Ministry of Science and Higher Education of the Republic of Kazakhstan

Suleyman Demirel University

Alua Bilakhanova

Kazakh language based Question Answering system using Deep Learning Approach

THESIS

Presented in Partial Fulfilment for the

Master of Technical Sciences Degree in Computer Science

(degree code: 7M06102)

Department of Computer Science

Faculty of Engineering and Natural Sciences

Supervisor: **PhD, Assist. Prof. Nazerke Sultanova**

Kaskelen 2023

Suleyman Demirel University
Faculty of Engineering and Natural Sciences
Department of Computer Science

Dean of Faculty

PhD, Associate Professor

Zhamanov A.

06 06 2023

Topic of the thesis:

Kazakh language based Question Answering system using Deep Learning approach

Thesis submitted as part of the requirements for the award of the MSc in "7M06102 - Computer Science" SDU, 2021-2023

Head of Department Assistant Professor, PhD Mukash Zh.

Academic Supervisor Assistant Professor, PhD Sultanova N.

Master student Bilakhanova Alua

Kaskelen 2023

Declaration

I confirm that this is my own work and the use of all material from other sources has been properly and fully acknowledged.

Alua Bilakhanova

2023

Acknowledgements

I would like to extend my heartfelt gratitude and appreciation to all individuals who have made valuable contributions to the completion of this thesis. Their unwavering support, guidance, and encouragement have been of immense value and significance throughout this transformative journey.

First and foremost, I extend my deepest thanks to my supervisor, Assist. Prof. Nazerke Sultanova, for her continuous guidance and unwavering support. Her expertise, patience, and constructive feedback have played a crucial role in shaping this thesis. I am truly grateful for her mentorship and for pushing me to achieve my best.

I would also like to acknowledge the faculty and staff of Suleyman Demirel University, whose commitment to excellence in education and research has created an environment conducive to intellectual growth.

My heartfelt appreciation goes to my colleagues and classmates who have been a constant source of inspiration and motivation. The insightful discussions, fruitful brainstorming sessions, and valuable collaborations have significantly contributed to the development of my ideas and research.

Dedication

This thesis is dedicated to:

My family, friends for their unwavering support, love, and encouragement throughout my academic journey. Their belief in my abilities and their sacrifices have been my driving force. I am truly fortunate to have such a loving and understanding family.

Abstract

The development and evaluation of question answering (QA) systems in specific languages present unique challenges, and the Kazakh language is no exception. In this study, we address these challenges by developing and evaluating Kazakh language-based QA systems using different deep learning (DL) models. Our objective is to enhance the accuracy and effectiveness of QA in the Kazakh language. We conduct various experiments using translated datasets and implement different models, including BERT, LSTM baseline, and End-to-End Memory Networks (MemN2N). Through rigorous performance evaluation and comparative analysis, we identify the most suitable models for achieving high accuracy in Kazakh language-based QA tasks. It was observed that the MemN2N model achieved the best average accuracy among the three models implemented. This research contributes to the advancement of Kazakh language processing and aims to enhance the accessibility and usability of QA systems in the Kazakh language.

Аңдатпа

Нақты тілдерде сұрақтарға жауап беру жүйесін (QA) әзірлеу және бағалау бірегей қиындықтармен ұштасады, ал қазақ тілі де ерекшелік емес. Бұл зерттеуде біз бұл қиындықтарды терең оқытудың әртүрлі үлгілері мен әдістерін пайдалана отырып, қазақ тіліне негізделген QA жүйелерін әзірлеу және бағалау арқылы шешеміз. Біздің мақсатымыз-қазақ тіліндегі QA дәлдігі мен тиімділігін арттыру. Біз аударылған деректер жиынтығымен әртүрлі эксперименттер жүргіземіз және BERT, LSTM baseline және End-to-end Memory Network (MemN2N) сияқты әртүрлі модельдерді енгіземіз. Өнімділікті мұқият бағалау және салыстырмалы талдау арқасында біз қазақ тіліндегі QA міндеттерінде жоғары дәлдікке қол жеткізу үшін неғұрлым қолайлы модельдерді анықтаймыз. MemN2N моделі іске асырылған үш модельдің ішінде ең жақсы орташа дәлдікке қол жеткізгені байқалды. Бұл зерттеу қазақ тіліндегі деректерді өңдеуді дамытуға үлес қосады және қазақ тіліндегі QA жүйелерінің қолжетімділігі мен пайдалану ыңғайлылығын арттыруға бағытталған.

Аннотация

Разработка и оценка систем ответов на вопросы (QA) на конкретных языках сопряжены с уникальными проблемами, и казахский язык не является исключением. В этом исследовании мы решаем эти проблемы путем разработки и оценки систем QA на основе казахского языка с использованием различных моделей и методов глубокого обучения. Наша цель - повысить точность и эффективность QA на казахском языке. Мы проводим различные эксперименты с использованием переведенных наборов данных и внедряем различные модели, включая BERT, LSTM baseline и End-to-end Memory Network (MemN2N). Благодаря тщательной оценке производительности и сравнительному анализу мы определяем наиболее подходящие модели для достижения высокой точности в задачах QA на казахском языке. Было замечено, что модель MemN2N достигла наилучшей средней точности среди трех реализованных моделей. Это исследование вносит вклад в развитие обработки данных на казахском языке и направлено на повышение доступности и удобства использования систем QA на казахском языке.

Abbreviations

BERT Bidirectional Encoder Representations from Transformers

CNN Convolutional Neural Networks

CSV Comma-Separated Values

DL Deep Learning

FFNN Feed-forward Neural Networks

GPT Generative Pre-trained Transformer

GRU Gated Recurrent Unit

LSTM Long Short-Term Memory

MemN2N End-to-End Memory Network

ML Machine Learning

MLM Masked Language Modeling

MLP Multi-Layer Perceptron

NLP Natural Language Processing

NSP Next Sentence Prediction

QA Question Answering

RNN Recurrent Neural Network

Table of Contents

Chapter 1

Introduction

1.1 Background and Motivation

The development of question-answering (QA) systems based on deep learning (DL) has gained significant interest in the field of artificial intelligence (AI) in recent years. The present study focuses on Natural Language Processing (NLP) tasks that involve designing computer systems capable of comprehending and answering questions posed in natural language. The evolution of these systems can be attributed to the progress made in Data Mining (DM), Machine Learning (ML), and DL technologies. The increasing value of these advancements has been observed in various industries including education, business, and others.

The success of deep learning techniques in the domain of natural language processing has opened up avenues for the advancement of QA systems that can furnish precise and dependable responses to intricate queries. The achievement of these systems is facilitated through the utilization of substantial quantities of structured and unstructured data for both training and output purposes. AI systems possess the ability to process a wide range of question formats and interpret the intricacies of natural language, allowing them to furnish informative answers.

As the field of AI progresses, it is anticipated that QA systems utilizing deep learning methodologies will undergo enhancements, resulting in increased sophistication and efficacy. The potential of these systems to transform human-machine

interaction and information accessibility lies in their ability to continuously learn from vast amounts of data. The aforementioned phenomenon presents novel prospects for enhancing diverse facets of our everyday existence.

However, there is still a lack of available data and limited experience in training and using these systems for different languages. Despite some efforts made to develop multilingual models. This issue is particularly evident for low-resource languages such as Kazakh, which has a rich linguistic heritage and unique features, grammar, and vocabulary. The complexity of the morphological structure of the Kazakh language requires sophisticated algorithms and techniques to accurately process its agglutinative nature and derivational morphology. Furthermore, the limited availability of linguistic resources, such as annotated data and language tools, further exacerbates the challenges of developing NLP systems for the Kazakh language. Therefore the development of QA systems for the Kazakh language remains relatively limited compared to other widely spoken languages.

Existing QA systems often rely on rule-based or statistical approaches, which may not fully capture the complexity and nuances of the Kazakh language. This creates a research gap that motivates the need for a DL-based approach to developing a QA system that can accurately understand and process the Kazakh language.

This work is motivated by a desire to contribute to the development of QA systems in the Kazakh language employing cutting-edge DL models. The proposed research will investigate the feasibility of DL models such as Long Short-Term Memory (LSTM), End-to-end Memory Networks (MemN2N), and BERT (Bidirectional Encoder Representations from Transformers) for implementing an effective and precise QA system for the Kazakh language. This study aims to fill a gap in the existing literature and shed light on the application of advanced NLP techniques to Kazakh language processing by leveraging the power of DL.

In summary, the background and motivation for this study highlight the limited development of QA systems for the Kazakh language and the need for advanced DL-based approaches. Motivation is based on the importance of promoting the Kazakh language in the digital era and the possibility of practical applications.

In addition, this research can contribute to the preservation and popularization of the Kazakh language, thereby fostering its modernization.

1.2 Aims and Objectives

The aim of this study is to create a proficient and precise QA system for the Kazakh language by utilizing DL techniques. The objective of this study is to leverage the capabilities of deep neural networks to tackle the unique challenges posed by the agglutinative nature and complex derivational structure of the Kazakh language. This study seeks to enhance the development of NLP for Kazakh by investigating and modifying cutting-edge techniques.

The specific objectives of the study are:

- Developing a QA system for the Kazakh language using DL models;
- Implementing and evaluating the performance of three DL models: LSTM-baseline, MemN2N, and BERT, for QA on Kazakh language datasets;
- Comparing the performance of the models based on accuracy and F1 score on the different Kazakh language datasets;
- Providing insights into the effectiveness of DL models for Kazakh language QA systems.

This thesis will concentrate on the design and implementation of a Kazakh language-specific DL-based QA system in order to achieve these research objectives. This system's performance will be evaluated on diverse datasets, and its capabilities will be analyzed in depth.

1.3 Thesis Outline

The rest of this thesis is organized as follows:

- Chapter 2 provides a comprehensive literature review on QA systems, DL techniques for NLP and QA systems, and existing Kazakh language-based QA systems.

- Chapter 3 presents the methodology employed in this research, including data collection and pre-processing, DL model selection, architecture, training, and evaluation.
- Chapter 4 presents the experimental results of the proposed system, including comparative analysis, and discussion of results.
- Chapter 5 discusses the findings and implications of our study, highlights key insights gained from the experiments, and provides a summary of the research findings, contributions to the field, and future research directions.

Chapter 2

Literature Review

2.1 Overview of Natural Language Processing and Question Answering

In today's era of digitalization, NLP and QA have become increasingly important due to the large amounts of data being generated, particularly in the form of user-generated content on social media platforms such as posts, comments, and messages. This data can be analyzed using NLP techniques to extract insights like sentiment analysis and topic modeling that can help organizations make better decisions and improve consumer experiences.

NLP is a branch of AI that focuses on how computers and human language may interact naturally. It makes it possible for computers to understand, perceive, and produce human language in a way that is meaningful to people. NLP has several applications in a variety of fields, such as sentiment analysis, speech recognition, and QA systems. This dynamic and rapidly developing field is pushing the limits of artificial intelligence by making significant progress toward enabling computers to interact with human language properly.

QA is a crucial NLP application that enables machines to find accurate answers to natural language questions asked by humans. It involves interpreting the question's significance and producing a precise response. Real-world applications for QA systems include customer service, online information retrieval, and

educational tools.

QA systems can be broadly classified into two categories based on the domain [1]:

- **Open-domain QA systems:** These systems are designed to answer questions on any topic, regardless of the domain. They typically use large-scale knowledge bases and advanced information retrieval techniques To find relevant information for a given question. Open-domain QA systems are often used for general information-seeking tasks and are typically more challenging to develop compared to closed-domain systems.
- **Closed-domain QA systems:** These systems are designed to answer questions within a specific domain, such as medicine, law, or finance. They are typically easier to develop than open-domain systems because the scope of the questions and the domain-specific knowledge required is more limited. Closed-domain QA systems can be more accurate than open-domain systems because they can leverage specialized knowledge and language specific to the domain.

QA systems can also be classified based on the type of questions they can answer [1]:

- **Factoid systems** are designed to answer questions with a single, objective answer, such as "What is the capital of France?" or "What is the boiling point of water?".
- **Non-factoid systems**, on the other hand, are designed to answer questions that require more complex, subjective answers, such as "What is the meaning of life?" or "What is your opinion on climate change?".

Developing QA systems is a challenging task, as it requires addressing a number of key challenges. One major challenge is handling natural language ambiguity. As a single question can have multiple possible interpretations. Another challenge is dealing with the large amount of information available. Since QA systems must be able to efficiently search and analyze large amounts of data. Furthermore, the accuracy and reliability of QA systems can be affected by biases in the data or

algorithms used to develop the system.

QA systems have the potential to revolutionize the way people interact with information and improve efficiency in a variety of applications. However, developing accurate and reliable QA systems requires addressing a number of key challenges and incorporating advanced NLP techniques.

2.2 Deep learning techniques and their application in NLP and QA

Deep learning is a subset of machine learning in which artificial neural networks are used to learn and predict data (Figure 2.1). It excels in processing large amounts of complex data with high-dimensional features. In DL, multiple layers of interconnected nodes, referred to as artificial neurons, are employed to extract meaningful features from the data and make predictions based on those features [2]. By leveraging these hierarchical representations, DL models have demonstrated remarkable success in capturing intricate patterns and relationships within diverse datasets.

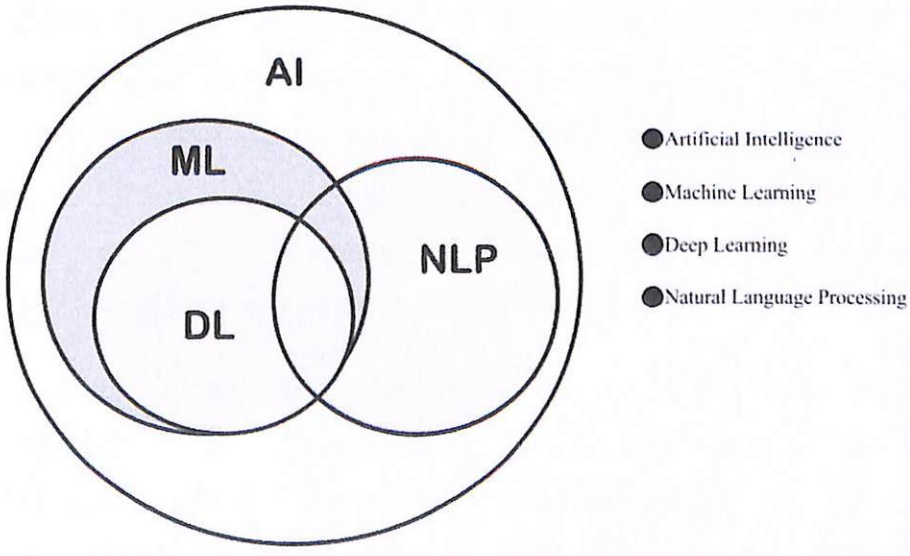

Figure 2.1: NLP is a branch of AI and uses ML/DL techniques [3].

One of the key differences between traditional ML and DL is the level of abstraction in feature engineering. Feature engineering is a crucial and time-consuming

step in traditional machine learning that requires a domain expert to manually extract and engineer the relevant features from the data. On the other hand, DL can automatically learn features from raw data, allowing it to handle a wider range of tasks without requiring as much domain expertise.

DL models have had a significant impact on the field of NLP. Particularly neural networks have shown enormous promise in improving the precision and effectiveness of various NLP tasks. This section will discuss the DL models that are applied in NLP.

