SOLVING TURKISH MATH WORD PROBLEMS BY SEQUENCE-TO-SEQUENCE ENCODER-DECODER MODELS

by

Esin Gedik

B.S., Computer Engineering, Yıldız Technical University, 2019

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2022

SOLVING TURKISH MATH WORD PROBLEMS BY
SEQUENCE-TO-SEQUENCE ENCODER-DECODER MODELS

APPROVED BY:

Prof. Tunga Güngör (Thesis Supervisor)	.....
Prof. Banu Diri	.....
Assist. Prof. Suzan Üsküdarlı	.....

DATE OF APPROVAL: 15.06.2022

ACKNOWLEDGEMENTS

First of all, I would like to express my thankfulness to my supervisor Prof. Tunga Güngör, who is one of the persons I respect the most and whose work I admire, for his continuous support and guidance throughout my research.

Thanks to my committee members, Prof. Banu Diri and Assist. Prof. Suzan Üsküdarlı for their valuable time and reviews.

Last but not least, I would like to express my deepest gratitude to my family, friends, and colleagues for the prodigious support and hope they have given to me. Thank you for believing me even the times that I do not.

ABSTRACT

SOLVING TURKISH MATH WORD PROBLEMS BY SEQUENCE-TO-SEQUENCE ENCODER-DECODER MODELS

It can be argued that solving math word problems (MWP) is a challenging task due to the semantic gap between natural language texts and mathematical equations. The main purpose of the task is to take a written math problem as input and produce a proper equation as output for solving that problem. This thesis describes a sequence-to-sequence (seq2seq) neural model for automatically solving MWPs based on their semantic meanings in the text. The seq2seq model has the advantage of being able to generate equations that do not exist in the training data. It comprises a bidirectional encoder to encode the input sequence and comprehend the problem semantics, and a decoder with attention to track semantic meanings of the output symbols and extract the equation. In this thesis, we investigate the successes of several pre-trained language models and neural models, including gated recurrent units (GRU) and long short-term memory (LSTM) seq2seq models. Our research is novel in the sense that there exist no studies in Turkish on this natural language processing (NLP) task that utilize the pre-trained language models and neural models. There is also no Turkish dataset designed to implement the neural models for MWP task. Due to the lack of data, we translated the well-known English MWP datasets into Turkish using a machine translation system. We performed manual adjustments, and built the corpora to contribute to the literature. Although Turkish is an agglutinative and grammatically challenging language to work on, our system correctly answers 71% of the questions in the corpora.

ÖZET

TÜRKÇE MATEMATİK PROBLEMLERİNİ KODLAYICI-KOD ÇÖZÜCÜ DİZİ-DİZİ MODELLERİYLE ÇÖZME

Matematiksel kelime problemlerini (MWP) çözmenin, doğal dil metinleri ve matematiksel denklemler arasındaki anlamsal boşluk nedeniyle zorlu bir görev olduğu söylenebilir. Görevin temel amacı, yazılı bir matematik problemini girdi olarak almak ve bu problemi çözmek için çıktı olarak uygun bir denklem üretmektir. Bu tez, metindeki semantik anlamlarına dayalı olarak MWP'leri otomatik olarak çözmek için diziden diziye (seq2seq) bir sinir modelini açıklamaktadır. Seq2seq modeli, eğitim verilerinde mevcut olmayan denklemleri üretebilme avantajına sahiptir. Giriş sırasını kodlamak ve problem semantiğini kavramak için çift yönlü bir kodlayıcı ve çıkış sembollerinin semantik anlamlarını izlemeye ve denklemi çıkarmaya yarayan bir kod çözücü içerir. Bu tezde, geçitli tekrarlayan birimler (GRU) ve uzun-kısa vadeli bellek (LSTM) seq2seq modelleri dahil olmak üzere, önceden eğitilmiş çeşitli dil modelleri ve nöral modellerin başarılarını araştırıyoruz. Araştırmamız, önceden eğitilmiş dil modellerini ve nöral modelleri kullanan bu doğal dil işleme (NLP) görevi hakkında Türkçe'de herhangi bir çalışma olmaması açısından yenidir. MWP görevi için nöral modelleri uygulamak için tasarlanmış bir Türkçe veri seti de bulunmamaktadır. Veri eksikliğinden dolayı, iyi bilinen İngilizce MWP veri setlerini makine çeviri sistemi kullanarak Türkçe'ye çevirdik. Literatüre katkı sağlamak için manuel ayarlamalar yaptık ve bir derlem oluşturduk. Türkçe sondan eklemeli ve gramer açısından üzerinde çalışılması zor bir dil olmasına rağmen, sistemimiz derlemdeki soruların %71'ini doğru yanıtlar.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

