretty obvious for the model to guess that the word that should be placed instead of three dots is "clouds". There is a pattern like the distance between dependent (relevant) words and the word that has to be predicted is small. However, things become a bit complicated when this gap starts to grow. In a sentence like "I grew up in Baku, I speak fluent…", relying only on the recent information will yield a result suggesting that the missing word is related to a language, which is correct, but which language it is is not included with the recent information. The model has to look deep inside in order to capture the relevant information in this case. The importance of each word in guessing the missing word is figured out with their weights after the training. Therefore, the weights of the past words fade slowly while the model processes the words further on.

The fact that the sequential input members are treated equally is another problem with RNN-based networks [12]. In other words, there is a lack of knowledge regarding the relative relevance of the inputs in an RNN network. At each timestep, the information is changed based on the new data without taking into account the significance or importance of the earlier data.

3.4 LONG SHORT TERM MEMORY

LSTM networks tackle this problem by introducing separate paths for long-term and short-term memories. An LSTM network features four layers to capture the importance of input components at a certain time step (Figure 3.2), as opposed to an RNN network, which only has one layer for each input cell. Each currently processed cell receives two pieces of information, being long-term and short-term memory values coming from the previously processed cell. The path through which the long-term memory flows is referred to as the cell state and it is modified by simple operations, such as multiplication and addition. Not having any weights in order to update this value effectively eliminates the problem of vanishing and exploding gradients which is an issue

for an RNN network. The aforementioned four layers that make up a cell of an LSTM network are:

- ***Forget layer or Forget Gate***
- ***Input layer or Input Gate***
- ***Cell state***
- ***Output layer or Output Gate***

The forget gate (Equation 1) makes a basic decision regarding how much of the long and short-term memory to exclude for the new cell. A value of 0 would basically mean forgetting all of the information coming from the previous state, while a value of 1 would tell the network to remember the previous information thoroughly. The input layer takes the new input value and calculates a value representing the potential memory to remember by taking into account the new input value (Equation 2). This layer also determines how much of the potential memory will be used by the network at the current cell. Last but not least, the output gate generates a new value for short-term memory that is fed into the next cell later on (Equation 3). The percentage values are determined with the help of sigmoid activation functions which output a value between 0 and 1. New states of the memory values are scaled by the tanh activation functions to limit the output between -1 and 1.

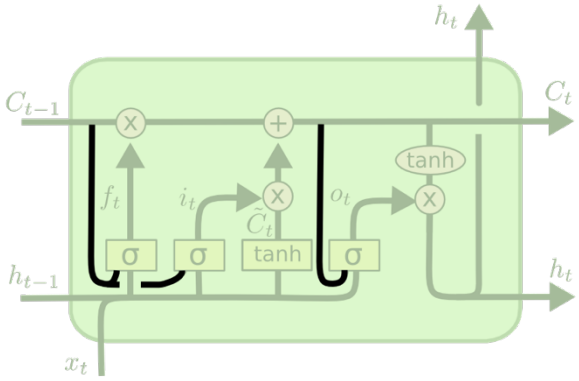


Figure 3.2. Single LSTM cell

$$f_t = \sigma(W_f * [C_{t-1}, h_{t-1}, x_t] + b_f \quad \textbf{(1)}$$

Equation 1: Sigmoid function for forget gate

$$i_t = \sigma(W_i * [C_{t-1}, h_{t-1}, x_t] + b_i \quad \textbf{(2)}$$

Equation 2: Sigmoid function for input gate

$$o_t = \sigma(W_o * [C_{t-1}, h_{t-1}, x_t] + b_o \quad \textbf{(3)}$$

Equation 3: Sigmoid function for output gate

Several LSTM cells are combined to establish a much complex architecture (Figure 3.3).

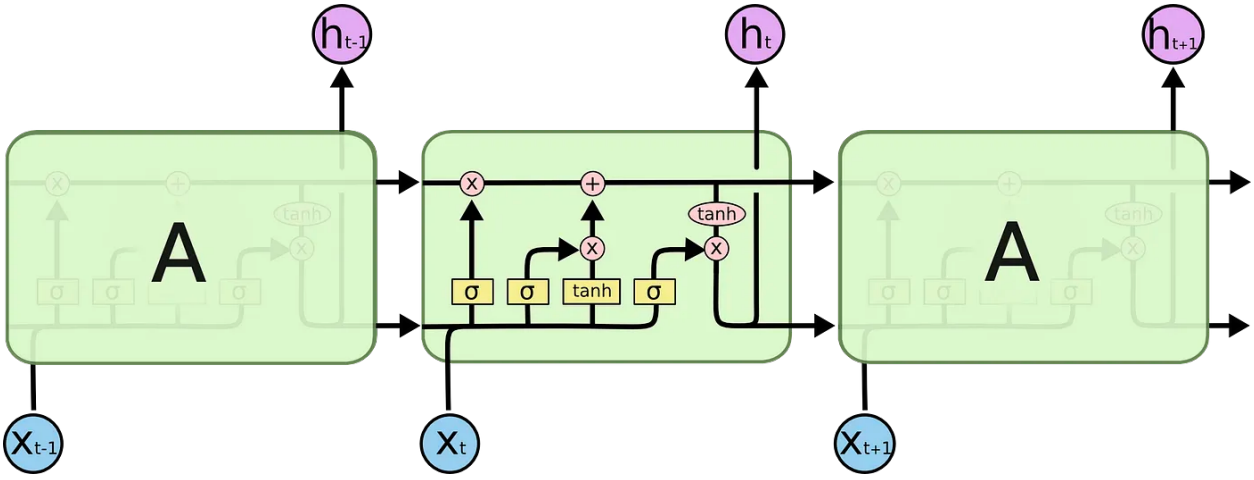


Figure 3.3. LSTM network

When sentences are excessively long, LSTMs still perform poorly, which is the same issue that often affects RNNs [12]. This is due to the fact that, with increasing distance from a word being processed, the likelihood that it will retain context drops dramatically. In other words, the model frequently forgets the content of far-off positions in the sequence when phrases are lengthy. RNNs and LSTMs also have the drawback of making it difficult to parallelise phrase-processing tasks because each word must be processed separately. Furthermore, there is no model that accounts for long- and short-term interdependence. To generalize the problems coming with using LSTM and RNN networks:

- ***Sequential inputs do not allow the model to perform parallel evaluation***
- ***No way of indicating dependencies throughout the input (long and short)***
- ***Distance problem between the dependent word and the word that is being predicted - Linear***

3.5 ATTENTION AND CONVOLUTIONAL NEURAL NETWORKS

In order to compensate for these issues a mechanism referred as attention came into the play. The basic thinking behind this idea is to provide each part of the input (words in case of textual input) with various attention values or weights so that the model would be able to determine the most important words while aiming to predict the missing word. Since a person attends to some words more than others while reading, the establishment of this is essential for machine intelligence to create human-like behaviour. With the addition of this technique, weights are updated by taking into account the attention vector [7]. The input to the following layer of the network is created by computing a weighted sum of the input sequence using these weights.

This method aids to solve the problem of short and long-term dependencies somehow. However, the problem of parallel computation still persists since the words are fed into the network one by one, making it computationally inefficient [7]. Convolutional

Neural Networks (CNNs) are able to achieve this as they do not require processing the previous words in order to be able to perform its tasks on the current word. Apart from that, the distance between the predicted word and the dependent one is of order LogN which was linear in the case of RNNs and LSTMs. LogN comes from the height of the tree that is acquired if the model is unfolded. In spite of the fact that CNNs make it possible to process the input parallelly, they miss interdependencies between the words. Therefore, transformers were introduced, taking the advantage of attention technique and the parallel computation of CNNs.

Transformers, as it was mentioned before, make use of the attention mechanism, however, in a slightly different way, referred to as self-attention. The primary difference between ordinary attention and self-attention is that the self-attention method generates the attention weights solely based on the input data itself whilst an additional context vector is needed for the calculation of attention weights in an ordinary attention method. Self-attention is achieved with the help of three vectors: query, key and value vectors which are going to be explained later on.