2.2.1 Deep learning in NLP

Multi-layer Perceptron (MLP)

The Multi-layer Perceptron (MLP) is a type of artificial neural network model that is composed of interconnected perceptrons arranged in multiple layers [4]. The perceptron model involves the computation of a weighted sum of input values, which is then passed through an activation function to produce an output. The inception of the model can be traced back to the late 1950s and early 1960s, when a group of researchers, namely Frank Rosenblatt, Bernard Widrow, and Arthur Samuel introduced the model [5, 6, 7]. The model was limited in its adoption prior to the 1980s due to computational constraints [4]. The development of the backpropagation algorithm enabled the training of this model and led to its increased usage. The model has gained significant popularity and utility in the field of machine learning. It is employed for various purposes, including but not limited to image recognition, speech recognition, and NLP.

The model's fundamental architecture comprises of an initial layer for input, one or more hidden layers for processing, and a final layer for output (Figure 2.2). The composition of each layer involves the integration of numerous neurons that execute a weighted summation of the inputs, succeeded by a non-linear activation function. During the training process, the connections between neurons are modified through backpropagation. This technique involves calculating the gradient of the error in relation to each weight and subsequently adjusting them using gradient descent. The model exhibits a notable advantage in its capacity to

acquire intricate non-linear associations between inputs and outputs, rendering it appropriate for tasks that are not readily solvable using linear models.

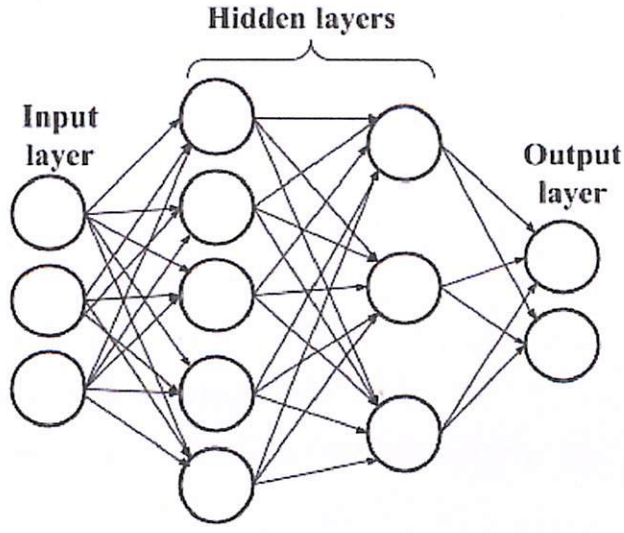

Figure 2.2: Multilayer Perceptron with two hidden layers [8].

Some of the applications of MLPs in NLP include language modeling, sentiment analysis, named entity recognition, and machine translation.

Convolutional Neural Networks (CNN)

The Convolutional Neural Network (CNN) model was initially developed by Yann LeCun and his team in the 1980s to address image recognition tasks [9]. The contemporary convolutional neural networks (CNNs) that are currently utilized, comprising of layered convolutional, pooling, and fully connected components (Figure 2.3), were formulated during the period of the late 1990s and early 2000s. The utilization of CNNs for image recognition tasks has gained popularity, particularly due to the significant contribution of Krizhevsky et al. in 2012 [10]. Their work on winning the ImageNet Large Scale Visual Recognition Challenge using CNNs is considered a breakthrough in this field. Since then, CNNs have been widely employed in various computer vision and image processing tasks.

Furthermore, CNNs have been found to be effective in multiple natural language processing (NLP) tasks, such as sentence classification. The CNN applies convolutional filters to a matrix of word embeddings, with the aim of capturing localized features within the sentence. Following the generation of feature maps,

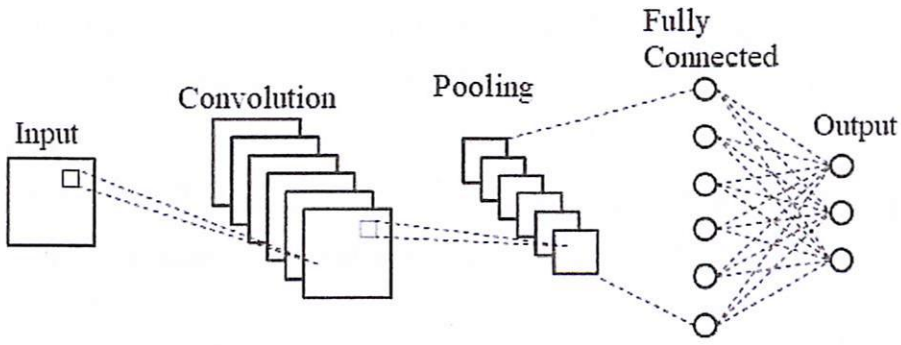

Figure 2.3: Basic CNN Architecture [11].

a max-pooling process is applied to obtain a feature vector of a predetermined length. The vector obtained is subsequently inputted into a fully connected layer to perform the classification task. The technique was first introduced by Yoon Kim in 2014 [12].

CNNs have demonstrated successful application in various NLP tasks, including named entity recognition and language modeling. According to previous studies [13], CNNs have demonstrated their efficacy in recognizing and categorizing particular entities in text for named entity recognition. According to another study [14], CNNs have exhibited potential in language modeling assignments by proficiently capturing contextual information and generating consistent language models.

The significance of CNNs is increasing among NLP researchers and practitioners. This is attributed to their capability of analyzing vast amounts of textual data. As the field of NLP progresses, it is probable that there will be further advancements in the application of CNNs for NLP tasks.

Recurrent Neural Networks (RNN)

A Recurrent Neural Network (RNN) is a specialized type of neural network that incorporates the ability to retain information from previous states of neurons. This unique characteristic allows RNNs to capture and utilize context information, making them particularly suitable for processing sequential data [15]. RNNs have gained significant prominence in the field of NLP and are extensively used in diverse tasks such as machine translation, speech recognition, text generation, and image captioning [16, 17, 18].

Unlike traditional Feed-forward Neural Networks (FFNN), RNNs have network loops that allow information to persist (Figure 2.4). This loop allows the network to store information about previous inputs, which can be used to inform subsequent outputs [19]. This makes RNNs particularly useful for sequential data, such as time series or language data, where the context of previous data points is crucial.

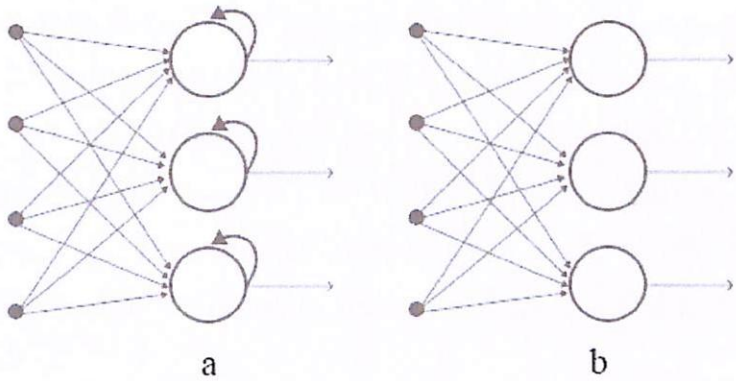

Figure 2.4: Comparison of RNN (a) and FFNN (b) [19].

This concept was introduced in 1982 by John Hopfield [20], and it was one of the earliest and most influential RNNs. It was named after him and has been widely used in various applications such as optimization, associative memory, and pattern recognition. And the first modern RNN was introduced in 1990, by Jeffrey Elman, also known as a simple recurrent neural network (SRNN) [21]. However, the first successful implementation of RNNs for NLP tasks is credited to the work of Hochreiter and Schmidhuber in 1997, where they introduced the Long Short-Term Memory (LSTM) [22] architecture, which addressed the vanishing gradient problem that faced earlier RNN models.

Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) is a type of RNN [22] that solves the vanishing gradient issue commonly encountered by conventional RNNs. This issue arises when errors are propagated backward through lengthy sequences, resulting in suboptimal performance in earlier layers. LSTM solves this issue by introducing a memory cell that can selectively retain or discard information at each time

step, with the assistance of three gates: the input, forget, and output gates [22].

The input gate determines how much new data is permitted to enter the memory cell. The forget gate regulates the retention or deletion of existing information in the memory cell. The output gate controls the output from the memory cell to the remainder of the network. The LSTM architecture is given in Figure 2.5.

LSTM excels at processing sequential data as a result of its ability to selectively retain or discard information over long sequences. In addition, it has been extensively utilized in numerous NLP applications, such as language modeling, machine translation, sentiment analysis, and QA tasks. However, despite its success, LSTM is not without limitations. For practical applications, the high computational cost of training large-scale LSTM models can be a challenge. Nevertheless, with ongoing research and advances in hardware technology, LSTM remains a powerful tool for processing sequential data and has created new opportunities in NLP and other fields.

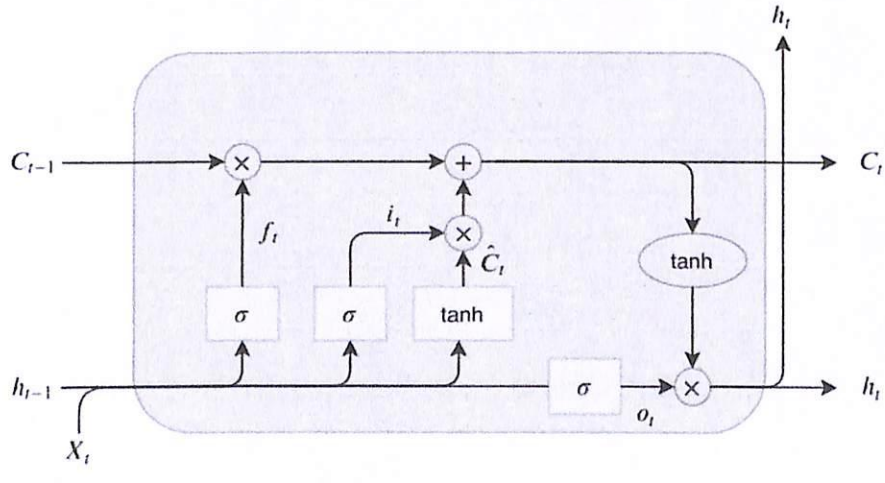

Figure 2.5: LSTM architecture [23].

Transformers

Transformer is a sort of neural network architecture, which was introduced by Vaswani et al [24] in 2017. Due to its capacity to deal with long-range dependencies and process input sequences in parallel, it has become a popular approach for NLP tasks.

Transformers, unlike traditional RNNs and LSTMs, do not employ recurrent connections. Instead, they use self-attention mechanisms to calculate contextual

representations for each element in the input sequence based on the relationship between all elements. This allows Transformers to capture complex relationships between dispersed elements in a sequence, which is crucial for NLP tasks. They have been utilized in numerous applications, such as language modeling, machine translation, speech recognition, and QA. The flexibility and power of Transformers make them a valuable tool for a wide range of NLP applications. The Figure 2.6 presents the Transformer architecture.

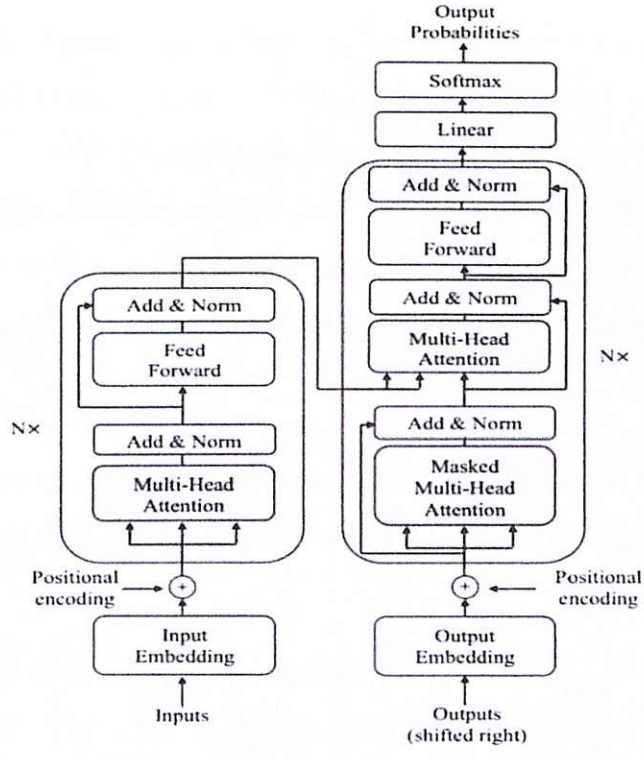

Figure 2.6: The transformer architecture from [24].

Summary

In summary, DL has had a transformative effect on the field of NLP by providing numerous advantages over traditional machine learning techniques. Firstly, DL models, particularly neural networks, can automatically extract complex features and patterns from raw input data, thereby eliminating the need for manual feature engineering. Secondly, DL models demonstrate a remarkable capacity to process vast quantities of data, which enables the development of more accurate and trustworthy models. Thirdly, DL models excel at processing sequential data, making them ideally suited for NLP tasks involving textual data. Moreover, DL models can be easily adapted to new tasks with minimal modifications

to the underlying architecture. Finally, DL models have demonstrated impressive performance on a variety of NLP tasks, such as sentiment analysis, machine translation, and QA. In general, the incorporation of DL into NLP has paved the way for significant advancements in this field, enabling the development of more precise and effective language models and applications.

2.2.2 Deep learning-based QA systems

For many years, QA systems have been a focus of active research, and the advent of deep learning neural networks has resulted in significant advancements in these systems. DL models have transformed the field by allowing QA systems to handle large amounts of data efficiently and provide precise answers in natural language [25]. This section provides a review of the literature on the use of DL techniques for developing QA systems, providing insights into the progress made in this field.

LSTM-based models

As previously stated, LSTM is a type of RNN that has shown promising results in NLP tasks such as QA systems. Here are some LSTM-based models that have been used for QA:

- **Dynamic Coattention Networks (DCN)** [26] is a state-of-the-art model for machine comprehension tasks such as QA. It uses bidirectional LSTMs for modeling the context and question and a coattention mechanism to fuse information from both sources.
- **Match-LSTM** [27] is a model for text-matching tasks that can be used for QA. It uses two LSTMs to model the context and question, and a matching mechanism to align and compare the two sequences.
- **LSTM-QA** [28] is a model for QA that uses LSTMs to encode the context and question and a multi-step attention mechanism to extract the answer.
- **Attentive Reader** [29] is a model for machine comprehension tasks, including QA. It uses a single LSTM to encode the context and another LSTM to encode the question. The coattention mechanism is used to obtain a

question-aware representation of the context, and an answer pointer network is used to select the answer span.

- **BiDAF (Bidirectional Attention Flow)** [30] is a state-of-the-art model for machine comprehension tasks, including QA. It uses a bidirectional LSTM for encoding the context and another bidirectional LSTM for encoding the question. A self-attention mechanism and a coattention mechanism are used to obtain a question-aware representation of the context, and an answer pointer network is used to extract the answer span.

In summary, the LSTM-based models produced promising results in QA tasks, particularly when tested on the SQuAD dataset. These models can detect long-term dependencies in textual data and effectively process input sequences of varying lengths.

Memory Networks

Although LSTM-based networks have made significant advances in NLP by addressing the issue of vanishing gradients and allowing the network to maintain long-term dependencies, the memory concept used in LSTM remains relatively short-term.This has led to the development of models that provide relatively long-term memory, allowing the network to better capture and remember important information.

One such example is the Memory Networks, which was introduced in 2014 by researchers at Facebook [31]. They have a structure that allows them to store and retrieve data from an external memory component, also known as working memory or short-term memory. This memory component can be updated with new information at each time step, and the network can use this information to make predictions or answer questions.