a	Alignment model
b	Bias
c	Memory cell
c_t	Context vector at time step t
$e_{t,i}$	Attention score of the i^{th} word at time step t
f	Nonlinear transition function, and forget gate
h_t	Hidden state at time step t
$\tilde{h_t}$	Attention vector at time step t
i	Input gate
o	Output gate
p	Aligned position
r	Reset gate
$Rate^{\rightarrow}_{num}$	Numerator unit from vertex u to vertex v in the unit dependency graph
$Rate^{\rightarrow}_{den}$	Denominator unit from vertex u to vertex v in the unit dependency graph
s	Hidden state of the decoder
u	Source vector in the unit dependency graph
U	Recurrent weight matrix from the hidden state to the hidden state; output word matrix in word2vec
U_r	Weight matrix from the hidden state to the hidden state for the reset gate
U_z	Weight matrix from the hidden state to the hidden state for the update gate
v	Destination vector in the unit dependency graph
V	Input word matrix in word2vec
W_{hh}	Weight matrix from the hidden state to the hidden state
W_{hy}	Weight matrix from the hidden state to the decoder output

$W_{h'h}$	Weight matrix from previous hidden state to current hidden state
W_r	Weight matrix for the reset gate
W_{xh}	Weight matrix from the input to the hidden state
W_z	Weight matrix for the update gate
x_t	Input at time step t
y_t	Output at time step t
z	Update gate
α	Attention weight
$\odot$	Element-wise multiplication
ϕ	Activation function
σ	Element-wise logistic sigmoid function

LIST OF ACRONYMS/ABBREVIATIONS

API	Application Programming Interface
BLEU	Bilingual Evaluation Understudy
BiRNN	Bidirectional Recurrent Neural Network
CASS	Copy and Alignment Mechanism to the Sequence-to-sequence
CBOW	Continuous Bag-of-Words
CLS	Classification Token
CNN	Convolutional Neural Networks
DRNN	Deep Recurrent Neural Network
GRU	Gated Recurrent Units
ID	Identifier
LSTM	Long Short-Term Memory
MWP	Math Word Problems
NER	Named Entity Recognition
NLP	Natural Language Processing
PAD	Padding Token
POS	Part-of-Speech
RNN	Recurrent Neural Network
SEP	Separating Token
Seq2prog	Sequence-to-program
Seq2seq	Sequence-to-sequence
SVM	Support Vector Machine
TF-IDF	Term Frequency–Inverse Document Frequency
UDG	Unit Dependency Graphs
UNK	Unknown
Word2vec	Word-to-vectors

1. INTRODUCTION

1.1. What is Math Word Problem Solving?

A math word problem (MWP) is a descriptive paragraph that depicts a real-world event or scenario, with mathematical relations stated in the narrative rather than explicit algebraic equations. Figure 1.1 represents an MWP and its generated equation. A semantic parsing component for building mathematical expressions receives a problem text as input. The output is an equation that symbolically describes the same mathematical relationship: "(36 + 34) * x = 420". The structure of the equation may vary according to the designed system, but the ultimate goal is to find the numerical answer of the problem. In other words, an answer is sought for the variable "x" in the equation.