Nevertheless, problems like language translation, and text summarization demand working with sequence-to-sequence models since inputs and outputs are sequential there. To increase the efficacy of these processes, parallel computation is a key factor to consider taking into account that these input sequences can be of a length of several thousand words which will take a significant amount of time if they are processed individually, not to forget about the complexity of the models. Apart from that, achieving an accurate result requires capturing the interdependencies between the parts of the input sequence. It was pointed out that CNNs, RNNs and LSTMs face several problems while aiming for an efficient and accurate model by taking into account these details. Transformers combine the characteristics of these network models and adjust them accordingly to suit for the needs of processing the sequential data in most of the NLP tasks.

3.6 TRANSFORMER ARCHITECTURE

It was previously made very clear that the purpose of transformers is to harvest the critical components of RNN, LSTM, and CNN networks in order to reduce their shortcomings. Transformers are therefore created on top of those networks with some significant modifications and changes, which will be covered in more detail as the paper continues to provide an explanation.

The core of the transformer architecture comprises encoder and decoder networks (Figure 3.4). Starting from this point, examples will be provided based on the primary goal of this research, which is text summarization. Since the input and the output are different sequences in this case, there is a necessity for establishing a common "language" for the model in order to be able to map the input into the output. Language translation can be referred to here for the sake of simplifying the explanation. When two persons speaking two different languages natively try to communicate, they need a common language to be able to express their thoughts. For one person, expressions will be subjected to a native-common conversion, whereas the second person will convert this received

expression (which is in the language that is common two both of these persons) into his/her native one. Thus, the working principle of the encoder-decoder network is based on this phenomenon. The input sequence is converted into some different form through a bunch of complex operations and the decoder interprets this received form into a suitable one.

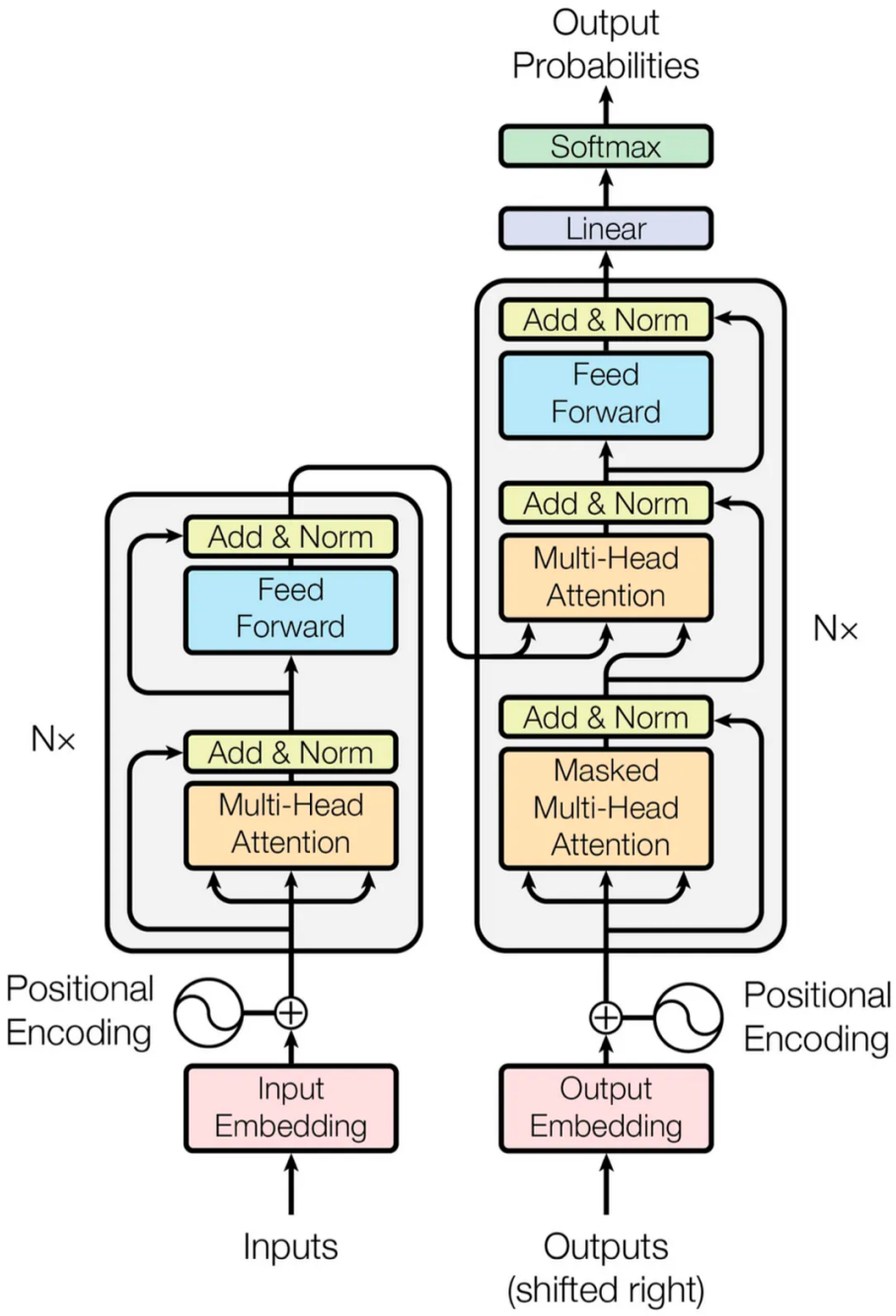


Figure 3.4. Transformer Architecture

Figure 3.4 will be broken down into individual pieces throughout the explanation of the transformer architecture.

Encoder and Decoder are on the left and right sides of the figure respectively. The modules that make up both Encoder and Decoder can be layered on top of one another several times; Nx in the image illustrates this. We can observe that Multi-Head Attention and Feed Forward layers comprise most of the modules. We can't use the inputs directly since they are simply strings, the inputs are first embedded into an n-dimensional space.

As it was previously mentioned, machines have a hard time while trying to understand the words or strings that are being fed into them. Since machines can only think with numbers, these inputs have to be replaced with their numerical representations and these numerical representations cannot be some random numbers. During a summarization process of a text, the importance of synonyms is extremely significant, specifically in the case of abstractive summarization. In other words, being able to express some word groups with a similar word or words is very crucial. Words are being saved in an n-dimensional space for that purpose. Similar words ought to be physically close to one another in this artificially created space and this space is referred to as an embedding space, which has to be either pre-trained or trained by the network itself. The input embedding layer converts the words from the input into vectors by referring to the embedding space. However, the same word can represent different meanings depending on its position in the sentence. Considering two example sentences:

"X's dog is so sweat."
"X looks like a dog."

Here, the contexts the words "dog" refer to are completely different. Therefore, information about the position of the words also needs to be saved and positional encoding comes to the rescue for that purpose. The positional encoding of the various words is small but of the utmost importance for the model. As a sequence depends on the order of its constituents, therefore there is a need to somehow assign every word or component in our sequence a relative position since there is a lack of recurrent networks that can recall how sequences are fed into a model. The embedded representation (n-dimensional vector) of each word is expanded to include these locations.

3.7 MORE ABOUT EMBEDDING SPACE AND POSITIONAL ENCODING

As textual input does not make sense for computers, the conversion of text into some kind of number representation is essential as was aforementioned and embedding space, combined with positional encoding aid to pull this task off. However, there are more ways of performing this action and all of them are going to be analyzed one by one. As an example, the following sentence will be referred to:

"The dog is playing with the toy"

3.7.1 ONE-HOT ENCODING

Looking at the one-hot encoding method, one-hot vectors are created in order to represent the words. For each of the words in the example input sentence, a vector, having a length equal to the size of the available vocabulary, is created. Each word in the input gets a vector created for itself in this way (Table 3.1).

Table 3.1. An example of a One-Hot encoded sentence

	the	*dog*	*is*	*playing*	*with*	*toy*
playing	*0*	*0*	*0*	*1*	*0*	*0*
dog	*0*	*1*	*0*	*0*	*0*	*0*
toy	*0*	*0*	*0*	*0*	*0*	*1*

The encoded words are then placed on top of one another (concatenated) to produce the sentence's actual encoding. Yet, because these representation vectors are sparse, it is acknowledged that this method is inefficient. In other words, a significant issue arises when a large vocabulary is used since a lot of extra information is stored as zeros.