Memory Networks can use a distinct mode architecture where information is stored in memory slots that can be dynamically updated based on model inputs and outputs. This design enables better handling of long-term dependencies and more efficient utilization of memory. These networks have been effectively applied to a variety of NLP applications, including QA, machine translation, and dialogue

systems. They have shown particular promise in tasks that require complex reasoning and inference over multiple pieces of information.

Some notable examples of Memory Networks for QA:

- **End-To-End Memory Networks (MemN2N)** [32] are one of the earliest Memory Networks proposed for QA. It uses an external memory matrix to store information and an attention mechanism to retrieve relevant information based on the input query.
- **Dynamic Memory Networks (DMN)** [33, 34] extend the Memory Network architecture by introducing an RNN to dynamically update the external memory matrix based on the input query. This allows the model to better adapt to new inputs and improve its accuracy.
- **Key-Value Memory Networks (KV-MemNNs)** [35] introduce the concept of key-value memory, where each memory element is associated with a key and a value. This allows the model to better handle complex inputs and retrieve relevant information more efficiently.

Overall, Memory Networks have shown significant potential for enhancing the performance of QA systems by enabling efficient storage and retrieval of relevant information from long-term memory.

Transformer-based models

Transformer-based models have become increasingly popular in NLP due to their ability to achieve state-of-the-art performance on many NLP tasks. They have largely replaced traditional RNN models like LSTM. Unlike RNNs, which process input sequences sequentially, Transformers can process the entire sequence in parallel. This enables them to capture long-range dependencies more effectively and make better use of context information.

BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformer) are among the most renowned and extensively employed Transformer-based models in the field of NLP.

BERT is a pre-trained language model introduced by Google in 2018 [36] that

can be fine-tuned for different downstream NLP applications such as sentiment analysis, QA, and NER. BERT is pre-trained on a large corpus of unannotated text using a masked language modeling (MLM) aim and is built on the Transformer architecture. During pre-training, random words in the input sequence are masked, and the model is tasked with guessing the correct phrase based on the surrounding context. Furthermore, BERT uses a next sentence prediction (NSP) task, which involves training the model to predict whether or not two phrases in the original text are consecutive.

OpenAI introduced GPT in 2018 as a generative language model [37] trained to predict the subsequent word in a sequence based on the preceding words. Like BERT, GPT is built upon the Transformer architecture and undergoes pre-training on a vast text corpus using a language modeling (LM) objective. GPT has been trained on several large datasets, including CommonCrawl (a web text dataset) and various book corpora, and has showcased impressive performance across multiple NLP tasks, including language modeling, machine translation, and QA.

Both BERT and GPT have gained extensive usage and adaptation across diverse NLP tasks, resulting in the development of numerous variations and extensions. BERT, for example, has inspired models such as RoBERTa [38], ALBERT [39], and ELECTRA [40], which have further improved upon BERT's performance on various tasks. Similarly, GPT has served as a foundation for models like GPT-2 [41] and GPT-3 [42], which have significantly expanded the scope and performance of generative language models.

2.3 Agglutinative Languages and QA

In agglutinative languages, morphemes are combined to form words. These morphemes have a consistent meaning and are added to create new words in a systematic manner, frequently resulting in long and complex words. These languages belong to a variety of language families, including Uralic, Altaic, Dravidian, and the majority of Native American languages. Examples of agglutinative languages include Kazakh, Turkish, Finnish, Mongolian, and Malayalam, as well as a num-

ber of American indigenous languages. Due to their rich morphology and complex word formation rules, they pose unique challenges for NLP tasks such as QA. However, numerous attempts have been made to develop QA systems for agglutinative languages cite [25].

The QA system for the Turkish language, which is an example of an agglutinative language, is one such instance. In a 2021 study conducted by Turkish researchers [43], a QA system was developed using advanced AI algorithms and extensive datasets, with a focus on the BERT algorithm. This system incorporated language models and a mechanism for fine-tuning machine reading in QA tasks, with a focus on banking-related documents. Using both original and translated datasets, the research team conducted a series of experiments to address the complexities resulting from Turkish's complicated morphology.

Another example is the QA system developed specifically for the Malayalam language, which is also classified as an agglutinative language. In this regard, researchers proposed a deep learning-based QA system [44] tailored for Malayalam. The system incorporates various models, including LSTM, Gated Recurrent Unit (GRU), and Memory Network models. To evaluate its performance, the system underwent training and testing using the Facebook bAbI dataset [45], which encompasses 20 tasks involving questions with multiple supporting facts, inductive and deductive reasoning, coreference, and more. The results demonstrated that among the three implemented models, the Memory Network model exhibited the highest average accuracy (80%) in retrieving precise answers in the Malayalam language.

Furthermore, Mongolian researchers have introduced a novel approach to Mongolian QA in the form of the Interactive Mongolian Question Answer Matching Model (IMQAMM) [46]. This model employs an attention mechanism and is comprised of two essential elements: interactive information enhancement and max-mean pooling matching. Using sequence enhancement and multi-cast attention, the interactive information enhancement generates scalar features via multiple attention mechanisms. In order to improve the learning of semantic features, Mongolian morpheme representation is incorporated into the model. The IMQAMM model was evaluated using a corpus of question-answer pairs from

the legal domain in Mongolian. The experimental results demonstrate that the IMQAMM model performs significantly better than the baseline models.

2.3.1 Previous Works on Kazakh Language Processing and QA

Limited research has been conducted on QA systems specifically for the Kazakh language, which is characterized by its agglutinative nature, complex derivational structure, and rich morphology. However, researchers have made efforts to develop Kazakh language models for other NLP tasks, such as sentiment analysis and text classification. For instance, in one study [47], a Kazakh text classification system was proposed utilizing CNNs. Similarly, another study [48] focused on creating a character-based language model for Kazakh using Deep Neural Networks, specifically the LSTM model, to generate contextually correct words. The model was trained on Kazakh-language texts to ensure accuracy.

In a different study [49], a multi-task learning model named MTQU (Multi-Task Query Understanding) was introduced, incorporating DL techniques and leveraging unique features of the Kazakh language. MTQU aimed to establish connections between tasks such as question categorization and NER. Notably, the model incorporated a multi-feature input layer, which proved beneficial for training performance. Experimental results demonstrated the effectiveness of the MTQU model in enhancing question classification and NER. However, further research is needed to address the specific challenges posed by the Kazakh language in the context of QA systems.

While DL neural networks have demonstrated significant potential in the development of QA systems, several challenges remain to be tackled. One of these challenges pertains to the availability of annotated datasets for training and evaluating these systems. Furthermore, the interpretability of these models is a challenge because understanding the reasoning behind the model's replies can be difficult. Moreover, agglutinative languages provide distinct obstacles in NLP, pushing researchers to investigate language-specific models for tackling these issues. Despite these challenges and the lack of research on QA systems in Kazakh,

the potential of using DL techniques to QA in natural language has promise for future study [25]. Therefore, this work concentrates on the development of QA systems for the Kazakh language, using DL approaches as a means to overcome these challenges.

Chapter 3

Methodology

3.1 Data Description

The study utilized a set of six QA tasks from the bAbI benchmark dataset [45] to assess the efficacy of the implemented models, as demonstrated in Table 3.1. The dataset is available in two languages: English and Hindi. It has also been translated into various languages, such as Chinese, Malayalam, and Indonesian, which enhances its usefulness as a research tool for investigating language comprehension across diverse settings.

The dataset comprises a set of unique question-answering tasks. The purpose of each assessment is to evaluate a specific aspect of language comprehension and reasoning. The assignments are supplemented with a set of fabricated narratives, which are concise depictions of occurrences. The structure of each story is composed of a series of statements that depict the events that unfold throughout the narrative.

The dataset was translated into Kazakh and saved as CSV files. Each dataset consists of three columns:

- **Story (S)** is a column that represents a statement containing the relevant facts regarding the given situation or context.
- **Question (Q)** is a column that represents the question related to the story's

information.

- **Answer (A)** is a column indicating the correct and accurate response to the question posed.

The datasets were subjected to a comprehensive analysis to verify their reliability and validity. The analysis involved checking for any instances of duplicate samples and assessing the quality of the data to ensure that it was unbiased and precise. The data was cleaned and prepared by removing duplicates, filling in missing values, and correcting any errors and inconsistencies. The data distribution was assessed to ensure its appropriateness and balance for the research.

Dataset 1: Single Supporting Fact	**Dataset 2: Two Supporting Facts**
Мэри жуынатын бөлмеге барды. Джон дәлізге шықты. Мэри кеңсеге барды. Мэри Қайда? Ж: кеңсе	Джон ойын алаңында. Джон футбол добын көтерді. Боб ас үйге барды. Футбол добы қайда? Ж: ойын алаңы
Dataset 3: Yes/No Questions	**Dataset 4: Counting**
Джон ойын алаңына бет алды. Даниэль жуынатын бөлмеге барды. Джон дәлізге оралды. Джон ойын алаңында ма? Ж: жоқ	Даниэль футбол добын көтерді. Даниэль футбол добын тастады. Даниэль сүт әкелді. Даниэль алманы алды. Даниелдің қолында қанша зат бар? Ж: екі
Dataset 5: Lists/Sets	**Dataset 6: Agent's Motivations**
Даниэль футбол добын көтереді. Даниэль қағазды тастайды. Даниэль сүт алады. Джон алманы алды. Даниелдің қолында не бар? Ж:сүт, футбол	Джон ашыққан. Джон ас үйге барады. Джон алманы ұстап алды. Даниэль ашыққан. Даниэль қайда барады? Ж: ас үй Неліктен Джон ас үйге барды? Ж: ашыққан

Table 3.1: Examples of each Dataset.

3.2 Preprocessing

The preprocessing of data is a crucial step in NLP and ML operations. The process involves the transformation of unprocessed data into a structured and

organized format that can be effectively utilized by various models. This section outlines the preparation methods employed for the dataset in our research.

The significance of data preprocessing is well-established due to various reasons:

- **Enhancing data quality.** Unprocessed data are frequently insufficient, noisy, or inconsistent. Data preprocessing assists in addressing these issues and enhancing data quality.
- **Error reduction.** Data preprocessing identifies and corrects data errors such as missing values, outliers, and incorrect data types.
- **Improving model accuracy and performance.** Preprocessing data can improve the accuracy and performance of models. Scaling or normalizing data, for instance, can help improve model convergence and decrease overfitting.
- **Saving time and resources.** By preparing the data beforehand, ML and DL algorithms can process the data more efficiently, saving time and resources.

In this step, we used several techniques, including converting to lowercase, removing punctuation, stemming, and tokenization. Each technique is employed to address specific data issues and make the data more manageable.

Data Cleaning

This methodology incorporates the identification and handling of missing values, duplication, and data errors. The utilization of a clean dataset serves as a robust foundation for subsequent data analysis, modeling, and downstream NLP operations, leading to outcomes that are more precise and dependable. The data-cleaning steps undertaken for the dataset were as follows:

- The dataset was thoroughly examined for duplicate samples and subsequently removed to prevent overfitting and enhance the models' performances.
- Missing values were filled in the dataset. The dataset was checked for any instances of missing values, and the gaps were subsequently filled with ap-

propriate values.

- The text data in the datasets underwent a conversion to lowercase to maintain uniformity in the word representation.
- Punctuation marks were removed from the text data to reduce noise and simplify the data.

Through the rigorous implementation of data-cleaning procedures, the dataset was refined and deemed reliable for subsequent analysis and modeling.

Stemming

The process of stemming is a crucial step in converting words to their fundamental or foundational form, which results in a more cohesive and consistent representation. The Kazakh text was subjected to the stemming method using the KazNLP toolkit [50]. This process involved reducing words to their basic form, which helped to reduce the total number of unique words present in the dataset.

The decrease in the number of distinct terms had a clear influence on the lexicon employed in our analyses. The study found that the models utilized were capable of analyzing the given text input and extracting significant patterns using a reduced vocabulary. The reduction of input dimensionality in models has resulted in improved computational efficiency and reduced learning process complexity.

Here is an example of using stemming:

Before stemming: 'аяжан алманы ас үйге тастады'.

After stemming: 'аяжан алма ас үй таста'

By incorporating stemming into our preprocessing pipeline, we were able to attain a more condensed representation of the Kazakh text. As a result, the performance of our models in downstream tasks was enhanced. The employed technique proved to be a valuable approach in achieving uniformity in word structures and

enabling enhanced analysis of the dataset.

Tokenization

To prepare the textual data for feeding into the models, we performed tokenization which is a process of segmenting text into meaningful units known as tokens. Tokenization enables us to break down the textual data into individual words or subwords, which can be analyzed and processed by the models. In other words, the tokenizer method helped us to convert the raw text data into numerical data that can be fed into the deep learning models.

Tokenization process: ['аяжан', 'алма', ' ас', ' үй', 'таста'].

3.3 Feature Extraction and Representation

Converting text data into a format that can be processed by computer systems poses a fundamental challenge when applying analytical tools. As computers primarily operate on numerical data, it becomes necessary to transform the text into a numerical representation.

In the context of training and evaluating our deep learning-based QA systems, we conducted feature extraction and representation of the textual data. Feature extraction involves transforming the raw textual data into numerical features that can be effectively processed by the models. The subsequent representation step focuses on encoding these extracted features into a suitable format that allows the models to learn and make predictions. One widely used way to address this problem is to express words as vectors, which have a meaningful interpretation and can be used efficiently across multiple tasks. [51].

LSTM baseline and MemN2N

For our LSTM baseline and MemN2N models, we implemented a *Vectorize* function. This function facilitates the transformation of stories, questions, and answers into numerical sequences of equal lengths. Initially, we determined the maximum length for both stories and questions. Then, we systematically processed each story, question, and answer in the dataset by converting the raw

words into their corresponding numerical values. To ensure uniformity in the input data, we employed sequence padding techniques, which involved adding appropriate tokens or zero values to align the sequences with the maximum length determined earlier [25].

Finally, the preprocessed data was organized into a tuple, encompassing the vectorized representations of stories and questions as padded sequences, and one-hot encoding representations of answers *(S, Q, A)*. This tuple encapsulated the numerical representations of the stories, questions, and answers. The *pad_sequence* represented the padded sequences of stories and questions, ensuring uniform length for input data across examples. On the other hand, the *one-hot encoding* scheme was employed to represent the categorical answers, converting them into binary vectors with a value of 1 for the correct answer and 0 for the incorrect ones. See Figure 3.1.

These transformed representations served as the input to our LSTM baseline and End-to-End Memory Network models, enabling them to effectively learn and make accurate predictions in the Kazakh language-based QA tasks.

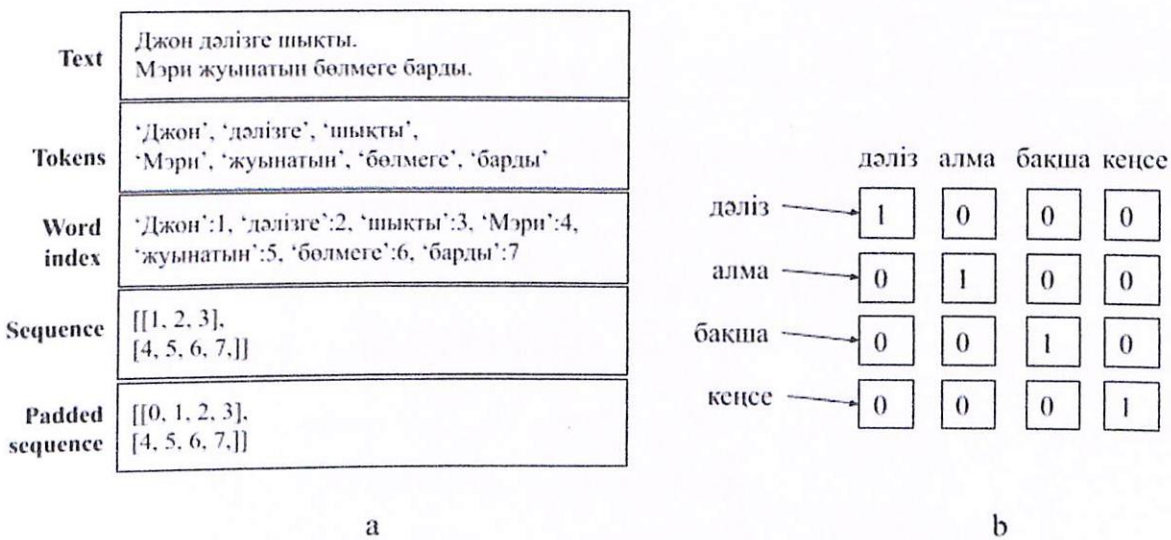

Figure 3.1: Examples of Padded Sequence (a) and One-hot Encoding (b).