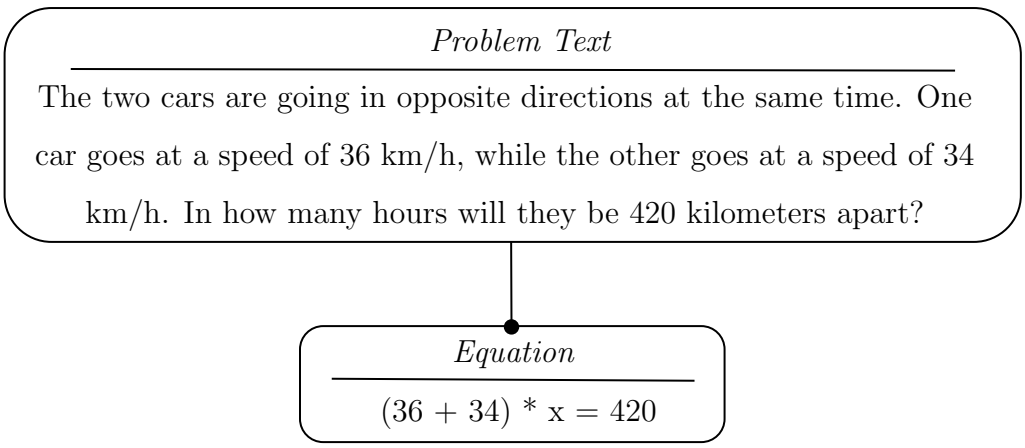


Figure 1.1. An MWP example

There are several types of MWPs. Those containing equations with one-unknown, where the result is a mathematical statement containing numbers and one or more arithmetic operators (+, −, /, *), are among the commonest. More sophisticated MWPs have systems of equations as output, include operators other than the four operations, or have problems involving more advanced calculations and domain knowledge.

1.2. Motivation

The system improvements that generate an automatic answer to an MWP have not progressed as successful as expected. One reason is that systems must map natural language text into machine-readable form in terms of semantic meaning of each number in the problem, and reasoning based on the comprehension to determine the equation. Another reason is that MWP complexity is determined by several dimensions, including reasoning, linguistic complexity, and domain knowledge.

For the last few decades, studies on solving MWPs using rule-based models have been conducted with a limited amount of data [1–5]. These models require pre-defined problem and equation templates. It is not possible to address problems that have never been encountered before or whose rules have not been stated. Because of this limitation, the MWP subtask is enhanced with the state-of-the-art neural models which provide further progress. There are many studies in the literature, particularly with English and Chinese MWP datasets comprised of manually constructed questions from websites [6–8]. However, there is neither a neural-based study nor corpus for solving Turkish MWPs, and the number of rule-based research is also relatively low [9–12].

1.3. Approach

Based on the previously stated motivation, in this thesis, we propose a deep neural solver to automatically solve Turkish MWPs. In contrast to prior template-based and statistical learning techniques, we utilize a sequence-to-sequence (seq2seq) model with an attention mechanism to directly translate MWPs to equation templates, with no complicated feature engineering. Seq2seq is a recurrent neural network (RNN) model, which employs an encoder to map the input sequence in a fixed-dimensional vector, followed by a decoder to generate the target sequence from this vector [13]. In our study, the encoder is intended to comprehend the semantics of the problem texts. In contrast, the decoder employs self-attention to vectorize the quantities and determine which symbol to generate next. There are two types of seq2seq models in our study.

One has the LSTM encoder-decoder, while the other has the gated recurrent units (GRU) encoder-decoder.

Computers can not understand the concepts of words. Therefore, word embeddings are used to represent words as real-valued vectors in a vector space. An embedding layer can be defined as the first hidden layer of a neural network model [14]. The problem texts are fed to the embedding layer before they are given to the encoder. In this layer, the embedding algorithms (word2vec and GloVe), and pre-trained language models (BERT, fastText, ConvBERT, and ELECTRA) are implemented for comparison.

Further, we also introduce new Turkish MWP corpora, by translating and combining English benchmark datasets to solve elementary level problems. After manual arrangements and preprocessing, we publish the corpora consisting of problem texts, equations, and answers customized to our model.

As a result of the comparative analysis, 71% accuracy and 72% bilingual evaluation understudy (BLEU) score are obtained with the seq2seq model using BERT embeddings and GRU encoder-decoder.

1.4. Contributions of this Thesis

The key contributions of the thesis are listed as follows:

- We construct the first Turkish MWP dataset to be used as a benchmark dataset in the MWP task.
- We deploy the first neural-based models to solve the MWP task.
- We achieve high accuracies and thus create a benchmark study for the future works on this task.

1.5. Outline

The organization of the thesis is as follows:

- Chapter 2 presents the literature review on the MWP domain.
- Chapter 3 covers the background theory of the associated models.
- Chapter 4 introduces the novel Turkish MWP dataset created from scratch.
- Chapter 5 describes the neural-based Turkish MWP solving system and methodologies.
- Chapter 6 provides the system analysis, model comparisons, model parameters, and the results of the experiments.
- Chapter 7 includes the final evaluations, describes the future work, and concludes the thesis.