3.7.2 UNIQUE-NUMBER ENCODING

In this technique, each word is presented as a unique number. Considering the example, it contains six unique words, being "the", "dog", "is", "playing", "with", and "toy". Therefore, the unique-number encoding method will yield the resulting vector of [1, 2, 3, 4, 5, 1, 6]. The main problem introduced with the one-hot encoding method is tackled here as there is no more storing of unnecessary information (zeros). This makes the unique-number encoding method more efficient than one-hot encoding. However, this representation method also contains several problems:

- *Random integer encoding is used, not taking into account any dependencies between the words.*
- *It can be difficult for a model to comprehend an integer encoding. For instance, a linear classifier learns a single weight for each feature. This feature-weight combination has no significance because there is no connection between a word's similarity to another word's and how those words are encoded.*

3.7.3 WORD EMBEDDINGS

Word embeddings allow for the employment of a dense, effective conversion of textual data into numerical representation through which similarities and dependencies between the words are captured. A crucial benefit is the elimination of the requirement for manual encoding specification. A dense floating point vector is what is known as an embedding and the length of this vector is a parameter that is specified. These are trainable parameters, avoiding the need for human embedding value specification. Similar to how a dense layer weight learning of a model is performed, a model learns

weights throughout training. Finer associations between words can be captured by a higher dimensional embedding, but it requires more learning data.

Thus, a table of weights is established comprising each of the words inside the available vocabulary (Table 3.2). This table is also known as a look-up table. Whenever the model encounters a word, it tries to look it up in that table and takes the weights of this word for further processing and tasks. By using the word embeddings, the prior two problems of sparse representation and capturing the semantic relationship between the words are handled to a great point. The models can perform better on a variety of NLP tasks by employing word embeddings to help them comprehend the meaning and context of words.

Table 3.2. An example of word embedding (values are random for demonstration purposes)

playing	*-1.2*	*0.2*	*3.1*	*-1.0*
dog	*2.1*	*1.7*	*0.3*	*0.2*
toy	*-0.5*	*0.6*	*1.0*	*3.0*

3.8 POSITIONAL ENCODING

Utilizing word embeddings is not sufficient to capture the information about the relations and positions of words in the sentence. Therefore, in order to compensate for that, positional encodings are used. Positional encodings, as can be guessed from the name, are vectors that are established to save the positional information of each word in a sentence. There are several ways of achieving this in a model and each of them has its own advantages and problems that are clarified by delving into these methods of positional representations.

3.8.1 INDEX WORDS TECHNIQUE

In this method, positional vectors are simply the index of the words that they belong to (Table 3.3). Referring to the example sentence (see previous section), there are six unique words and indices will be from one to six for the words.

Table 3.3. Positional Encoding vector with word indexing method

P0	*P1*	*P2*	*P3*
0	*1*	*2*	*3*
0	*1*	*2*	*3*
0	*1*	*2*	*3*
0	*1*	*2*	*3*

Although this method provides a unique number for different positions, there is one major drawback introduced by it. If the length of the sentence is huge, the emphasis put on the last words will be higher than the other words and when word embeddings are combined with such positional embeddings, the last words will have a huge number added to them, making them feel like most important words in the sentence.

3.8.2 SENTENCE LENGTH FRACTION TECHNIQUE

Previous problem creates a demand for the normalization of the positional data in order to eliminate the dominance of some words due to poor indexing. The idea behind the sentence length fraction technique is to divide these indices by the length of the sentence. The values of the positional vector for each word are calculated using the Equation (4), where N is the length of the sentence

$$P_i = Index * \frac{1}{N-1} \quad (4)$$

The table now becomes (Table 3.4):

Table 3.4. Positional Encoding with Sentence Length Fraction (N=4)

P0	*P1*	*P2*	*P3*
0	*0.33*	*0.66*	*1*
0	*0.33*	*0.66*	*1*
0	*0.33*	*0.66*	*1*
0	*0.33*	*0.66*	*1*

This method introduces a new problem. Since the evaluation is based on the length of a sentence, the same word receives different positional values in sentences with various lengths, taking into account the fact that this word is in the same place in each of the sentences. Even if the values of the positional vectors are normalized in the 0-1 range, this will indeed lead to confusion when the model tries to learn the words in the sentence. Hence, this method is also not desirable for encoding positional information.

3.8.3 FREQUENCY-BASED TECHNIQUE

In order to tackle these problems, a frequency-based approach was adopted to establish positional vector values for each word [7]. The dimension of the embedding space is also included in this method.

$$PE_{(pos,2i)} = sin(\frac{pos}{10000^{\frac{2i}{d}}}) \quad (5)$$

i - i-th dimension
pos - position of the word
d - dimension of the embedding space

This method solves previous two problems in an excellent way. The values are normalized between -1 and 1 and also, and the sentence length dependency is also eliminated. In lower frequencies, the positional values for different words may become the same. However, the positional difference between these words becomes clearer on higher frequencies, which is reflected in the dimensional values of the positional vector. In the original transformer paper, authors have utilized cosine functions additionally and these functions are alternated based on the odd and even position of the words.

$$PE_{(pos,2i+1)} = sin(\frac{pos}{10000^{\frac{2i}{d}}}) \quad (6)$$

Finally, the frequency-based positional encoding vectors are added with word embeddings, finishing with the conversion of textual data into numeric representation for further processing.

3.9 DATA COLLECTION AND PRE-PROCESSING

In order to create a text summarization model for the Azerbaijani language, a good data source is necessary. During the time period of conducting this research, there was a lack of a huge corpus for the Azerbaijani language. It should be noted that acceptable performance is not only achieved by selecting a powerful NLP model. The availability of valuable data is of the utmost importance to be able to make use of the benefits of powerful language models.

The models are trained on the Azerbaijani news dataset that comprises 50000 news, having three types of data for each of the 50000 news (Figure 3.5):

- ***Category***
- ***Title***
- ***News article***

News article columns are passed as the article input to the models that are utilized for the purpose of this research and titles (headlines) were used as target recaps

of this news article inputs. The model tries to learn how to match the news to its headline basically.

The amount of data is not that significant. However, it helps to achieve the goal to some extent, which is described in the results section of this paper.

During the data pre-processing, various steps have been taken. Firstly, by analyzing the dataset it was observed that there was significant amount of mentions of websites as a source of the news. Therefore, strings containing domain names were removed from the dataset. Secondly, most of the articles inside the dataset included reference parts that yields no significant information about the main point of that article. After a meticulous analysis, it was determined that this referencing was following a specific structure (see example below).

Example article from news: "Milli.Az gazetevatan-a istinadən xəbər verir ki, məşhur aktrisanın geyimi gündəm olub***"***(reference: azeri_news_dataset).

The beginning part of the example sentence up to comma usually indicates referencing which can also be determined by the word "istinadən***" (Eng: referring to). Thus, a regex expression to detect sentence parts that includes this word and has a comma after it was taken into account to be removed from the dataset. Apart from that, a frequent words carrying no significant meaning on titles were taken out. These words were 'VİDEO', 'FOTO', 'FOTOLAR', 'YENİLƏNİB', 'ŞOK'. Another important point was to remove the parts that was included inside the parentheses since these parts usually show abbreviations or quick explanations of some specific words or collocations inside the article. Last but not least, all the symbols but numbers and letters (including language specific characters with Unicode representations) were removed and these letters were converted into lowercase since vocabulary built for tokenization also performed the lowercase conversion. This allows to remove most of the unknown tokens in our model. It should be noted that during the lowercase conversion, Azerbaijani capital letter "İ" were not being converted into "i", so a manual conversion was carried out to compensate for this situation.***