BERT

By using *tokenizer.batch_encode_plus* [52] and *Dataset.from_tensor_slices* [53] together, the input data is tokenized, encoded, and formatted into Tensor-

Flow datasets suitable for training the BERT model. Here's an overview of the input encoding steps:

- *Tokenization:*
  - The input text is split into individual tokens using the tokenizer.
  - Additional tokens are included in the sequence to provide special functionalities. For instance, the **[CLS]** token marks the beginning of the sequence, while the **[SEP]** token is used to distinguish between sentences.
- *Token to ID Conversion:*
  - Each token is mapped to its corresponding token ID from the BERT vocabulary.
  - The BERT model has a pre-defined vocabulary where each token has a unique ID. The tokenizer maps the tokens to their respective IDs using the vocabulary.
- *Padding and Truncation:*
  - BERT models typically have a fixed maximum sequence length.
  - If the tokenized input exceeds the maximum length, it is truncated by removing tokens from the end of the sequence.
  - If the tokenized input is shorter than the maximum length, it is padded with a special **[PAD]**token to match the desired length.
  - Padding ensures that all input sequences have the same length, which is necessary for efficient batch processing.
- *Attention Masks:*
  - To guide the BERT model in focusing on relevant tokens during its processing, an attention mask is generated. This mask helps determine which tokens should receive attention and which ones can be disregarded. By using the attention mask, the BERT model can effectively allocate its attention resources to the appropriate tokens in the

input sequence.

- Attention masks are binary tensors of the same length as the input sequence, with 0s indicating padded tokens and 1s indicating non-padded tokens.
- The attention mask helps the model distinguish between real tokens and padding tokens, preventing the model from attending to irrelevant positions.

The encoded input data, which returned by the tokenizer.batch_encode_plus includes the following components:

- **input_ids:** The token IDs representing the input sequence.
- **attention_mask:** The attention mask indicating which tokens to attend to.

3.4 Model Selection

3.4.1 LSTM baseline

The provided Baseline model is implemented using the Keras library. Here is a breakdown of its architecture (Figure 3.2):

- **Input Layer:** Two input tensors are defined for the story and question sequences.
- **Embedding Layer:** Two embedding layers are utilized for both the story and question sequences. These layers are responsible for mapping each word within the input sequences to a dense vector representation. They convert discrete words into continuous vector representations that capture the semantic meaning and word relationships.
- **LSTM Layer:** Two LSTM layers are used, one for the story sequence and one for the question sequence.
- **Concatenation:** The outputs of the LSTM layers are combined through

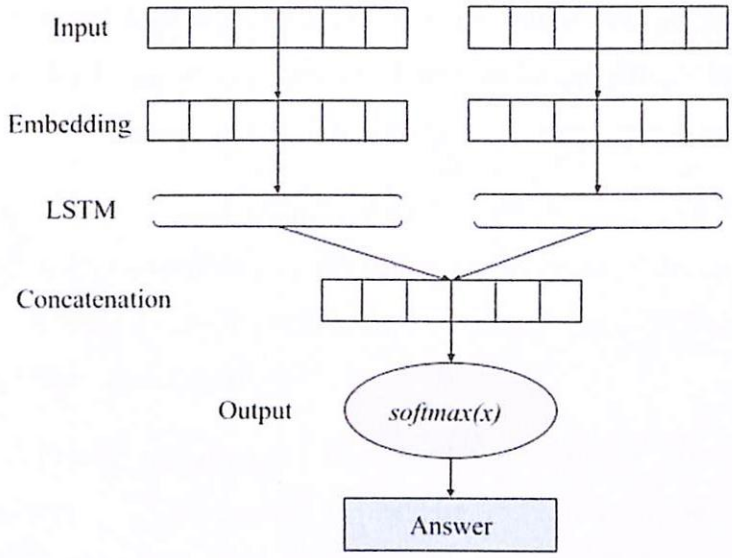

Figure 3.2: LSTM baseline.

the concatenation layer. This layer enables the model to merge and integrate the information from both the story and question sequences. By concatenating the outputs, the model can effectively capture and utilize the relevant information from both sources in order to make accurate predictions or decisions.

- **Output Layer:** A dense layer with a softmax activation function makes up the output layer. The output layer is responsible for generating the predicted probabilities for each class or potential answer.

3.4.2 End-to-End Memory Network (MemN2N)

End-to-End Memory Network (MemN2N) model was introduced by Sukhbaatar et al. in 2015 [32]. This model was created primarily for QA activities that require reasoning over long-term memory. To enable the storing and retrieval of information from a memory matrix, the MemN2N model incorporates many neural network layers, including an input module, a memory module, and an output module. The model learns to perform QA tasks by using specified input and output examples through an end-to-end training procedure. The MemN2N model's effectiveness has been proved on several benchmark datasets, outperforming previous approaches in solving problems that need complicated reasoning capabilities.

The key idea behind the MemN2N is to use an external memory component that allows the model to read and write information during the QA process. The model's architecture consists of the following primary components:

- **Input Encoding.** The input, which can be a story or a document, is transformed into a collection of memory vectors during the input encoding phase. These memory vectors represent individual pieces of data or sentences within the input.
- **Memory Representation.** Memory vectors are stored in a module of external memory. This memory module serves as the model's form of long-term memory, allowing it to store and retrieve information over time. The external memory enables the model to store and retrieve relevant information for responding to questions.
- **Attention Mechanism.** An attention mechanism is used to identify the most pertinent memory vectors for answering a given question. This mechanism assigns memory vector weights proportional to their relevance to the question. By doing so, the model can prioritize and focus on the most pertinent pieces of information, enabling the generation of more accurate and targeted answers.
- **Output Generation.** The attended memory vectors, along with the question, are combined to generate the final output or answer. This can involve various operations such as concatenation, summation, or more complex transformations. The output generation step aims to produce a response that is coherent and relevant to the question.

The model receives a discrete set of inputs, denoted as $X_1, ..., X_n$, which are subsequently stored in the memory. These inputs may take the form of real sentences or stories. Additionally, the model is provided with a question Q, and an answer A. The symbols of X_i, Q, A and A are drawn from a lexicon containing a vast number of V words. The variable V encompasses the complete lexicon present in all datasets. Upon reaching a predetermined buffer size, the model proceeds to store all instances of X in memory. Subsequently, the model proceeds to establish a continuous representation for X and Q. Subsequent to that, the

continuous representation is utilized to process output A through a series of hops. The Figure 3.3 illustrates the End-to-end Memory Network architecture.

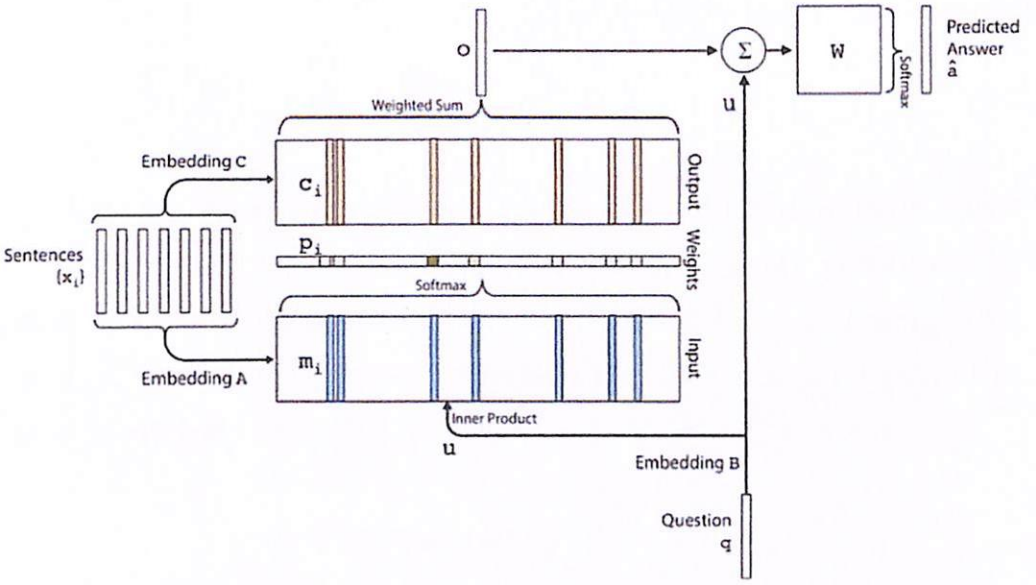

Figure 3.3: End-to-end Memory Network (MemN2N) from [32].

Representation of the input memory: Let's consider an input set $x_1, x_2, ..., x_i$ that needs to be stored in memory. Each element x_i in the set is transformed into memory vectors m_i of dimension d. This transformation is achieved by embedding each x_i into a continuous space. In the simplest scenario, an embedding matrix A of size $d \times V$ is used for this purpose. The query q is also embedded, using another embedding matrix B of the same dimensions as A, resulting in an internal state u. In the embedding space, we compute the similarity between u and each memory vector m_i by taking their inner product. This similarity measure is then passed through a softmax function, yielding probabilities p_i (3.4.1):

$$p_i = softmax(u^T m_i) \tag{3.4.1}$$

Representation of the output memory: Each input element x_i corresponds to an output vector c_i. In the simplest scenario, the output vectors c_i are obtained by applying another embedding matrix C to the inputs x_i. The response vector o, which is generated from the memory, is computed as a weighted sum of the transformed output vectors c_i. The weights are determined by the probability

vector obtained from the input memory. This means that the contribution of each output vector c_i to the response vector o is determined by the corresponding probability in the input memory (3.4.2):

$$o = \sum(p_i c_i) \tag{3.4.2}$$

Generating the final prediction: In the case of a single layer, the response vector o from the memory and the input embedding u are combined by adding them together. This combined vector is then multiplied by a weight matrix W of size $V \times d$. The resulting vector is passed through a softmax function, which produces the predicted label (3.4.3):

$$output = softmax(W(o + u)) \tag{3.4.3}$$

3.4.3 BERT

The BERT (Bidirectional Encoder Representations from Transformers) architecture builds upon the Transformer model initially proposed by Vaswani et al. in 2017 [24]. However, BERT incorporates several modifications to make it particularly well-suited for pre-training on extensive amounts of unlabeled data and subsequent fine-tuning on specific downstream tasks. This influential model was introduced by researchers from Google AI Language in 2018 [36].

Encoder

The encoder is the core component of BERT and is responsible for processing the input text. It comprises several layers of transformers arranged in a stacked manner. Each transformer layer consists of two sub-layers: a multi-head self-attention mechanism and a feed-forward neural network (Figure 3.4).

- **Self-Attention:** The self-attention mechanism in BERT enables individual words in the input sequence to consider and weigh the importance of other words, thereby capturing the contextual relationships between them. It calculates attention weights for each word based on its relevance to the other words in the sequence. This bidirectional approach allows BERT to

effectively model the dependencies between words.

- **Feed-Forward Neural Network:** After the self-attention step, a position-wise feed-forward neural network is applied to each word representation independently. This network applies non-linear transformations to capture complex relationships between words.

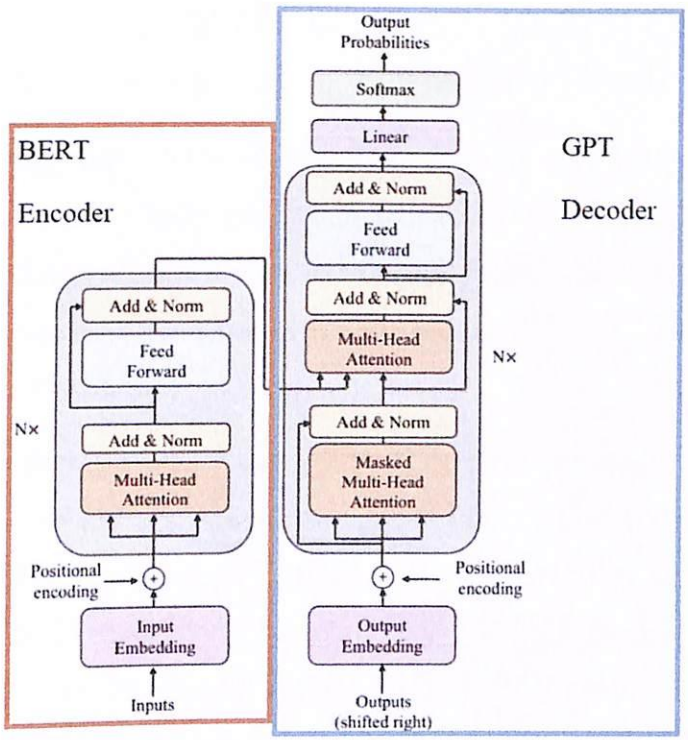

Figure 3.4: Transformer architecture from [24].

The input to BERT is a sequence of tokens, which are typically words or sub-word units. These tokens are first embedded into vector representations called word embeddings or token embeddings.

One of the notable features of BERT is the use of attention masks. Attention masks indicate which tokens are actual words and which are padding or special tokens. The masks ensure that the model does not attend to the padded or masked tokens, allowing it to focus on the relevant information. The transformer layers in BERT enable the model to capture deep contextual information from both the left and right contexts of each word in the input sequence.

BERT improves transformer architecture by including two different stages. The model is pre-trained on a massive amount of unlabeled text data in the first step,

allowing it to learn general language patterns and structures. The model is further trained on specific natural language understanding (NLU) tasks using labeled data in the second stage, referred to as fine-tuning. This approach allows the model to adapt and specialize its learned representations to the tasks at hand.

Pre-training

In the pre-training phase, BERT leverages a substantial corpus of unannotated text and concentrates on accomplishing two primary objectives:

- **Masked Language Modeling (MLM).**In the MLM process, a portion of the words in the input sequence are randomly masked. BERT has been taught to predict these masked words by taking the surrounding words' context into account. This procedure enhances BERT's contextual knowledge of word representations.
- **Next Sentence Prediction (NSP).**In the pre-training phase, BERT is subjected to sentence pairs in the NSP task. The model is trained to determine whether the second sentence in the original text is a continuation of the first. By focusing on this objective, BERT is able to comprehend the sentence-level semantics and capture the relationships between sentences.

In contrast to previous models that relied on sequential left-to-right or right-to-left approaches, BERT utilizes an innovative technique. The methodology assesses the linguistic units that precede and follow each word in a given phrase, enabling the construction of contextual representations that encompass a wider scope. This technique yields embeddings that are more advanced and exhibit contextual awareness.

The pre-trained representations of BERT can be further refined or fine-tuned for specific tasks, such as QA, sentiment analysis, or named entity recognition. Through pre-training on a comprehensive corpus of text with the aforementioned objectives, BERT acquires the capability to capture the contextual connections among words and sentences. This enables the model to be effectively adapted to various downstream tasks with improved performance.