2. RELATED WORKS

Previous works on automatically solving MWPs can be grouped into four main categories in terms of the models used: rule-based models, statistic-based models, tree-based models, and neural-based models. This chapter also covers Turkish MWP studies that have been carried out.

2.1. Rule-Based Models

Most previous works on solving MWPs focus on rule-based methods and feature engineering, which need more human sensitivity [2]. Since these features are frequently at the lexical level, it is unclear whether such models truly comprehend math problems. These models also include a substantial number of manually generated templates associated with specific problem spaces.

Bakman [3] solves free-format multi-step MWPs by developing a schema-based system and defining six types of schemas which are transfer-in-ownership, transfer-out-ownership, transfer-in-place, transfer-out-place, creation, and termination. After instantiating the schemas using lexical verb-based rules, the MWP system extracts the suitable equation. As an alternative, to provide a new perspective on the solution of MWPs, Liguda and Pfeiffer [4] suggest augmented semantic networks. According to this network, nodes indicate quantities while edges indicate transition states. For multi-step addition and subtraction MWPs, Yuhui et al. [5] demonstrate the problem text with frames to store the essential semantic relations. Each preposition in the problem text is categorized into a slot consisting of the object, quality (number), specification, time, and role. The main limitation of the approach is that it only covers addition and subtraction problems.

2.2. Statistic-Based Models

Earlier studies tend to use semantic parsing methods [15] or statistical learning methods [16–20] for solving the task. Although these models work effectively on large datasets, they could not preserve specific MWP properties.

Semantic parsing methods parse the problem text into an intermediate structured representation, typically annotated in the training set. In other words, they attemp to establish a direct mapping from the problem text to the equation structure. One of the most well-known studies that used this method binds the specific lexicon-based properties to the equation operators [15]. The equation is constructed by applying state transitions depending on the operators activated by the verb categories.

Statistical learning methods utilize typical machine learning models to recognize entities, quantities, and operators in the problem text and provide the numerical solution using a basic logic inference. They improve the semantic parsing methods by pre-defined logic templates. Kushman et al. [16] extract the templates about math expressions from the training answers, train models to select templates, and map quantities of the problem to the slots of the template. In another study [20], there are also templates with multiple unknown slots to select a suitable template, then complete both the number and unknown slots with the extracted information from the problem text. Mitra and Baral [18] categorize addition and subtraction problems and propose three problem templates which are part-whole, change, and comparison. Concerning the categories for problems, they generate the equation.

As a different perspective, Liang et al. [19] present a tag-based statistical MWP model to identify the desired operand and remove irrelevant quantities. The system analyzes the text corresponding to the syntactic tree and linguistic information, and then annotates with resolved co-reference chains. The text is transformed into logic form by using the mapping rules. For instance, “Marry borrowed 20 books from a bookstore.” can be written as “quan(q1, #, book)&verb(q1, borrow)&nsubj(q1,

Marry)=20", where "&" links each of the tags together. The quantity "20" is labeled by "q1" and is related to the associated tags to provide syntactic and semantic information. Upadhyay et al. [17] use a semi-supervised technique to forecast templates and match alignments by combining explicit and implicit supervision.

Statistical learning methods increase the complication of the text annotation with the appropriate template, which can be extremely costly when learning from large-scale datasets.

2.3. Tree-Based Models

An MWP can be shown with a tree structure, with the highest priority operators at the bottom and lowest priority operator at the top. The benefit of this approach is that no extra annotations such as equation templates or tags are required. Roy et al. [6] construct an equation tree by parsing the problem to generate the described mathematical equation in the sentence and map the variables in the equation to their corresponding noun phrases. In this approach, the tree comprises a series of predictions, eliminates irrelevant quantities, detects grounded variables, and generates the final equation tree. A support vector machine (SVM) classifier retrieves the relevant quantities and variables, then uses them as the leaf nodes in the equation tree.