11	Dünya	Mərakeşli sə	Mərakeşdən Türkiyəyə gələn sərnişinin mədəsindən 1
12	Maraqlı	Məşhur müğ	Müğənni Sərdar Ortaç məşhur otellərdən birində kor
13	Maraqlı	Banklar üçü	"Kasperski laboratoriyası" antivirus şirkəti 2016-cı i
14	Maraqlı	Açıq məkanc	Art-instalyasiyalar ustası Ayzek Kordel Polşadakı Lo
15	Dünya	Corc Kluni su	Aktyor Corc Kluni və onun həyat yoldaşı Amaliya baz
16	İqtisadiyyat	Forbes: "Bak	ABŞ-ın Fransa və Böyük Britaniyada geniş oxucu kütl
17	İdman	Şenol Günəş	Şenol GünəşTürkiyə millisinin baş məşqçisi olmaqda
18	Maraqlı	“GoPro Quik	"GoPro" şirkəti qısa videoların yaradılması və onları
19	İdman	Tilsim qırıldı	23:00 Bu gün Çempionlar Liqasında III təsnifat mərh
20	Dünya	Venesuelad	Venesuelada aprelin 4-dən bu günədək davam edən
21	Maraqlı	İki “Apple” q	ABŞ-ın "Apple" şirkəti "iPod shuffle" və "iPod nano" r
22	Maraqlı	İnsan bədən	ABŞ, Çin və İsrail alimlər qrupu insan orqanizminə ye

Figure 3.5. Azerbaijani news dataset
(Category - 1st column, Title - 2nd column, News_Article - 3rd column)

3.10 SUMMARIZATION PIPELINE

The main pipeline after considering all the details and necessities for text summarization in the Azerbaijani language is described below (Figure 3.6).

- ***Data Acquisition***
- ***Text processing***
  - ***Cleaning data***
  - ***Tokenization***
    - ***Normalization***
    - ***Pre-tokenization***
    - ***Model (WordPiece)***
    - ***Post-processing***
- ***Implementing the deep learning model***
  - ***BERT language model***
- ***Evaluation***
  - ***Rogue score***

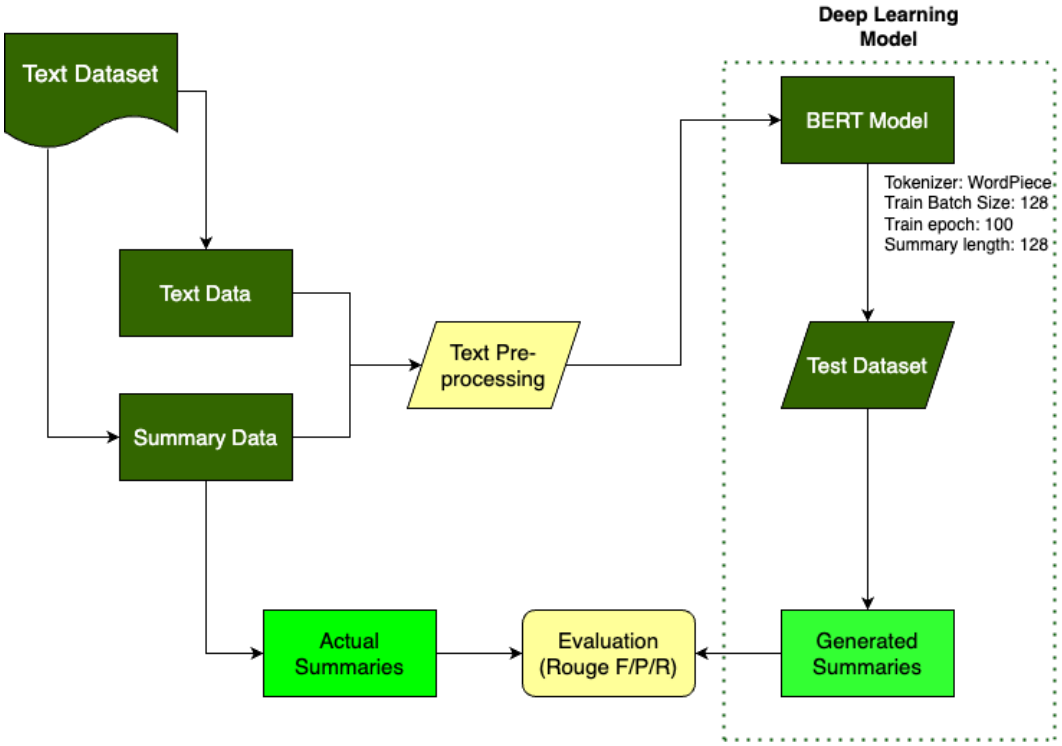


Figure 3.6. Summarization Architecture

To briefly go over the architecture, the text dataset is simply divided into two parts as text data being articles and summary data being titles of that articles. Both of them undergoes a pre-processing stage (see section 3.9 for the details). The pre-processed data is split into training, validation and test sets and once this has been done, training data and validation data is fed into the deep learning model, which is BERT. After finishing the model training, the test dataset is used to generate summaries and a comparison is carried out with actual summaries, revealing the evaluation of the results based on the rouge scores.

3.11 IMPLEMENTATION OF BERT MODEL

BERT - Bidirectional Encoder Representations from Transformers is a powerful natural language processing model that is utilized for a myriad of tasks, such as next word predictor, language translation, summarization and so on. It is purely based on Masked Language Modelling (MLM). In MLM, part of the words in a sentence are hidden, and the model is trained to predict the hidden words based on the context that the other words in the sentence offers. BERT learns to produce contextualized word embeddings by

anticipating the masked words in the input text during the pre-training phase after being trained on a sizable corpus of text. When a word is hidden, the model creates a prediction for the hidden word based on the context of the words around it. BERT gains the ability to recognize intricate connections between words and their context by practising on this task, and as a result, performs well on a variety of NLP tasks. BERT can be fine-tuned for particular NLP tasks like masked word generation or sentence categorization after pre-training. Referring to masked word generation, the model first replaces some of the input words with a special token reserved for masking the word and learns upon predicting these masked words since it is a supervised training process. Sentence completion, language translation, and question-answering are just a few of the uses for this activity.

Various versions of the BERT model make use of a different number of layers which are described below.

- *Base version- 12 layers*
- *Large version - 24 layers*
- *Small version - 8 layers*
- *Tiny version - 2 layers*
- *Mini version - 4 layers*

The architecture of BERT small was recreated with the help of the Keras and Transformers libraries with the parameters specified (Figure 3.7).

- *Number of layers - 4*
- *Token hidden representation size - 128*
- *Hidden layer size - 512*
- *Number of heads - 8*
- *Dropout rate - 0.2*
- *Input vocabulary size - depending on the vocabulary*
- *Target vocabulary size - depending on the vocabulary*

```
[ ]  num_layers = 4
     d_model = 128
     dff = 512
     num_heads = 8
     dropout_rate = 0.1
```

```
[ ]  transformer = Transformer(
         num_layers=num_layers,
         d_model=d_model,
         num_heads=num_heads,
         dff=dff,
         input_vocab_size=VOCABULARY_LENGTH,
         target_vocab_size=VOCABULARY_LENGTH,
         dropout_rate=dropout_rate)
```

Figure 3.7. Establishing Transformer parameters

In order to be able to define the vocabulary sizes of input and target vocabularies, vocabulary from the dataset has to be created first. Making a vocabulary from the dataset involves tokenizing the data appropriately so that our vocabulary includes words and prefixes. In other words, the news dataset has to be tokenized to create the vocabulary, which is one of the data cleaning and preparation steps that is shown on the pipeline of the model part that is utilized for text summarization in the Azerbaijani language.

3.12 TOKENIZATION

Since the model is not able to process textual data directly, there is a need for the conversion of the textual data into numerical representation by using suitable tools and in order to achieve this task, tokenization comes to the rescue. Tokenization simply converts the parts of the textual data (here it is words) into tokens or IDs that are fed into the model network later on. Various tokenization methods exist and each of them

functions on different techniques. However, tokenizing the data follows the same strategy no matter what model is chosen to pull it off. The strategy is mentioned again to recall, which is:

- *Normalization*
- *Pre-tokenization*
- *Training the model*
- *Post-processing*

Normalization is the heart of the tokenization process. It is extremely important for languages that contain many language-specific characters. Azerbaijani language has several language-specific characters that do not exist in English. Since nearly all of the tokenization models are trained for English, the text data in the Azerbaijani language has to undergo the normalization process in order to preserve information and representation of language-specific characters during the encoding and decoding of the tokenized data.