Fine-tuning

During an additional training phase, a smaller labeled dataset specific to a given task is utilized to develop a fine-tuned BERT model. Following the preliminary pre-training phase of the model, it acquires comprehensive linguistic representations from an extensive collection of unannotated textual data. The process of fine-tuning involves the adaptation of pre-trained BERT data to meet the specific demands of a given task and is considered to be a crucial phase in this regard.

During the fine-tuning phase, the BERT model is trained on a reduced dataset that is specifically related to the given task. The dataset comprises labeled instances that pertain to various tasks, including labeled sentences for text classification, named entity recognition, and QA. Phase strategy involves adjusting the model's weights using labeled data and a loss function specific to the task at hand. This approach enables the model to learn task-specific patterns and enhance its performance on the target task. The Figure 3.5 illustrates the End-to-end Memory Network architecture.

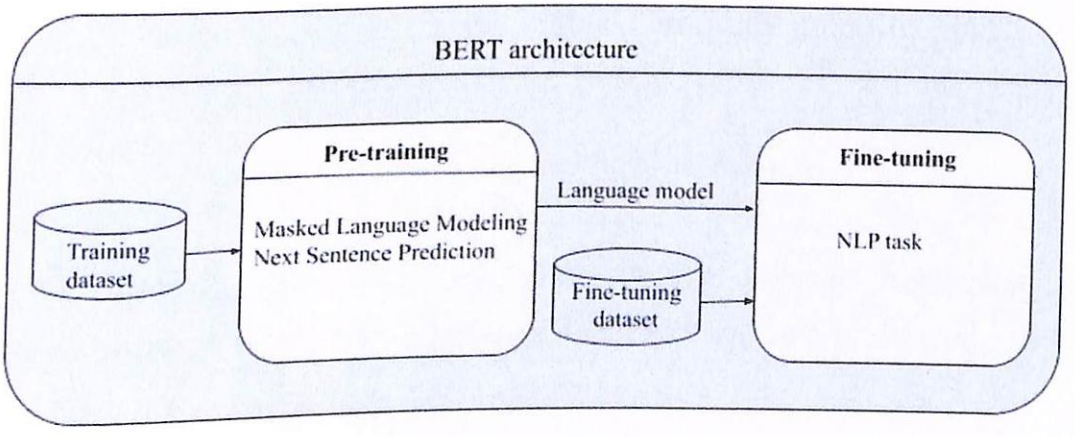

Figure 3.5: BERT architecture.

3.5 Evaluation Metrics and Procedures

We evaluated the performance of the implemented models using standardized procedures and evaluation metrics. The evaluation metrics utilized in this study yielded quantitative measurements to assess the effectiveness and accuracy of the models. The study's evaluation procedures were developed to guarantee impartial and uniform evaluations of the models on different datasets.

Evaluation Metrics

The evaluation of the performance of the implemented models was conducted using the following evaluation metrics:

- **Accuracy:** The metric of accuracy is utilized to assess the performance of models by measuring the proportion of correctly predicted responses. The accuracy metric is determined through the computation of the ratio between the number of correctly predicted responses and the total number of samples present in the test set. The following is equation of Accuracy 3.5.1:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{3.5.1}$$

  Here, **TP** = true positive, **TN** = true negative, **FP**= false positive, **FN** = false negative.

- **F1 Score:** The F1 score is a commonly used evaluation metric in various fields of research. It is a measure that combines precision and recall, providing a single value through the harmonic mean. The following is equation of F1 Score 3.5.2:

$$F1 = 2 * \frac{precision * recall}{precision + recall}. \tag{3.5.2}$$

  The F1 score is particularly useful in situations where both precision and recall are important, and a balance between the two is desired. In here:

  - *Precision:* Precision measures the proportion of correctly predicted positive instances (i.e., correct answers) among all instances predicted as positive. The following is equation of Precision 3.5.3:

$$Precision = \frac{TP}{TP + FP} \tag{3.5.3}$$

  - *Recall:* Recall measures the proportion of correctly predicted positive instances (i.e., correct answers) among all actual positive instances. The following is equation of Recall 3.5.4:

$$Recall = \frac{TP}{TP + FN} \tag{3.5.4}$$

Evaluation Procedures

To ensure fair evaluation and reliable results, we conducted the following procedures during the evaluation of our QA system:

- **Test Data Split:** To ensure a balanced distribution of samples across classes, we employed a stratified approach to divide the preprocessed dataset into training and testing sets. The training set was utilized to train the models, while the testing set remained separate for the purpose of evaluating their performance.
- **Hyperparameter Tuning:** In our experiments, we explored different hyperparameters such as the number of LSTM units, learning rate, size of the batch, and number of training epochs. The objective was to refine the model's performance by fine-tuning these parameters. By evaluating the model with various hyperparameter configurations, we identified the specific set that yielded the most favorable results based on the evaluation metrics. These optimal hyperparameters were subsequently chosen for the final models, ensuring their suitability for the given task.
- **Model Comparison and Analysis:** Using the aforementioned assessment metrics, we compared the performance of the models, notably BERT, LSTM baseline, and MemN2N. We studied the findings to determine each model's strengths and limitations and to get insight into their performance on Kazakh language-based QA tasks.

The evaluation metrics and procedures outlined above allowed us to comprehensively assess the effectiveness and accuracy of our QA system. They provided objective measures of performance, facilitated fair comparison between models, and aided in the interpretation and analysis of the results.

3.6 System architecture

The overall system architecture consists of four steps, i.e. preprocessing, vectorization, training models, and prediction. The system architecture is illustrated in Figure 3.6.

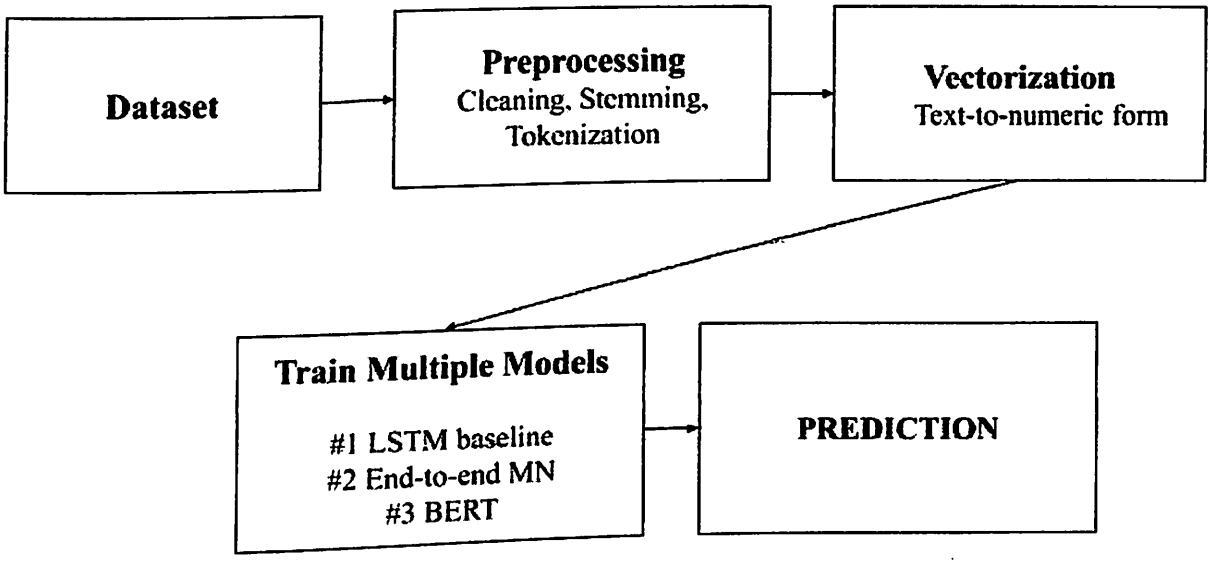

Figure 3.6: System architecture.

Dataset: The system starts with a dataset that consists of contextual information, questions, and corresponding answers in the Kazakh language. This dataset serves as the foundation for training and evaluating the QA models.

Preprocessing: The dataset is preprocessed before training the models, including cleaning, stemming, and tokenization. Cleaning is the process of removing any useless or noisy data, whereas stemming is the process of reducing words to their basic or root forms. Tokenization divides the text into individual tokens, such as words or subwords, which are required for subsequent processing.

Vectorization: Following preprocessing, the dataset is converted into numerical representations that models can understand. In this step, text data were transformed into numerical feature vectors. This process typically involves vectorization techniques. Vectorization captures the text's underlying semantic meaning and structure, allowing models to learn and predict based on these representations.

Train Models: The preprocessed and vectorized dataset is used to train multiple models for the QA task. In the system architecture, three models are employed: LSTM baseline, MemN2N, and BERT. The LSTM baseline model utilizes LSTM layers to process the sequential information in the dataset. The MemN2N model leverages the memory network architecture to store and retrieve informa-

tion from the context. The BERT model, based on the Transformer architecture, captures bidirectional contextual relationships between words. Each model is trained on the dataset to learn the patterns and relationships between the context, questions, and answers in the Kazakh language.

Prediction: Once the models are trained, they are used for prediction. Given a new context and question, the trained models generate predictions for the corresponding answer. This phase involves feeding the input to the models and obtaining the output based on their learned representations and patterns. The model with the highest confidence or probability for the predicted answer is selected as the final prediction.

Each step contributes to the overall performance of the QA system, with the goal of accurately predicting answers for given contexts and questions in the Kazakh language.

3.7 Training and Testing process

We provided a detailed description of the training and testing processes for the implemented models in this section. The process of training involves feeding the models preprocessed and vectorized input data, iterating over multiple epochs, and updating the model's weights based on the optimizer and loss function that has been chosen.

Loss Function

Loss functions are important in training models, particularly neural networks, because they quantify the difference between predicted and true values and serve as a measure of the model's success. During training, the goal is to minimize the loss across multiple training examples.

In our work, we utilized two common loss functions: *categorical cross-entropy* and *sparse categorical cross-entropy*. These loss functions facilitate model learning and improvement by measuring the difference between predicted class distributions and the true class distributions.

Both categorical cross-entropy and sparse categorical cross-entropy use the same loss function, which can be expressed as follows (3.7.1):

$$L_{CE} = -\sum_{i=1}^{n} y\_true_i \times log(y\_pred_i), \text{ for n classes} \tag{3.7.1}$$

However, the distinction lies in how the truth labels are represented. Categorical cross-entropy is employed when the true class labels are in the form of one-hot encoding, where only one element is active (1) and the rest are inactive (0). On the other hand, sparse categorical cross-entropy is used when the true class labels are provided as integers.

Despite this discrepancy, both loss functions optimize the same underlying loss function during model training, aiming to enhance the model's classification performance.

Optimization

Optimization plays a crucial role in ML and DL, aiming to find the optimal set of parameters or weights for a model to minimize the error or loss function and enhance its performance on unseen data.

There exist several optimization algorithms, each with its own approach to updating the parameters. Popular choices include Gradient Descent, Stochastic Gradient Descent (SGD), Adam, RMSprop, and Adagrad, among others.

In our work, we utilized the Adam optimizer to train our models. Appendix A.1 provides an illustration of the Adam optimizer algorithm. The Adam optimizer combines the benefits of AdaGrad and RMSprop by dynamically adapting the learning rates based on the first and second moments of the gradients. It is widely employed in various DL applications due to its fast convergence and robustness.

Learning Rate

The learning rate determines the step size at each iteration during the optimization process. Setting the learning rate too high can lead to the network overshooting the optimal solution and failing to converge. This results in insta-

bility and the inability to find an effective solution. On the other hand, using a learning rate that is too small can cause the training process to be excessively slow and time-consuming, as it requires more iterations to reach convergence.

In Keras [54], the default learning rate value for most optimizers is 0.001. However, it is often necessary to experiment with different learning rate values to find the optimal one for a specific problem. As a rule of thumb, it is recommended to explore a range of initial learning rates, typically within the range of [0.001, 0.01, 0.1, 1]. By trying different learning rates within this range, you can observe their effects on the model's convergence and performance.

Regularization

To address the issue of overfitting, which occurs when a model becomes too specialized in the training data and struggles to generalize to unseen data, the regularization technique was applied. Regularization helps control the complexity of the model and reduces its dependence on specific training examples.

The regularization method, which is frequently used in neural networks is Dropout. It functions by randomly deactivating a portion of input units during the training phase, usually ranging between 0.2 and 0.5. This technique prevents the network from excessively relying on any specific feature or combination of features by randomly eliminating units.

This regularization method provides several advantages, including mitigating overfitting, improving generalization, and potentially accelerating training by reducing reliance on specific training examples.

Tables 3.2, 3.3, and 3.4 detail the parameters of the models that we utilized in our study. The implementations of the models in Appendices A.2, A.3, and A.4 respectively.

Model	LSTM baseline
Optimizer	Adam
Learning Rate	2e-5
Loss Function	Categorical Cross-entropy
Evaluation Metrics	Accuracy, F1-score
Embedding Dimension	64
LSTM Units	64
Dropout Rate	0.3
Batch Size	32
Epochs	120

Table 3.2: LSTM baseline's parameters.

Model	BERT
Optimizer	Adam
Learning Rate	2e-5
Loss Function	Sparse Categorical cross-entropy
Evaluation Metrics	Accuracy, F1-score
Dropout Rate	0.3
Batch Size	32
Epochs	5

Table 3.3: BERT parameters.

Model	End-to-end MN
Optimizer	Adam
Learning Rate	0.001
Loss Function	Categorical Cross-entropy
Evaluation Metrics	Accuracy, F1-score
Embedding Dimension (input_encoder_m)	64
Dropout Rate (input_encoder_m)	0.3
Embedding Dimension (input_encoder_c)	max_question_len
Dropout Rate (input_encoder_c)	0.3
Embedding Dimension (question_encoder)	0.3
Dropout Rate (input_encoder_c)	0.3
LSTM Units	0.3
Dropout Rate (answer)	64
Batch Size	32
Epochs	120

Table 3.4: End-to-end Memory Network model parameters.

Chapter 4

Results and Analysis

4.1 Dataset Analysis

As mentioned before, we utilized the bAbi dataset [45]. We have translated the six bAbi datasets into Kazakh, creating a comprehensive and diverse collection of data for analysis. Each dataset consists of 10,000 training set and 1,000 testing set, ensuring a comprehensive evaluation framework (Table 4.1).

These translated datasets cover a wide range of question types, providing a comprehensive assessment of our models' performance. The question types included in our datasets encompass single fact, two facts, yes/no questions, counting, lists/sets, and agents' motivations. This diverse coverage allows us to thoroughly evaluate our models' capabilities.

Dataset	**Training Samples**	**Testing Samples**
1: Single Supporting Fact	10000	1000
2: Two Supporting Facts	10000	1000
3: Yes/No Questions	10000	1000
4: Counting	10000	1000
5: Lists/Sets	10000	1000
6: Agent's Motivations	10000	1000

Table 4.1: Training and testing samples of each dataset.