The necessary linguistic information can be obtained through tokenization, sentence splitting, part-of-speech (POS) tagging, named entity recognition (NER), parsing, and co-reference resolution. Dries et al. [7] design an automated approach for solving probability problems by generating multiple parse trees for each of the sentences in the problem. They find all numbers in the problem with POS tags and capture the set of actions with a Bayesian network. On the other hand, declarative rules can be also defined to govern the translation of natural language description for basic operations such as addition, subtraction, multiplication, and division. Suppose that there is an MWP example like "Tom has 4 apples. John has 9 more apples than Tom. How many apples does John have?". Roy and Roth [8] indicate that the unit of 4

and 9 refer to apples in the sentences. The system may then conclude that these same type of units should be added together. Therefore, the output is an equation of 4+9. This approach shows how to link textual expression patterns formed from dependency parsing to specific operations.

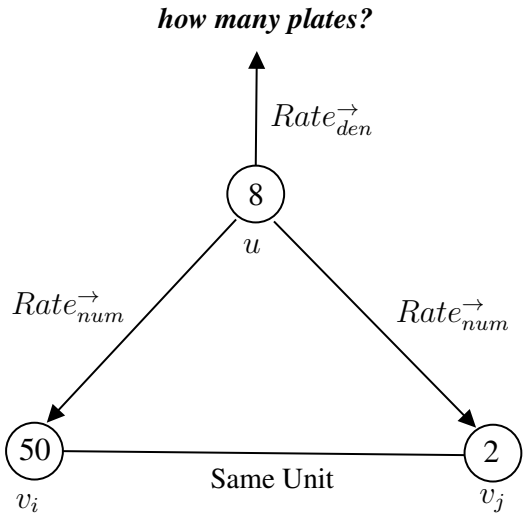


Figure 2.1. UDG example

Units related to the quantities provide information to support the reasoning, such as election results, news about casualties, or finance. Roy and Roth [21] aim to indicate the relationships among the units of different numbers and questions. The study suggests the unit dependency graphs (UDG), representing the dependencies between units of numbers described in a problem compactly. However, constructing a UDG involves extra annotation effort since it needs to train two classifiers for the nodes (vertices) and edges [22]. Assume that there is such an MWP: "Anna made 50 cookies for her birthday party. When she places the cookies on plates in groups of eight, she increases by 2 cookies. How many plates does she use in total?". In Figure 2.1, there is the UDG representation of the mentioned MWP. The source vertex u is labeled as *rate*, which refers to any quantity that is a measure that corresponds to one unit of another quantity. An undirected edge has the label *same unit* which is the connected vertices with the same unit. On the other hand, a directed edge from vertex u to vertex v can be either the numerator unit or the denominator unit. The numerator unit refers that the unit of vertex u matches the unit of the destination vertex v. The denominator unit means that the source of the vertex u matches the unit of the destination vertex

v. Here, "50" and "2" are connected via the *same unit*, thereby they can be added or subtracted. "8" is connected to a question by the denominator unit. It shows that an expression is divided by "8" for the unit of the final answer.

2.4. Neural-Based Models

As in many areas of NLP, neural models have recently been frequently preferred in solving MWPs, and successful results have been obtained. These utilize an encoder-decoder architecture and train it end-to-end without requiring customized rules or templates.

In the study conducted by Wang et al. [23], which is one of the most known and first applied neural approaches, an RNN model is developed to map the problem statements to the equation templates without feature engineering. They solve MWPs by generating the equation templates through a seq2seq model. The input of the seq2seq model is a word sequence after number mapping, and the output is an equation template. They use GRU as the encoder since it has fewer parameters and is less likely to be overfitted on a small dataset and a two-layer LSTM as the decoder.

There are two popular complementary directions for learning to solve MWPs which are semantic and purely data driven. Data-driven models easily learn to match problem texts in the MWP with the equations given sufficient training data. On the other hand, semantic methods figure out how to map problem texts to a semantic representation to generate an equation. Robaidek et al. [24] generate equation templates with seq2seq models with attention mechanism and evaluate the model with both LSTMs and convolutional neural networks (CNN) as the encoder and decoder.