Pre-tokenization is another important step of the tokenization process that directly affects the quality of the output of the model. Pre-tokenization straightforwardly is about breaking the input down into smaller pieces or chunks that are fed into the tokenizer model one by one later on. Parallel execution is not possible with current tokenization models, therefore splitting the input into smaller parts is necessary. In most of the cases, the input is divided into words as smaller chunks. This method is referred to as the whitespace technique. Other techniques, such as punctuation and sentence methods. Pre-tokenization, which specifies the granularity of the units the tokenizer will work on, can have a substantial impact on the performance and accuracy of a tokenizer. When dealing with complicated or ambiguous input text, a well-designed pre-tokenization method can help to increase the effectiveness and accuracy of a tokenizer.

After performing normalization and pre-tokenization steps, the model can be trained for tokenization. There are various models that can be utilized for this purpose. In this case, the WordPiece tokenizer is preferred since in the original BERT paper authors also made use of this tokenizer. It is a type of subword tokenizer that tries to divide words into subwords. The process is overall iterative and the algorithm behind the process is greedy. It starts by splitting all of the input into individual characters and then tries to combine them depending on the likelihood estimation. For instance, the algorithm observes that some subwords are frequent in the input. Therefore, it is more likely that these subwords can be considered core words and prefixes. Prefixes are determined in a simple way. If the frequent subwords are not encountered at the beginning of the words they are considered prefixes. WordPiece tokenizer indicates prefixes with the help of double hashes (##).

One main advantage offered by WordPiece tokenizer is its ability to handle the Out of Vocabulary (OOV) words in a feasible way. It is for sure that datasets are not able to represent all of the words that the language makes use of. Therefore, a strategy is necessary to tackle this kind of state and WordPiece tokenizer approaches this problem in the following way. Since WordPiece tokenizer divides the words into subwords, there is a good chance that these OOV words can be represented as a combination of subwords. This primarily happens while dealing with the proper nouns. For instance, if there is no

learned word to represent "İngiltərə" (name of England in Azerbaijani), it can be shown as "in" "##gil" "##tə" "##rə" only if these subwords exist inside the generated vocabulary. However, in order for this to become possible, the importance of having a huge vocabulary is very critical.

The BERT language model requires the addition of some special tokens in order to be able to pull language processing tasks off. These tokens are:

- *[START]*
- *[END]*
- *[SEP]*
- *[PAD]*
- *[MASK]*
- *[UNK]*

[START] and [END] tokens straightforwardly indicate the start and the end of the input sequences. In BERT, these can also be done with the help of the [CLS] token and this primarily depends on the implementation style of the language model. [SEP] token is utilized to separate sentences inside the input sequence so that the model is able to process them one-by-one instead of processing the whole input at the same time.

The special token used to pad input sequences to a predetermined length during training and inference is referred to as a PAD token in BERT.

Although input sequence lengths can vary, they must all have the same length in order to be processed in batches. In order to accomplish this, the shorter sequences are filled in with the meaningless PAD token, which serves simply as a filler for the input sequences.

In order to prevent the PAD tokens from influencing the model's output, the model learns to ignore them during training. By doing this, the model may be trained well on sequences of various lengths without being predisposed to shorter sequences.

Natural language processing activities that require variable-length inputs, like text classification, named entity recognition, and question answering, frequently make use of PAD tokens.

[MASK] and [UNK] tokens are two most important ones when it comes to training the model for the text summarization task. When the model is trained for the summarization, parts of the input sequence are converted into [MASK] token. For instance, let's say that the following sentence is being fed into the model for during the training.

"Bugün səhər saatlarında havanın yağmurlu keçəcəyi gözlənilir. (Az)"
"It is expected that the weather will be rainy this morning. (Eng)"

The model tries to replace some words with [MASK] tokens.

"[MASK] səhər saatlarında [MASK] havanın yağmurlu [MASK] gözlənilir."

In this manner, model tries to predict the words that can be placed on the position of the [MASK] tokens throughout the training. This kind of models are also referred as

Masked Language Models (MLM). Weights of the models are updated based on the fact that how well model can estimate the actual words in this position, which is determined by calculating the loss of the prediction.

When it comes to [UNK] token, it is utilized when the model encounters a word that is not included inside the vocabulary. This also happens, when subword tokenizers are not able to construct OOV words. Thus, in both of these cases, the unknown words are marked as [UNK] as model does not have any idea about what word it is. In other words, these word is not tokenized by the tokenizer.

3.13 NORMALIZATION

Normalization, as it is aforementioned, is another important step into achieving acceptable results by the language model. Since tokenizers are primarily trained by focusing on English, there are many language specific characters in other languages that has to be processed somehow. This procedure aids in preventing spelling, capitalization, punctuation, and other formatting differences from impairing the analytic or modeling process. Basic normalization steps include:

- *Case adjustment*
- *Punctuation handling*
- *Stop words consideration*
- *Lemmatizing and stemming*

Case adjustment ensures that all input parts are either converted into lowercase or uppercase in order to eliminate capitalization variations. Punctuation handling is about removing commas, dots, periods and etc. from the input. Apart from that, frequent words, which are also referred as stop words, are removed from the input since they do not possess any important meaning. Last but not least, lemmatizing and stemming is carried out to detect the core parts of the words and remove prefixes, which is performed by the WordPiece tokenizer in our case.

Unicode normalization is also the important part of the normalization step while performing the tokenization. Since BERT tokenizer is built on English language as it was aforementioned, unicode normalization has to be interfered in order to make the tokenizer suitable for the Azerbaijani language. The behavior of transforming a Unicode string into a standardized format that enables reliable and accurate text comparisons is referred as "Unicode normalization". Every character used in the majority of the world's writing systems, including Latin, Cyrillic, Arabic, Chinese, and Japanese, has a specific number to represent the character according to the specified Unicode standard. Unicode normalization entails converting similar character sets into a single, standard representation. This procedure aids in ensuring that computer algorithms treat text that appears visually identical to the human eye as being visually identical.

Several types of unicode normalization exists and each of them is analyzed and the choice of the form is explained as follows:

- *NFD*
- *NFC*
- *NFKD*

- *NFKC*

Default normalization form used by the WordPiece tokenizer is NFD. NFD form straightforwardly decomposes the language specific characters into pieces. For instance, letter "ğ" in Azerbaijani language will be decomposed into two parts. The main part of this division will be "g" and the second one will be the hat symbol on the main character. In other words, NFD normalization aims to represent language specific characters with their closest representatives. The results of this normalization form is indicated below for the Azerbaijani language specific characters.

- *ğ - g*
- *ç - c*
- *ş - s*
- *ə - e*
- *ö - o*
- *ü - u*

Thus, initial BERT tokenizer was trying to convert these language specific characters into their closest representatives. This was resulting in loss of information and incorrect vocabulary context, that is discussed throughout the results section of this paper. In order to tackle this issue, the normalization form has to be changed from NFD to NFC (Figure 3.8). In NFC, canonical composition is performed right after canonical decomposition, which is not the case with NFD. Many natural language processing tasks, including as text categorization, sentiment analysis, and named entity recognition, are suited for NFC because it combines characters and diacritical marks in a form that is optimized for display and storage.

```
[ ] bert_tokenizer_params=dict(lower_case=False, normalization_form="NFC")
    reserved_tokens=["[PAD]", "[UNK]", "[START]", "[END]"]
    VOCABULARY_LENGTH = 8000

    bert_vocab_args = dict(
        # The target vocabulary size
        vocab_size = VOCABULARY_LENGTH,
        # Reserved tokens that must be included in the vocabulary
        reserved_tokens=reserved_tokens,
        # Arguments for `text.BertTokenizer`
        bert_tokenizer_params=bert_tokenizer_params,
        # Arguments for `wordpiece_vocab.wordpiece_tokenizer_learner_lib.learn`
        learn_params={},
    )

    START = tf.argmax(tf.constant(reserved_tokens) == "[START]")
    END = tf.argmax(tf.constant(reserved_tokens) == "[END]")
```

Figure 3.8. Code snippet of tokenizer parameter initialization

Overall, three various tokenizers were tested for the Azerbaijani language and the list is provided below:

- *WordPiece tokenizer*
- *SentencePiece Byte-Per-Encoding (BPE) tokenizer*
- *Byte Level BPE tokenizer*

To recall, WordPiece is a type of subword tokenizer. SentencePiece itself is also of the subword tokenizers family. However, BPE type of it is considered in this research. This is a character level BPE tokenizer which makes it different from the third one, being byte level BPE tokenizer. In general, subword tokenization entails dividing words into subword units based on the fact that how many times they are encountered inside the training corpus and the likelihood of their co-occurrence with other subword units, whereas BPE replaces the most often occurring pairings of letters in a corpus with a new symbol. That makes subword tokenizers space efficient since subwords are not contained separately after tokenization. An example is shown for the word "salam" ("hello" in English) considering the best possible tokenization output.