To gain further insights into the datasets, we conducted an analysis to determine the maximum and average lengths of stories, questions, and answers for each

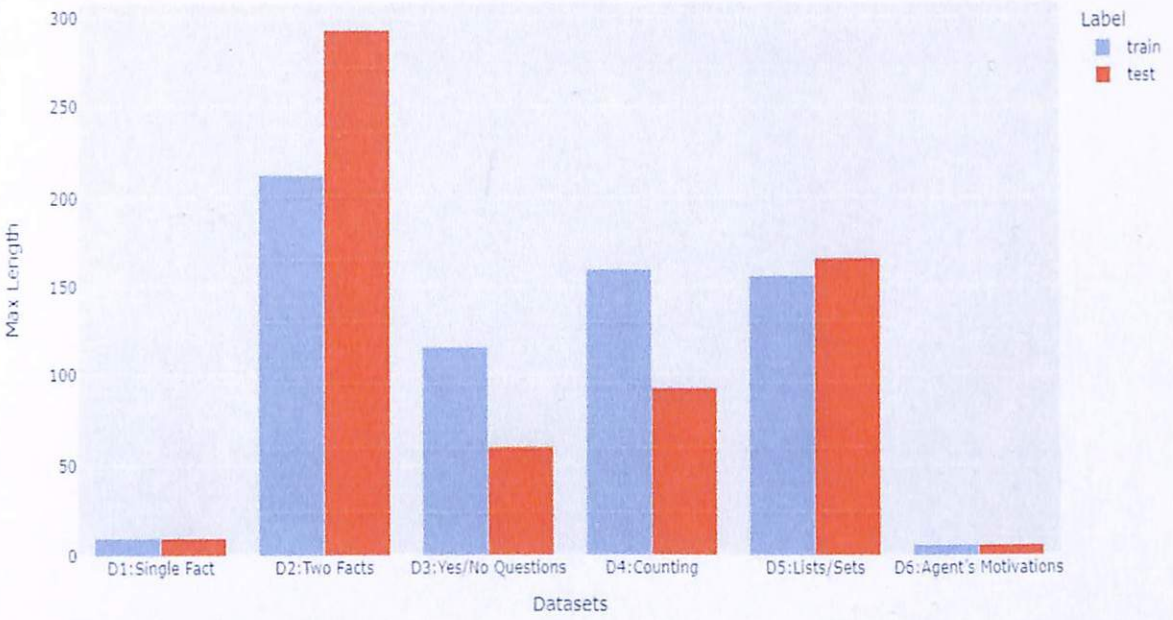

Figure 4.1: Maximum length of the story for each dataset

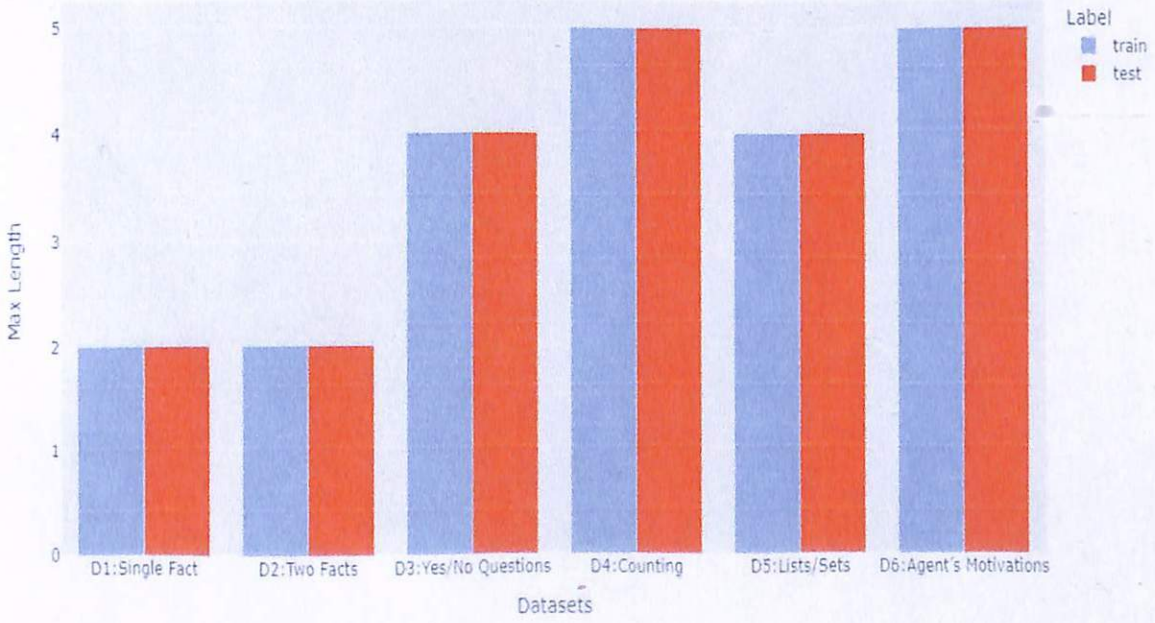

Figure 4.2: Maximum length of the question for each dataset.

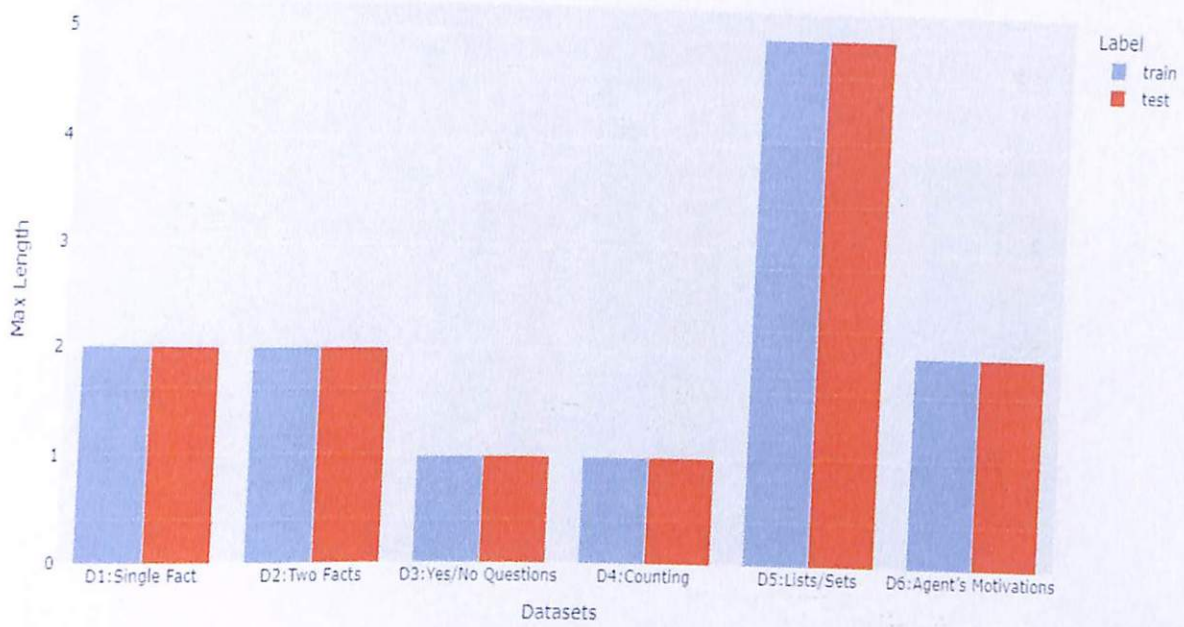

Figure 4.3: Maximum length of the answer for each dataset.

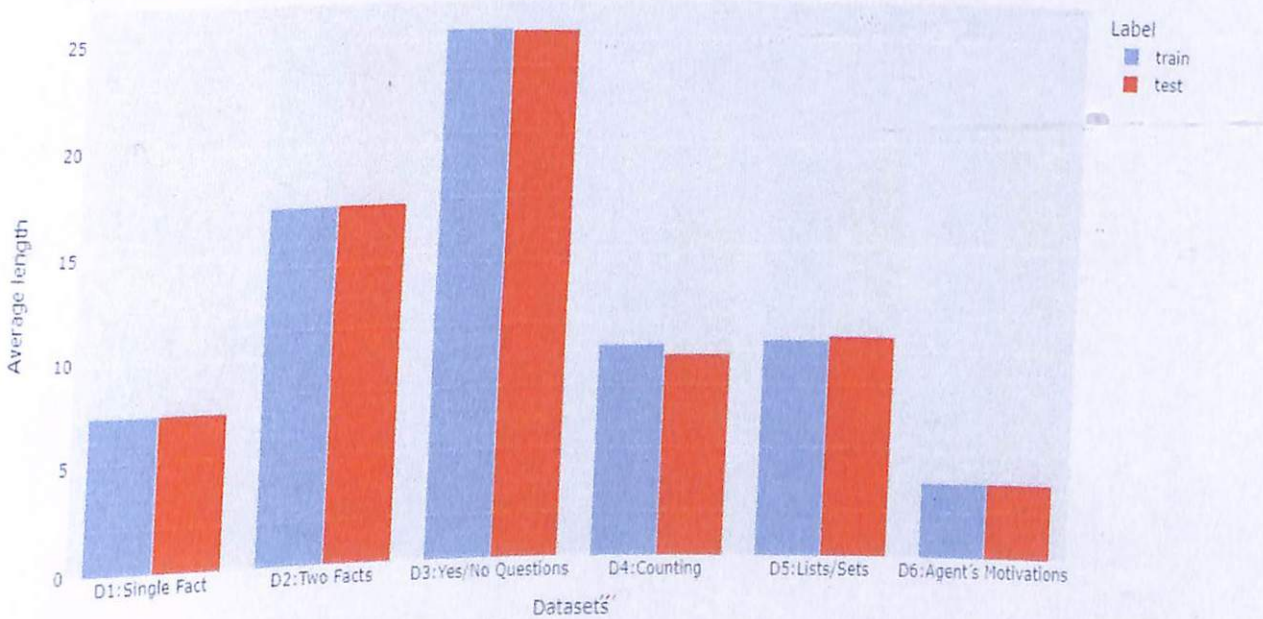

Figure 4.4: Average length of the story for each dataset.

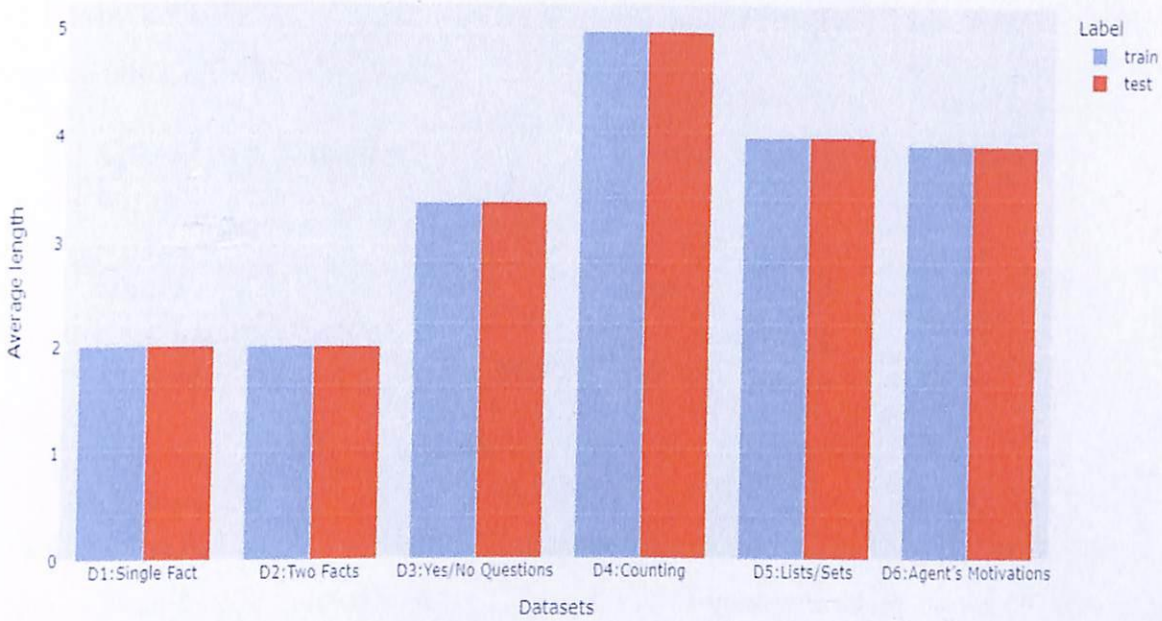

Figure 4.5: Average length of the question for each dataset.

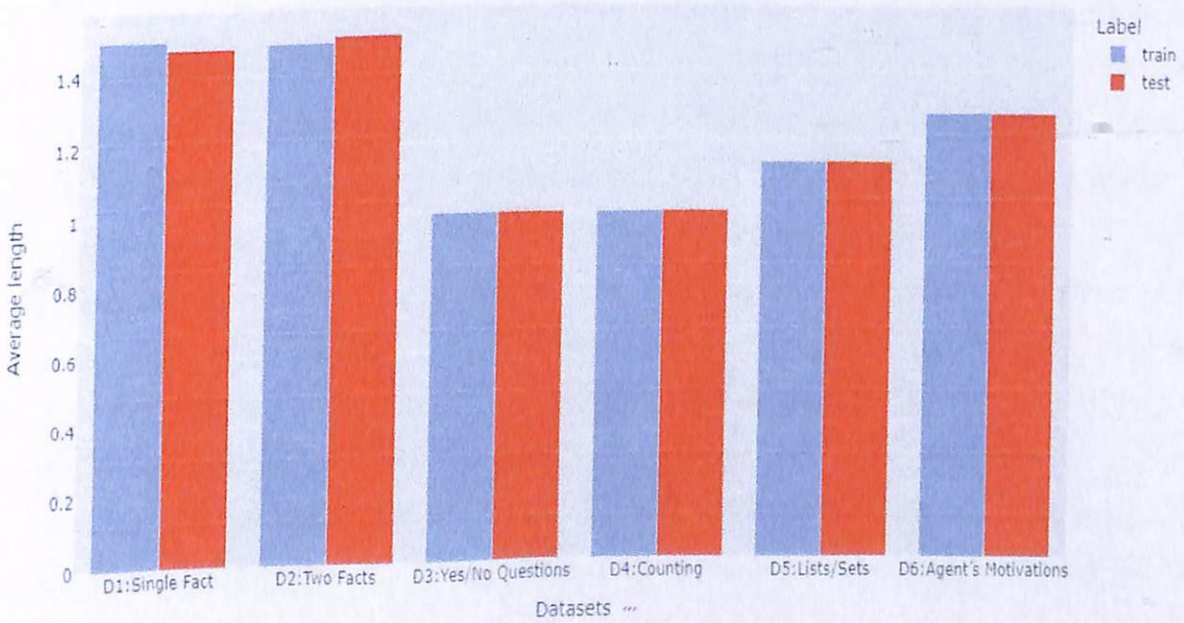

Figure 4.6: Average length of the answer for each dataset.

dataset. The results of this analysis are presented in Figures 4.1, 4.2, 4.3, 4.4, 4.5, and 4.6, respectively. By comprehending these lengths, we can optimize the model architectures and preprocessing steps accordingly. This ensures efficient processing and effective learning.

Question Classes	**Kazakh Question Words**
what	не
why	неге, неліктен
where	қайда
how many	неше, қанша
Interrogative sentence makers	ма, ме, па, пе, ба, бе

Table 4.2: Question classes in datasets.

In addition, we have provided the question classes in the Kazakh language that are utilized in our datasets (Table 4.2). These question classes serve as a categorization mechanism and aid in organizing and structuring the dataset.

4.2 Comparative Analysis

According to the results, the MemN2N model achieved the highest accuracy and F1 scores across the majority of datasets among the implemented models in a number of instances. This indicates that the model is proficient at extracting and storing information from the input data. On the other hand, BERT demonstrated competitive performance on select datasets. BERT's pre-trained language representation capabilities enable it to comprehend the contextual meaning of words and phrases, which can be advantageous for certain quality assurance tasks. However, its performance varied across datasets, indicating that its effectiveness may depend on the nature of the data and the difficulty of the questions. In general, the performance of the LSTM baseline model lagged behind that of the other models. This may be due to the inability of the LSTM architecture to capture long-term dependencies and contextual data. The outcomes are presented in Table 4.3 and Figure 4.7.