"salam#lar" - an example result of the subword tokenizer, 1 token
"salam", "lar" - an example result of the BPE tokenizer, 2 separate tokens

Another difference between them is the fact that BPE tokenizer does not require a vocabulary to be trained on, which is the case for the subword tokenizers.

In terms of the differences between byte level and character level tokenizers, byte level tokenizer are handling the laguage specific characters better since they are saving those caharacter like a one piece, unlike character level tokenizers which are separating language specific pieces from the main letter itself.

4 RESULTS

The model was trained several times which was possible by saving the checkpoints of the last training process. Whole training was performed in Google Colab environment. Firstly, as it was aforementioned, a vocabulary was built from the training dataset with by also making appropriate input transformations to carry out the preprocessing stage (Figure 4.1).

```
[ ]  %%time
     en_vocab = bert_vocab.bert_vocab_from_dataset(
         ds_train.batch(1000).map(to_lowercase).map(get_transcript).prefetch(2),
         **bert_vocab_args
     )

     CPU times: user 18min 10s, sys: 6.08 s, total: 18min 16s
     Wall time: 18min 28s
```

Figure 4.1. Building the vocabulary

Vocabulary was built upon the training dataset. That is to say, the split of data into training, test and validation sets are as follows:

- ***Overall data size - 50000 news articles***
- ***Training set split percentage - 80%***
- ***Validation set split percentage - 10%***
- ***Test set split percentage - 10%***

4.1 TOKENIZER RESULTS

By utilizing the news articles included inside the train dataset, a vocabulary consisting of 10000 tokens was established. Each of the chosen tokenizers were trained from scratch in order to make it suitable for the Azerbaijani language and the training parameters were kept the same. All of them was defined as requiring a minimum frequency of 2 for the input parts to be considered as a separate token and having a token size of 8000. The results can be observed by referring the Table 4.1.

Example sentence: "Salamlar uşaqlar, necəsiz" (Eng: "Hi guys, how are you")

Table 4.1. Tokenizer results

WordPiece	***"sal##am##lar uşaq##lar necə##siz"***
SentencePiece BPE	***"sal" "am" "lar" "uşaq" "lar" "n" "ec" "ə" "siz"***
Byte level BPE	***"sa" "lam" "lar" "uşaq" "lar" "ne" "cə" "siz"***
Correct representation (WordPiece)	***"salam##lar uşaq##lar necə##siz"***

It was decided that the best result is offered by the WordPiece tokenizer while looking for the language elements' capturing processs of the tokenizers. It has not splitted the first word correctly, however, this is acceptable for subword tokenizer in order to be able to handle the OOV words. A better result can be achieved by increasing the value of

the minimum frequency parameter. This can generate much more precise word splitting, but since the corpus size is relatively small, increasing this parameter can lead to many OOV words that can be difficult for the model to cope with. Therefore, minimum frequency of 2 for WordPiece tokenizer was accepted as a feasible option.

Part of the vocabulary created while training the WordPiece tokenizer for the BERT model can be observed from Table 4.2.

Table 4.2. Part of the vocabulary built by WordPiece tokenizer for BERT model

verilən	bakıda	##lardan	efa	##san
şəkildə	##nun	müzakirə	gedir	məşqçisi
olur	gəlib	etməyə	kredit	qüvvələri
30	ötən	polis	xanım	layihəsi
müəyyən	##er	un	üzərinə	etməsi
##an	keçirilib	##mız	##niz	kürd
bildirir	yerli	bank	ağdam	danışıb
##sə	edərək	ildən	qalan	##ləyib
yerləşən	olunmuş	##lərdə	##ti	valyuta
bmt	##lərdən	altında	##mir	qızıl
iqtisadi	fəaliyyət	dünyanın	##ün	barselona
yoxdur	verilib	başçısı	##es	aşkar
##lərinin	həyat	lazımdır	nümayəndələri	lazım

The next step was to determine maximum number of tokens per example in order for the model to know the size of the input sufficient enough to be able to process the artciles. It was observed that 3438 tokens would be able to cover all of the input articles to fed them into the model (Figure 4.3). However, it is not necessary to cover all of the input articles, a number that will enable to process most of them is sufficient enough since higher number of tokens increases the complexity of the model. Therefore, the model was trained by setting the maximum tokens per transcript variable to a smaller value of 256 tokens. After getting the results, this number was increased gradually to get better results by enabling the model process more input tokens. The final number was 600.

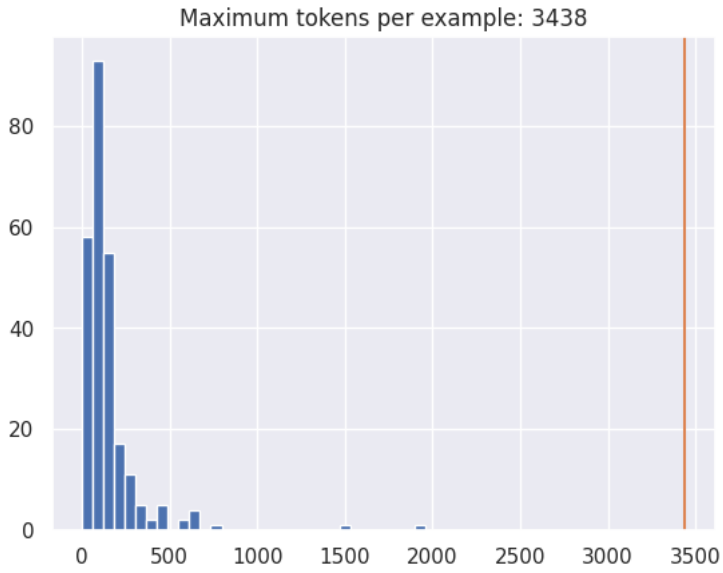


Figure 4.2. Histogram analysis of maximum tokens per example

Before the start of the training, the size of the input tokens will be 602 since it is also necessary to add [START] and [END] tokens to our input. This is performed while creating input batches. Also, the size of the output tokens would be 127 since we drop the [END] token for recap inputs and [START] token for recap labels. Considering the batch size is set to be 64, the shapes are as follows:

- ***Article inputs (64, 602)***
- ***Recap inputs (64, 127)***
- ***Recap labels (64, 127)***

Inputs are trimmed to Max_Tokens_Trancript value during the establishment process of the batches.

After that point, model layers are defined and training is started for 100 epochs. The training was done gradually since the weight of the model were saved as checkpoints in order not to lose information and start the training from scratch during next training. However, the checkpoints has to be deleted if the parameters has to be changed.

The loss was determined based on the masked loss of the model during the prediction. It was mentioned that model replaces some words with [MASK] token and tries to predict that words and masked loss is just about how well the model predicted the masked word when it is compared to the actual word in that position, labels to be said in other words.

Adam optimizer was utilized with the parameters specified below:

- ***Learning rate: With Custom Scheduler, having warmup steps of 40000.***
- ***Beta 1: 0.9***
- ***Beta 2: 0.98***
- ***Epsilon: 1e-9***

4.2 BERT MODEL RESULTS

Firstly, training for the BERT model was done several times since fine-tuning was necessary in order to better the results of the training. During the tokenization process normalization layer was not not changed to NFC in the start. The model was trained in that way, however, as it was aforementioned, Azerbaijani language specific characters were being converted into their closest representatives (see Normalization part). The input sentence utilized in this case is provided.

Input Text #1: (taken from local news page, person names are arbitrary) (Figure 4.3)

Ağstafa rayonunda quldurluq etməkdə şübhəli bilinən şəxslər saxlanılıb.

Daxili İşlər Nazirliyinin Mətbuat Xidmətinin Gəncə regional qrupundan "Report"un Qərb bürosuna verilən məlumata görə, Polis Şöbəsinə rayon sakini olan bir qadının evinə naməlum şəxslərin qanunsuz olaraq daxil olduğu, ev sahibini ölümlə hədələyərək 250 manat nağd pul, 1 ədəd qızıl üzük və 1 ədəd qızıl sırğanı ələ keçirərək quldurluq etmələri barədə məlumat daxil olub.

Rayon Polis Şöbəsinin və Baş Cinayət Axtarışı İdarəsinin əməkdaşlarının birgə keçirdikləri əməliyyat-axtarış tədbirləri nəticəsində cinayəti törətməkdə şübhəli bilinən əvvəllər məhkum olunmuş Pərviz Yolçuyev və Ümid Zeynalov saxlanılıb.

Pərviz Zeynalovun ətrafında davam etdirilən istintaq-əməliyyat tədbirləri nəticəsində onun rayon sakininə məxsus evdən 39000 manat nağd pul eləcə də paytaxtın Abşeron və Suraxanı rayonlarındakı mənzillərdən müxtəlif qızıl-zinət əşyaları da oğurlaması məlum olub.

Faktlarla bağlı araşdırmalar davam etdirilir.

Figure 4.3. Input text #1

Eng: "People suspected of committing robbery were detained in Agstafa district. According to the information provided by the Ganja regional group of the Ministry of Internal Affairs, the Police Department reported that unknown persons illegally entered the house of a resident of the region, threatened the owner of the house with death and took 250 manats in cash, a gold ring and a gold earring. information entered. Parviz Yolchuyev and Umid Zeynalov, who were suspected of committing the crime and previously convicted, were detained as a result of the operation and search measures conducted jointly by the employees of the District Police Department and the Chief Criminal Investigation Department. As a result of ongoing investigative and operational

measures around Parviz Zeynalov, it became known that he stole 39,000 manats in cash from a house belonging to a resident of the region, as well as various gold and jewelry items from apartments in Absheron and Surakhani districts of the capital. Investigations into the facts are ongoing."

The output without NFC normalization was not satisfactory. The model was probably losing information because of the character conversion during the normalization stage. The output can be seen below:

Output #1 (without NFC normalization)
"Qulduqqluq kisi saxlandi ve qadina sxblar cixardi"

After making necessary modifications to normalization layer, the model was trained again from scratch. The results were extremely better in this case, which can be seen below:

Output #1 (with NFC normalization) (Figure 4.6)
"[UNK] külli miqdarda əmlak ələ keçirmək istəyən şəxs saxlanılıb"

Eng: "A person who wanted to seize a large amount of property was detained"

Input Text #2 (Figure 4.4)

Əfqanıstanın paytaxtı Kabildə İraq səfirliyinə yaxınlığında baş vermiş canlı bomba hücumundan sonra təhlükəsizlik qüvvələri və silahlılar arasında atışma baş verib.

Milli.Az Trend-ə istinadən bildirir ki, səfirliyin darvazası önündə canlı bomba özünü partladdıqdan sonra üç silahlı içəriyə daxil olub.

Özünü İslam Dövləti (İŞİD) adlandıran qruplaşma hücuma görə məsuliyyəti öz üzərinə götürüb. Bir neçə saat çəkən atışmadan sonra, Əfqanıstan hakimiyyəti hücumun sona çatdığını və silahlıların öldürüldüyünü deyib.

İraq səfirliyi Kabilin mərkəzi Şər-e-Nav məhəlləsində yerləşir. Bu il şəhər İŞİD və Talibanın həyata keçirdiyi bir neçə ölümcül hücumların hədəfi olub.

Figure 4.4. Input text #2

Eng: "After a live bomb attack near the Iraqi embassy in Kabul, the capital of Afghanistan, a shootout broke out between security forces and militants. Milli.Az reports

with reference to Trend that after a live bomb exploded in front of the gate of the embassy, three armed men entered. The Islamic State (ISIS) claimed the full guilt for the attack. After several hours of gunfire, Afghan authorities said the attack had ended and the militants had been killed. The Iraqi embassy is located in the central Shar-e-Naw district of Kabul. The city has been the target of several deadly attacks by ISIS and the Taliban this year."

Output #2 (with NFC normalization)
"Əfqanıstanda bomba həyəcanı"

Eng: "Bomb scare in Afghanistan"

Generally, the model is able to catch the main points of the input text and summarizes it better. Here, it is observed that there is a [UNK] token in the output of the model. That is because the first word "Ağstafa" is a proper noun, which is not learned during the training. That is not the models fault indeed. Making a search on our created vocabulary, there is no such word that can be encountered. That is primarily the problem with the size of the training corpus that was utilized for this task.

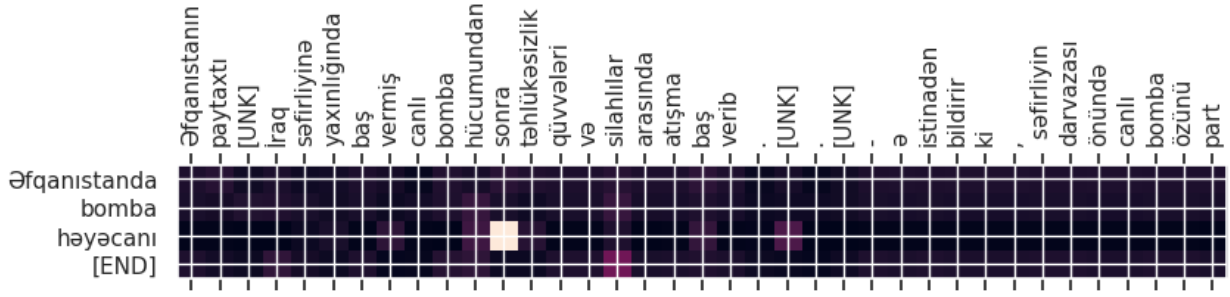
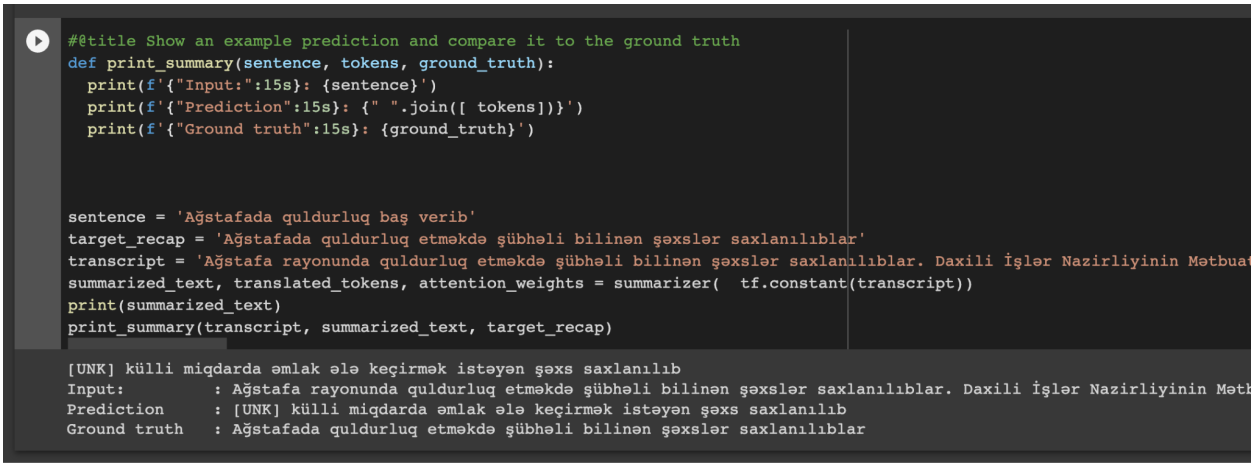