In addition, the performance of our Long Short-Term Memory (LSTM)-based MemN2N model was compared to that of a Gated Recurrent Unit (GRU)-based

MemN2N model. This was accomplished after preliminary experiments demonstrated the superior performance of the MemN2N model compared to other models.

Dataset	LSTM baseline		MemN2N		BERT	
	Acc	F1	Acc	F1	Acc	F1
Single Supporting Fact	48.19	46.24	76.39	76.59	64.30	64.64
Two Supporting Facts	27.30	27.29	51.40	50.64	18.10	18.10
Yes/No Questions	49.90	49.80	92.40	92.40	88.00	87.92
Counting	72.20	70.50	79.91	79.78	78.70	76.59
Lists/Sets	62.19	59.65	69.60	67.59	20.70	20.70
Agent's Motivations	91.20	91.16	91.20	91.78	84.30	84.16

Table 4.3: Accuracy and F1 score of each model for each dataset.

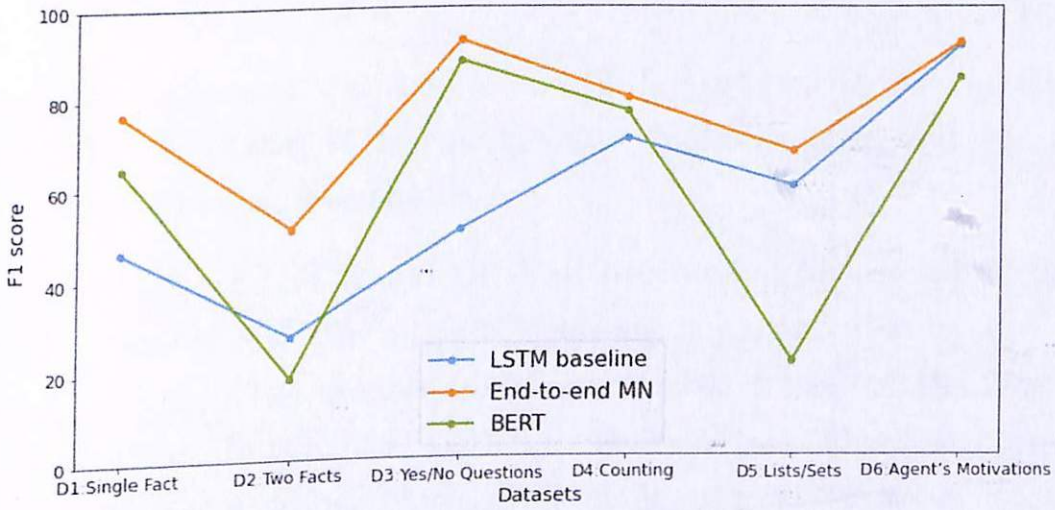

Figure 4.7: F1 score of each model for each dataset.

To determine the relative performance of the two MemN2N models, a comprehensive experiment was conducted. The objective of this experiment was to investigate the capabilities and nuances of various RNN structures in the MemN2N.The experiment parameters included a batch size of 32, 150 epochs, a dropout rate of 0.30 percent, and a learning rate of 0.01 percent. The results are presented in Table 4.4.

The impact of layer configuration on model performance was studied through observation. The study found that the accuracy scores of the two models were comparable. Nevertheless, the performance of the LSTM-based model was consistent across a variety of datasets. The experiment yielded significant findings

Layers	Dataset	GRU		LSTM	
		Acc	F1	Acc	F1
(32)	Single Supporting Fact	76.30	77.32	76.70	76.70
	Two Supporting Facts	31.20	30.42	30.39	30.36
	Yes/No Questions	83.39	83.29	92.41	92.41
	Counting	78.29	78.30	78.60	76.76
	Lists/Sets	67.00	66.73	69.19	67.35
	Agent's Motivations	91.20	91.78	92.23	91.76
(64)	Single Supporting Fact	75.65	75.59	76.89	76.96
	Two Supporting Facts	31.40	31.27	30.09	29.79
	Yes/No Questions	83.20	83.18	81.90	81.89
	Counting	77.70	77.00	79.19	78.06
	Lists/Sets	67.69	65.89	74.80	73.37
	Agent's Motivations	91.20	91.18	91.20	91.18

Table 4.4: Comparison of LSTM-based MemN2N and GRU-based MemN2N models.

concerning the performance of LSTM and GRU models with varying layer sizes when applied to QA tasks. The experimental results demonstrated that the performance of both models was similar.

The applicability of LSTM and GRU architectures depends on the characteristics of the dataset and the intended learning outcome. Recent research has demonstrated that LSTM models are exceptionally effective at identifying long-term dependencies. In contrast, GRU models have been found to have a more streamlined and computationally efficient architecture. The purpose of the experiment was to determine the optimal model configuration for our specific QA task by comparing and contrasting both alternatives.

Overall, the research results indicate that the MemN2N model is the most effective across the majority of datasets, suggesting its potential for our specific QA task. The objective of the study was to optimize the performance and accuracy of the approach by comparing and contrasting various models.

Chapter 5

Discussion and Conclusion

The present study provides a comprehensive examination and summary of the outcomes and implications derived from the investigation. The results of the experiment have been thoroughly examined and interpreted in detail. The research discussion provides a thorough examination of the experimental findings, with a focus on the noteworthy results and providing insightful observations on the models' efficacy.

Performance Comparison

In this study, we evaluated the performance of three different deep learning-based QA models, namely BERT, LSTM baseline, and MemN2N. The aim was to compare and contrast the effectiveness of these models in answering questions. The results of the assessment were analyzed in detail. The results of the study suggest that the MemN2N model outperformed the other models across most datasets, as evidenced by its higher accuracy and F1 score. The results indicate that the ability of the model to handle and analyze sequential data played a significant role in achieving better performance results. The pre-training of language representations has proven to be a successful technique in improving the performance of natural language processing models. BERT, in particular, has demonstrated impressive results in specific datasets. This can be attributed to its capability to pre-train language representations, which allows the model to better understand the nuances of language and improve its accuracy in various

tasks. The findings suggest that the LSTM baseline model exhibited subpar performance relative to the other models. This suggests the LSTM architecture may have limitations in effectively capturing contextual information and long-term dependencies.

We also aimed to enhance the performance of the MemN2N model, which has been reported to exhibit superior performance compared to other models. An experiment was conducted to achieve this objective. We compared the performances of our LSTM-based MemN2N and a GRU-based variant. The experiment sought to investigate the capabilities and nuances of different RNN architectures within the MemN2N framework. We focused on manipulating various parameters, including batch size, number of epochs, and dropout rates. The study's findings shed light on the comparative performance of the LSTM-based MemN2N model and the GRU-based model. The results indicate a slight variation in their performance. The results of this experiment contribute to the existing knowledge on the impact of different architectures and parameter combinations on the effectiveness of deep learning-powered QA models.

Dataset Characteristics

The analysis of performance variations across datasets can offer valuable insights into the impact of dataset characteristics on the performance of models. The models exhibited relatively better performance in handling datasets containing single facts and datasets comprising yes/no questions. The results suggest a higher level of accuracy and F1 scores. The datasets commonly entail uncomplicated information retrieval or binary decision-making tasks.

The study found that models encountered greater difficulties when dealing with datasets that involved more intricate questioning and reasoning demands, such as those that presented two-fact scenarios, counting tasks, and list/set-related questions. The superior performance of the MemN2N model on the datasets indicates its ability to effectively capture and process intricate sequential information.

Model Selection and Generalizability

The choice of the model depends on various factors, including dataset char-

acteristics and task requirements. The MemN2N, particularly the LSTM-based variant, demonstrated consistent and strong performance across different datasets, making it a suitable choice for our Kazakh language-based QA system. Its ability to retain information in a memory component and reason over sequential data proved advantageous for answering questions accurately.

However, it is important to note that the performance of the models may vary when applied to other languages or domains. The generalizability of the models should be carefully evaluated before deploying them in real-world applications. Fine-tuning and adapting the models to specific languages and domains may be necessary to achieve optimal performance.

Limitations and Future Directions

While our study provides valuable insights into the performance of different models, there are limitations that should be acknowledged. Firstly, the size of our datasets may limit the generalizability of the findings. Expanding the datasets and incorporating diverse sources of information could further enhance the performance and robustness of the models. Secondly, our evaluation focused on accuracy and F1-score as performance metrics, but other metrics such as precision and recall could provide additional insights.

Additionally, the usability and interpretability of the models should be considered in future work. Providing explanations or justifications for the generated answers could enhance user trust and acceptance of the QA systems. Furthermore, investigating the impact of different hyperparameters, such as batch size, learning rate, and model architecture variations, can help optimize the models' performance.

Conclusion

In conclusion, our study showcases the effectiveness of the MemN2N model, specifically the LSTM-based variant, for question answering in the Kazakh language. Through performance comparison and analysis, we gained insights into the strengths, weaknesses, and suitability of the implemented models for different datasets and tasks. These findings contribute to the development of robust QA

systems in the Kazakh language and provide a foundation for future research in this field.

Furthermore, our experimental results demonstrate the superior performance of the MemN2N model over the BERT model, which is renowned for its exceptional performance in various natural language processing tasks. Remarkably, the MemN2N model achieves better results while requiring less training time compared to fine-tuning pre-trained models. These results emphasize the importance of considering the specific task and dataset when choosing a model, rather than solely relying on its popularity. It is essential to evaluate the performance of models in the context of the given task and data in order to make informed decisions.

Bibliography

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

[2] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. nature, 521(7553):436–444, 2015.

[3] Balakrishnan Sathiyakugan. Learn natural language processing from scratch, Jul 2018. URL https://blog.goodaudience.com/learn-natural-language-processing-from-scratch-7893314725ff.

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

[5] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review, 65(6):386, 1958.

[6] Bernard Widrow and Marcian E Hoff. Adaptive switching circuits. Technical report, Stanford Univ Ca Stanford Electronics Labs, 1960.

[7] Arthur L Samuel. Some studies in machine learning using the game of checkers. IBM Journal of research and development, 3(3):210–229, 1959.

[8] Francke Peixoto. A simple overview of multilayer perceptron(mlp), Dec 2020. URL https://www.analyticsvidhya.com/blog/2020/12/mlp-multilayer-perceptron-simple-overview/.

[9] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. Communications of the ACM, 60(6):84–90, 2017.

[11] M K Gurucharan. Basic cnn architecture: Explaining 5 layers of convolutional neural network, Oct 2022. URL https://www.upgrad.com/blog/basic-cnn-architecture/.

[12] Yoon Kim. Convolutional neural networks for sentence classification, 2014.

[13] Xuezhe Ma and Eduard Hovy. End-to-end sequence labeling via bi-directional lstm-cnns-crf. arXiv preprint arXiv:1603.01354, 2016.

[14] Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In International conference on machine learning, pages 933–941. PMLR, 2017.

[15] Alex Graves. Generating sequences with recurrent neural networks. arXiv preprint arXiv:1308.0850, 2013.

[16] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In 2013 IEEE international conference on acoustics, speech and signal processing, pages 6645–6649. Ieee, 2013.

[17] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. Advances in neural information processing systems, 27, 2014.

[18] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 3128–3137, 2015.

[19] Ashkan Eliasy and Justyna Przychodzen. The role of ai in capital structure to enhance corporate funding strategies. Array, 6:100017, 07 2020. doi: 10.1016/j.array.2020.100017.

[20] John J Hopfield. Neural networks and physical systems with emergent collec-

tive computational abilities. Proceedings of the national academy of sciences, 79(8):2554–2558, 1982.

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

[22] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural computation, 9(8):1735–1780, 1997.

[23] Thorir Mar Ingolfsson. Insights into lstm architecture, Nov 2021. URL https://thorirmar.com/post/insight_into_lstm/.

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

[25] Alua Bilakhanova, Abdulla Ydyrys, and Nazerke Sultanova. Kazakh language-based question answering system using deep learning approach. Suleyman Demirel University Bulletin: Natural and Technical Sciences, 62(1): 113–121, 2023. ISSN 2709-2631. doi: 10.47344/sdubnts.v62i1.974. URL https://journals.sdu.edu.kz/index.php/nts/article/view/974.

[26] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. arXiv preprint arXiv:1611.01604, 2016.

[27] Shuohang Wang and Jing Jiang. Learning natural language inference with lstm. arXiv preprint arXiv:1512.08849, 2015.

[28] Danqi Chen, Jason Bolton, and Christopher D Manning. A thorough examination of the cnn/daily mail reading comprehension task. arXiv preprint arXiv:1606.02858, 2016.

[29] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. Advances in neural information processing systems, 28, 2015.

[30] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi.

Bidirectional attention flow for machine comprehension. arXiv preprint arXiv:1611.01603, 2016.

[31] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. arXiv preprint arXiv:1410.3916, 2014.

[32] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. Advances in neural information processing systems, 28, 2015.

[33] Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. Ask me anything: Dynamic memory networks for natural language processing. In International conference on machine learning, pages 1378–1387. PMLR, 2016.

[34] Caiming Xiong, Stephen Merity, and Richard Socher. Dynamic memory networks for visual and textual question answering. In International conference on machine learning, pages 2397–2406. PMLR, 2016.

[35] Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. Key-value memory networks for directly reading documents. arXiv preprint arXiv:1606.03126, 2016.

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

[37] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[38] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692, 2019.

[39] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. arXiv preprint arXiv:1909.11942, 2019.

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

[41] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. OpenAI blog, 1(8):9, 2019.

[42] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. Advances in neural information processing systems, 33:1877–1901, 2020.

[43] Cavide Balkı Gemirter and Dionysis Goularas. A turkish question answering system based on deep learning neural networks. Journal of Intelligent Systems: Theory and Applications, 4(2):65–75, 2021.

[44] Reji Rahmath K, PC Reghu Raj, and Rafeeque PC. Malayalam question answering system using deep learning approaches. IETE Journal of Research, pages 1–13, 2022.

[45] Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M Rush, Bart Van Merriënboer, Armand Joulin, and Tomas Mikolov. Towards ai-complete question answering: A set of prerequisite toy tasks. arXiv preprint arXiv:1502.05698, 2015.

[46] Peng Yutao, Wang Weihua, and Bao Feilong. Interactive mongolian question answer matching model based on attention mechanism in the law domain. In Proceedings of the 21st Chinese National Conference on Computational Linguistics, pages 896–907, 2022.

[47] Sardar Parhat, Gao Ting, Mijit Ablimit, and Askar Hamdulla. A morpheme sequence and convolutional neural network based kazakh text classification. In 2019 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), pages 1903–1906. IEEE, 2019.

[48] Nazerke Sultanova, Gulshat Kessikbayeva, and Yerbolat Amangeldi. Kazakh language open vocabulary language model with deep neural networks. In 2019

15th International Conference on Electronics, Computer and Computation (ICECCO), pages 1–4. IEEE, 2019.

[49] Gulizada Haisa and Gulila Altenbek. Multi-task learning model for kazakh query understanding. Sensors, 22(24):9810, 2022.

[50] Olzhas Makhambetov, Aibek Makazhanov, Islam Sabyrgaliyev, and Zhandos Yessenbayev. Data-driven morphological analysis and disambiguation for kazakh. In Computational Linguistics and Intelligent Text Processing: 16th International Conference, CICLing 2015, Cairo, Egypt, April 14-20, 2015, Proceedings, Part I 16, pages 151–163. Springer, 2015.