Figure 4.5. Attention plot of example #2



```
#@title Show an example prediction and compare it to the ground truth
def print_summary(sentence, tokens, ground_truth):
  print(f'{"Input:":15s}: {sentence}')
  print(f'{"Prediction":15s}: {" ".join([ tokens])}')
  print(f'{"Ground truth":15s}: {ground_truth}')


sentence = 'Ağstafada quldurluq baş verib'
target_recap = 'Ağstafada quldurluq etməkdə şübhəli bilinən şəxslər saxlanılıblar'
transcript = 'Ağstafa rayonunda quldurluq etməkdə şübhəli bilinən şəxslər saxlanılıblar. Daxili İşlər Nazirliyinin Mətbuat
summarized_text, translated_tokens, attention_weights = summarizer(  tf.constant(transcript))
print(summarized_text)
print_summary(transcript, summarized_text, target_recap)
```

```
[UNK] külli miqdarda əmlak ələ keçirmək istəyən şəxs saxlanılıb
Input:          : Ağstafa rayonunda quldurluq etməkdə şübhəli bilinən şəxslər saxlanılıblar. Daxili İşlər Nazirliyinin Mətb
Prediction      : [UNK] külli miqdarda əmlak ələ keçirmək istəyən şəxs saxlanılıb
Ground truth    : Ağstafada quldurluq etməkdə şübhəli bilinən şəxslər saxlanılıblar
```

Figure 4.6. Output #2 taken from Google Colab notebook

	ROUGE1_precision	ROUGE1_recall	ROUGE1_fmeasure	ROUGEL_precision	\
count	690.000000	690.000000	690.000000	690.000000	
mean	0.310058	0.270467	0.275316	0.292491	
std	0.261556	0.262651	0.247880	0.260928	
min	0.000000	0.000000	0.000000	0.000000	
25%	0.119485	0.095238	0.100641	0.100000	
50%	0.250000	0.196154	0.210526	0.210526	
75%	0.444444	0.363636	0.359744	0.425595	
max	1.000000	1.000000	1.000000	1.000000	

	ROUGEL_recall	ROUGEL_fmeasure
count	690.000000	690.000000
mean	0.256114	0.260277
std	0.264597	0.249641
min	0.000000	0.000000
25%	0.083333	0.086957
50%	0.166667	0.181818
75%	0.333333	0.342857
max	1.000000	1.000000

Figure 4.7. Rogue scores

Figure 4.7 illuminates the results based on the Rogue score values. Utilized scores are based on overlap due to unigrams and longest common subsequences. Precision, recall and F1-scores can be observed from the table. Overall, 690 test units were referred to generate evaluation numbers. Highest mean value achieved is 0.31. 75% of the observations has a maximum of 0.44 Rouge1 precision, which is very close to RougeL precision score.

Higher precision and recall levels typically signify that the model-generated summary and the actual summary are more similar, which means that the summary generated by the model accurately reflects the main points and meaning of the actual summary. However, it must be pointed out that lower Rouge scores does not necessarily mean that the model is performing badly. As the task is abstractive text summarization, the output can be totally different since synonyms and similar words are utilized by the model to capture the main meaning provided inside the input text. Looking at the results, it can be empathized that the model is able to generate good summaries regarding the fact that rogue scores are not that significant. In other words, human evaluation is necessary here to further confirm the quality of the results.

5 SUMMARY AND FEATURE WORK

Text summarization in the Azerbaijani language aims to investigate the works done in this topic and also tries to find out how models built for the English language performs for the case of the Azerbaijani language.

Firstly, the morphological and syntactic structure of the Azerbaijani language was explained, emphasizing on the points of main differences between Azerbaijani and English languages. Abstractive text summarization was chosen as the main goal of this research. In the beginning, multilingual models were mentioned and by referring to the papers and works conducted related to abstractive text summarization in other languages, for example Turkish, it was determined and observed that multilingual models (ones that also works for Azerbaijani language) was not able to perform well due to the limited data utilized during the training process.

Apart from that, the importance of tokenization process were underlined for the Azerbaijani language. Different techniques was analyzed and again by also comparing with the Turkish language tokenization methods, ones that might produce better results were chosen for testing purposes. These tokenizers were:

- *WordPiece tokenizer*
- *SentencePiece BPE tokenizer*
- *Byte Level BPE tokenizer*

It was observed that WordPiece tokenizer performed well in comparison with the other ones for the Azerbaijani language. By making necessary modification to it, it was trained from scratch and fed into the model for the next stage.

Several architectures and methods were discussed throughout the paper which could be considered helpful for the summarization task. The advantages and disadvantages coming with RNNs, LSTMs, and CNNs were clearly pointed out and it was shown why Transformer architecture is the most suitable one for this task. To briefly go over it, transformers are able to process words in input parallelly and capture the word dependencies and positional importances of them inside the input much better compared with other methods.

Overall, BERT model was able to give decent results when it is utilized for the task of text summarization in the Azerbaijani language. The results can be seen on results section.

The model has problems related to the size of corpus utilized for training. Since the dataset used for the purposes of this research were not huge, [UNK] tokens were being observed in some cases. Hyperparameters of the model can be modified more to be able to get much better results. Also, the size of the output summary is small, which is primarily due to the summary labels used during the training. Since the titles of the news articles were utilized as summary labels, they are smaller relatively as it is for the news headlines. Apart from that, the dataset only contained inputs related to news, therefore the results are news biased. Other input sources can be used in order to make model suitable for most of the environments.

REFERENCES

[1] Toraman, C., Yilmaz, E. H., Şahinuç, F., & Ozcelik, O. (2022). Impact of Tokenization on Language Models: An Analysis for Turkish. *arXiv*. https://doi.org/10.48550/arXiv.2204.08832

[2] de Vries, W., & Nissim, M. (2020). As Good as New. How to Successfully Recycle English GPT-2 to Make Models for Other Languages. *arXiv*. https://doi.org/10.18653/v1/2021.findings-acl.74

[3] Baykara, B., & Güngör, T. (2022). Turkish abstractive text summarization using pretrained sequence-to-sequence models. *Natural Language Engineering*, 1-30. doi:10.1017/S1351324922000195

[4] Pires, T., Schlinger, E., & Garrette, D. (2019). How multilingual is Multilingual BERT? *arXiv*. https://doi.org/10.48550/arXiv.1906.01502

[5] Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv*. https://doi.org/10.48550/arXiv.1810.04805

[6] Gupta, S., & Gupta, S. K. (2019). Abstractive summarization: An overview of the state of the art. *Expert Systems with Applications, 121*, 49-65. https://doi.org/10.1016/j.eswa.2018.12.011

[7] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention Is All You Need. *arXiv*. https://doi.org/10.48550/arXiv.1706.03762

[8] Can, Çağdaş & koşaner, Özgün & Aktaş, Özlem. (2016). A Survey to Text Summarization Methods for Turkish. International Journal of Computer Applications. 144. 23-28. 10.5120/ijca2016910358.

[9] Allahyari, M., Pouriyeh, S., Assefi, M., Safaei, S., Trippe, E. D., Gutierrez, J. B., & Kochut, K. (2017). Text Summarization Techniques: A Brief Survey. *arXiv*. https://doi.org/10.48550/arXiv.1707.02268

[10] Choe, D., Constant, N., Guo, M., & Jones, L. (2018). Character-Level Language Modeling with Deeper Self-Attention. *arXiv*. https://doi.org/10.48550/arXiv.1808.04444

[11] Fedus, W., Goodfellow, I., & Dai, A. M. (2018). MaskGAN: Better Text Generation via Filling in the_____. *arXiv*. https://doi.org/10.48550/arXiv.1801.07736

[12] Graves, A. (2012). Sequence Transduction with Recurrent Neural Networks. *ArXiv*. /abs/1211.3711

[13] Ay, B., Ertam, F., Fidan, G., & Aydin, G. (2023). Turkish abstractive text document summarization using text to text transfer transformer. *Alexandria Engineering Journal, 68*, 1-13. https://doi.org/10.1016/j.aej.2023.01.008

[14] Song, S., Huang, H. & Ruan, T. Abstractive text summarization using LSTM-CNN based deep learning. *Multimed Tools Appl* **78**, 857–875 (2019). https://doi.org/10.1007/s11042-018-5749-3