[51] Felipe Almeida and Geraldo Xexéo. Word embeddings: A survey. arXiv preprint arXiv:1901.09069, 2019.

[52] URL https://huggingface.co/docs/transformers/model_doc/bert.

[53] URL https://www.tensorflow.org/api_docs/python/tf.

[54] Keras Team. Keras documentation: Optimizers. URL https://keras.io/api/optimizers/.

Appendix A

Appendices

A.1 Adam optimizer algorithm

Algorithm 1 Adam algorithm

Require: α ▷ Learning rate

Require: $\beta_1, \beta_2 \in [0, 1)$ ▷ Exponential decay rates for the moment estimates

Require: $f(\theta)$ ▷ Stochastic objective function with parameters θ

Require: θ_0 ▷ Initial parameter vector

$m_0 \leftarrow 0$ ▷ Initialize 1^{st} moment vector

$v_0 \leftarrow 0$ ▷ Initialize 2^{nd} moment vector

$t \leftarrow 0$ ▷ Initialize timestep

while θ_t not converged **do**

  $t \leftarrow t + 1$

  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ ▷ Get gradients w.r.t. stochastic objective at timestep t

  $m_t \leftarrow \beta_1 \times m_{t-1} + (1 - \beta_1) \times g_t$ ▷ Update biased first moment estimate

  $v_t \leftarrow \beta_2 \times v_{t-1} + (1 - \beta_2) \times g_t^2$ ▷ Update biased second raw moment estimate

  $\overline{m_t} \leftarrow m_t / (1 - \beta_1^t)$ ▷ Compute bias-corrected first moment estimate

  $\overline{v_t} \leftarrow v_t / (1 - \beta_2^t)$ ▷ Compute bias-corrected second raw moment estimate

  $\theta_t \leftarrow \theta_{t-1} - \alpha \overline{m_t} / (\sqrt{\overline{v_t}} + \epsilon)$ ▷ Update parameters

return θ_t ▷ Resulting parameters

A.2 LSTM baseline

```
import pickle
import csv
import pandas as pd
import numpy as np
import string
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
from keras.models import Sequential, Model
from keras.layers import Embedding
from keras.layers import Input, Activation, Dense, Permute, Dropout
from keras.layers import add, dot, concatenate
from keras.layers import LSTM
from nltk.tokenize import sent_tokenize, word_tokenize
from keras.utils.vis_utils import plot_model
from sklearn.metrics import confusion_matrix, accuracy_score

cleanned_train = pd.read_csv(files_name)
cleanned_train

cleanned_test = pd.read_csv(files_name)
cleanned_test

all_data = pd.concat([cleanned_train, cleanned_test],  ignore_index=True)
vocab = set()
for i in range (len(all_data)):
    vocab |= set(word_tokenize(all_data['Cl_story'][i]) + word_tokenize(
    all_data['Cl_question'][i])+ [all_data['Answer'][i]])
vocab = sorted(vocab)
vocab_size = len(vocab) + 1
max_story_len = max([len(word_tokenize(all_data.iloc[i,0])) for i in range (
    len(all_data))])
max_question_len = max([len(word_tokenize(all_data.iloc[i,1])) for i in
    range (len(all_data))])
tokenizer = Tokenizer(filters=[])
tokenizer.fit_on_texts(vocab)
word_idx = dict((c, i + 1) for i, c in enumerate(vocab))
idx_word = dict((i+1, c) for i,c in enumerate(vocab))
word_idx

def vectorize_stories(data, word_index=word_idx, max_story_len=max_story_len
    ,max_question_len=max_question_len):
    # X = STORIES
    X = []
```

```
    # Xq = QUERY/QUESTION
    Xq = []
    # Y = CORRECT ANSWER
    Y = []

    for i in range (len(data)):
        x = [word_index[word.lower()] for word in word_tokenize(data.iloc[i
    ,0])]
        xq = [word_index[word.lower()] for word in word_tokenize(data.iloc[i
    ,1])]
        y = np.zeros(len(word_index) + 1)
        y[word_index[data.iloc[i,2]]] = 1
        X.append(x)
        Xq.append(xq)
        Y.append(y)

    return (pad_sequences(X, maxlen=max_story_len),pad_sequences(Xq, maxlen=
    max_question_len), np.array(Y))

inputs_train, queries_train, answers_train = vectorize_stories(
    cleanned_train)
inputs_test, queries_test, answers_test = vectorize_stories(cleanned_test)

num_classes = len(word_idx)+1

#MODEL
# Define input tensors for story and question sequences
input_story = Input(shape=(max_story_len,))
input_question = Input(shape=(max_question_len,))

# Define embedding layer for story and question
embedded_story = Embedding(input_dim=vocab_size, output_dim=max_story_len)(
    input_story)
embedded_question = Embedding(input_dim=vocab_size, output_dim=
    max_question_len)(input_question)

# Define LSTM layer for story and question with dropout
lstm_story = LSTM(units=64)(embedded_story)
lstm_story = Dropout(0.3)(lstm_story)  # Add dropout layer

lstm_question = LSTM(units=64)(embedded_question)
lstm_question = Dropout(0.3)(lstm_question)  # Add dropout layer

# Concatenate LSTM outputs from story and question
merged = concatenate([lstm_story, lstm_question])
```

```
# Define output layer
output = Dense(units=num_classes, activation='softmax')(merged)

# Define the model with multiple inputs and single output
model = Model(inputs=[input_story, input_question], outputs=output)

optimizer = tf.keras.optimizers.Adam(learning_rate=2e-5)

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['
    accuracy'])

# train
history = model.fit([inputs_train, queries_train], answers_train,batch_size
    =32,epochs=120,validation_data=([inputs_test, queries_test], answers_test
    ))

%matplotlib inline
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# Evaluate the model
loss, accuracy = model.evaluate([inputs_test, queries_test], answers_test)
print('Test Loss:', loss)
print('Test Accuracy:', accuracy)

from sklearn.metrics import precision_score, recall_score, f1_score,
    classification_report

# Evaluate the model on the test data
predictions = model.predict([inputs_test, queries_test])
predicted_labels = np.argmax(predictions, axis=1)
true_labels = np.argmax(answers_test, axis=1)

# Calculate evaluation metrics
precision = precision_score(true_labels, predicted_labels, average='weighted
    ')
recall = recall_score(true_labels, predicted_labels, average='weighted')
f1 = f1_score(true_labels, predicted_labels, average='weighted')
```

```
123
124 print("Precision: {:.2f}".format(precision*100))
125 print("Recall: {:.2f}".format(recall*100))
126 print("F1 Score: {:.2f}".format(f1*100))
```

A.3 End-to-end Memory Network (MemN2N)

```
import pickle
import csv
import pandas as pd
import numpy as np
import string
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
from keras.models import Sequential, Model
from keras.layers import Embedding
from keras.layers import Input, Activation, Dense, Permute, Dropout
from keras.layers import add, dot, concatenate
from keras.layers import LSTM
from nltk.tokenize import sent_tokenize, word_tokenize
from keras.utils.vis_utils import plot_model
from sklearn.metrics import confusion_matrix, accuracy_score

cleanned_train = pd.read_csv(files_name)
cleanned_train

cleanned_test = pd.read_csv(files_name)
cleanned_test

all_data = pd.concat([cleanned_train, cleanned_test],  ignore_index=True)
vocab = set()
for i in range (len(all_data)):
    vocab |= set(word_tokenize(all_data['Cl_story'][i]) + word_tokenize(
    all_data['Cl_question'][i])+ [all_data['Answer'][i]])
vocab = sorted(vocab)
vocab_size = len(vocab) + 1
max_story_len = max([len(word_tokenize(all_data.iloc[i,0])) for i in range (
    len(all_data))])
max_question_len = max([len(word_tokenize(all_data.iloc[i,1])) for i in
    range (len(all_data))])
tokenizer = Tokenizer(filters=[])
tokenizer.fit_on_texts(vocab)
word_idx = dict((c, i + 1) for i, c in enumerate(vocab))
idx_word = dict((i+1, c) for i,c in enumerate(vocab))

def vectorize_stories(data, word_index=word_idx, max_story_len=max_story_len
    ,max_question_len=max_question_len):
    # X = STORIES
    X = []
```

```
    # Xq = QUERY/QUESTION
    Xq = []
    # Y = CORRECT ANSWER
    Y = []

    for i in range (len(data)):
        x = [word_index[word.lower()] for word in word_tokenize(data.iloc[i
    ,0])]
        xq = [word_index[word.lower()] for word in word_tokenize(data.iloc[i
    ,1])]
        y = np.zeros(len(word_index) + 1)
        y[word_index[data.iloc[i,2]]] = 1
        X.append(x)
        Xq.append(xq)
        Y.append(y)

    return (pad_sequences(X, maxlen=max_story_len),pad_sequences(Xq, maxlen=
    max_question_len), np.array(Y))

inputs_train, queries_train, answers_train = vectorize_stories(
    cleanned_train)
inputs_test, queries_test, answers_test = vectorize_stories(cleanned_test)

#Model implementation
input_sequence = Input((max_story_len,))
question = Input((max_question_len,))

# Input gets embedded to a sequence of vectors
input_encoder_m = Sequential()
input_encoder_m.add(Embedding(input_dim=vocab_size,output_dim=64))
input_encoder_m.add(Dropout(0.3))

# This encoder will output:
# (samples, story_maxlen, embedding_dim)

# embed the input into a sequence of vectors of size query_maxlen
input_encoder_c = Sequential()
input_encoder_c.add(Embedding(input_dim=vocab_size,output_dim=
    max_question_len))
input_encoder_c.add(Dropout(0.3))
# output: (samples, story_maxlen, query_maxlen)

# embed the question into a sequence of vectors
question_encoder = Sequential()
question_encoder.add(Embedding(input_dim=vocab_size,
                               output_dim=64,
```

```
                                        input_length=max_question_len))
question_encoder.add(Dropout(0.3))
# output: (samples, query_maxlen, embedding_dim)


# encode input sequence and questions (which are indices)
# to sequences of dense vectors
input_encoded_m = input_encoder_m(input_sequence)
input_encoded_c = input_encoder_c(input_sequence)
question_encoded = question_encoder(question)


# shape: `(samples, story_maxlen, query_maxlen)`
match = dot([input_encoded_m, question_encoded], axes=(2, 2))
match = Activation('softmax')(match)

# add the match matrix with the second input vector sequence
response = add([match, input_encoded_c])  # (samples, story_maxlen,
    query_maxlen)
response = Permute((2, 1))(response)  # (samples, query_maxlen, story_maxlen
    )

# concatenate the match matrix with the question vector sequence
answer = concatenate([response, question_encoded])

# Reduce with RNN (LSTM)
answer = LSTM(32)(answer)  # (samples, 32)

# Regularization with Dropout
answer = Dropout(0.3)(answer)
answer = Dense(vocab_size)(answer)  # (samples, vocab_size)

# we output a probability distribution over the vocabulary
answer = Activation('softmax')(answer)
# optimizer = tf.keras.optimizers.Adam(learning_rate=2e-5)

# build the final model
model = Model([input_sequence, question], answer)
model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics
    =['accuracy'])

# train
history = model.fit([inputs_train, queries_train], answers_train,batch_size
    =32,epochs=150,validation_data=([inputs_test, queries_test], answers_test
    ))

```

```
123 loss, accuracy = model.evaluate([inputs_test, queries_test], answers_test)
124 print('Test Loss:', loss)
125 print('Test Accuracy:', accuracy)
126
127 from sklearn.metrics import precision_score, recall_score, f1_score
128
129 # Evaluate the model on the test data
130 predictions = model.predict([inputs_test, queries_test])
131 predicted_labels = np.argmax(predictions, axis=-1)
132 true_labels = np.argmax(answers_test, axis=-1)
133
134 # Calculate evaluation metrics
135 precision = precision_score(true_labels, predicted_labels, average='weighted
      ')
136 recall = recall_score(true_labels, predicted_labels, average='weighted')
137 f1 = f1_score(true_labels, predicted_labels, average='weighted')
138
139 print("Precision: {:.2f}".format(precision*100))
140 print("Recall: {:.2f}".format(recall*100))
141 print("F1 Score: {:.2f}".format(f1*100))
```

A.4 BERT

```
import tensorflow as tf
import pandas as pd
from transformers import TFBertTokenizer, TFBertForQuestionAnswering
from tensorflow.keras.losses import SparseCategoricalCrossentropy
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import classification_report
from sklearn.preprocessing import LabelEncoder

# Load the dataset from the CSV file
df_train = pd.read_csv(files_name)
df_test = pd.read_csv(files_name)

# Define hyperparameters
batch_size = 32
max_seq_length = 256
learning_rate = 2e-5


# Load the BERT tokenizer and model
tokenizer = TFBertTokenizer.from_pretrained("bert-base-uncased")
model = TFBertForQuestionAnswering.from_pretrained("bert-base-uncased")

# Tokenize the input data
train_encodings = tokenizer.batch_encode_plus(
    zip(df_train['Cl_question'], df_train['Cl_story']),
    truncation=True,
    add_special_tokens=True,
    max_length=max_seq_length,
    padding=True
)

test_encodings = tokenizer.batch_encode_plus(
    zip(df_test['Cl_question'], df_test['Cl_story']),
    add_special_tokens=True,
    truncation=True,
    max_length=max_seq_length,
    padding=True
)

train_answer_list = df_train.Answer.values
le = LabelEncoder()
train_answer_list=le.fit_transform(train_answer_list)

test_answer_list = df_test.Answer.values
test_answer_list=le.fit_transform(test_answer_list)
```

```
46
47 # Create TensorFlow datasets
48 train_dataset = tf.data.Dataset.from_tensor_slices((
49     dict(train_encodings),
50     train_answer_list
51 )).batch(batch_size)
52
53 test_dataset = tf.data.Dataset.from_tensor_slices((
54     dict(test_encodings),
55     test_answer_list
56 )).batch(batch_size)
57
58 # Set optimizer and loss function
59
60
61 optimizer = tf.keras.optimizers.Adam(learning_rate=2e-5, epsilon=1e-08,
       clipnorm=1.0)
62 loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
63 metric = tf.keras.metrics.SparseCategoricalAccuracy('accuracy')
64
65
66 # Compile the model
67 model.compile(optimizer=optimizer, loss=loss_fn, metrics=[metric])
68
69 # Fine-tuning loop
70 history = model.fit(train_dataset, epochs=5, validation_data=test_dataset)
71
72 %matplotlib inline
73 import matplotlib.pyplot as plt
74 print(history.history.keys())
75 # summarize history for accuracy
76 plt.plot(history.history['accuracy'])
77 plt.plot(history.history['val_accuracy'])
78 plt.title('Model accuracy')
79 plt.ylabel('Accuracy')
80 plt.xlabel('epoch')
81 plt.legend(['train', 'test'], loc='upper left')
82 plt.show()
83
84 # Evaluate on the test set
85 from sklearn.metrics import accuracy_score, precision_score, recall_score,
       f1_score
86 y_true = df_test['Answer'].values
87 y_pred = model.predict(test_dataset)[0].argmax(axis=-1)
88 y_pred = le.inverse_transform(y_pred)
89
```

```
accuracy=accuracy_score(y_true, y_pred)
precision = precision_score(y_true, y_pred, average='weighted')
recall = recall_score(y_true, y_pred, average='weighted')
f1 = f1_score(y_true, y_pred, average='weighted')

print("Accuracy: {:.2f}".format(accuracy*100))
print("Precision: {:.2f}".format(precision*100))
print("Recall: {:.2f}".format(recall*100))
print("F1 Score: {:.2f}".format(f1*100))
```