As a different neural approach to improve the seq2seq model with reinforcement learning, the copy and alignment mechanism to the sequence-to-sequence (CASS) model is proposed to solve MWPs. Copy model directly copies the numbers in the problem text to the equation. Alignment means connection between the numbers in

the equations and the numbers in the problem description. The model could learn the alignment in a supervised way. Experimental results show that reinforcement learning leads to improve the performance [25]. According to another research [26], a new intermediate meaning representation scheme for solving MWPs reduces the semantic gap between natural language and equations. They claim that using the intermediate forms for training performs better than directly mapping problems to equation systems. Their baseline model is seq2seq with attention and copy mechanism. The encoder is implemented as a single-layer bidirectional RNN with GRUs. It reads words of the input text and produces a sequence of hidden states. The decoder receives the word embedding of the previous one and an attention function is applied to attend over the input words. At each step, the model decides whether to generate a word from target vocabulary or to copy a number from the problem description. They choose the seq2seq model because it provides a chance to generate equations of which problem types do not present in the training data.

2.5. Related Works on Turkish MWP

The primary motivation for this study is that deep learning models are not applied for solving MWPs in Turkish. However, several studies use rule-based and shallow models. The two earliest Turkish studies suggest a program which can solve primary-school-level MWPs [9, 10]. This program consists of several sub-tasks which are morphological analysis of each word in the input, syntactic analysis of each sentence, semantic analysis of words, and providing a description of the commonsense world large enough to allow correct answers to the problems. The commonsense knowledge required includes many items such as “Cows have 4 legs” and “There are 52 weeks in a year”. The basic idea is to have a template that will handle all sentences for the semantic categories. A semantic preprocessor rearranges the sentences with verbs that have the same semantic effect before finding the appropriate template. For example, the verbs “to get out”, “to die”, “to be sold”, “to sell” all indicate a decrease, but the noun phrase is in the subject position for “to get out” and in the direct object position for “to sell”. The preprocessor detects it and passes the components to the semantic

template in a standardized order.

Çakıroğlu [11] also presents an MWP solving system that uses semantic networks for storing data. The proposed system has three main steps, which are morphology, syntax, and semantics. The difference from other systems is that data is modeled as semantic networks in semantic analysis. This type of modeling realizes correct and fast understanding by reducing unnecessary data. During the morphological analysis, word types are determined, affixes and suffixes of the words are searched, and affix types are identified. The syntactic analysis is a hierarchical structure of the sentence units. The semantic networks in the implemented system can recognize the subjects, objects, cases, adverbs, and verbs in the sentence. Before the semantic analysis, the word groups are checked to determine whether the analyzer recognizes a sentence. The problem texts are grouped as addition, subtraction, division, and multiplication.

Another Turkish study investigates the morphological analysis of the words, the definition of the problem types (addition, subtraction, division, multiplication), and the computing of the results [12]. The problem types are determined by an XML document. The program can evaluate and conclude specific problem patterns that consist of no more than three sentences and contain no more than two numbers. In order to determine which pattern the problem belongs to, the question sentence is divided into words and numbers, and it is decided whether it is addition, subtraction, multiplication, or division. These procedures are carried out using keywords contained in XML. Since the keywords in XML are root or stem, the words in the question sentence should also have roots and stems. Zemberek, an open-source NLP library developed for Turkish languages, is used for this process.

3. BACKGROUND THEORY

This chapter gives a brief introduction to neural networks. In particular, we will discuss several RNN-based models, including recurrent language models, the encoder-decoder paradigm, and the seq2seq model with attention.

3.1. Recurrent Neural Networks

RNNs are a type of neural network architecture mainly used to detect useful patterns in a sequence of data. They have been used in state-of-the-art problems such as language modeling [27], text generation, speech recognition [28], image description generation, and video tagging. RNNs can process any length sequence by recursively applying a transition function to their internal hidden states for each token in the input sequence. They have a memory that tracks the history based on their long-distance characteristics [29–31].

RNNs consist of an input layer x, hidden layer h, and output layer y. The activation of the hidden state h at time step t is computed as a function f of the current input symbol x_t and the previous hidden states h_{t-1} as shown in Equation 3.1. The transition function f is defined as the product of an element-wise non-linearity with an affine transformation of both x_t and h_{t-1}.

$$h_t = f(x_t, h_{t-1}) \tag{3.1}$$

$$h_t = \phi(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \tag{3.2}$$

The activations of RNNs are calculated by Equation 3.2, where ϕ is the activation function which is generally a logistic sigmoid function or a hyperbolic tangent function, W_{xh} is the input-to-hidden-state weight matrix, W_{hh} is the hidden-state-to-hidden-state recurrent weight matrix, and $\mathrm{b_h}$ is the bias term [32].

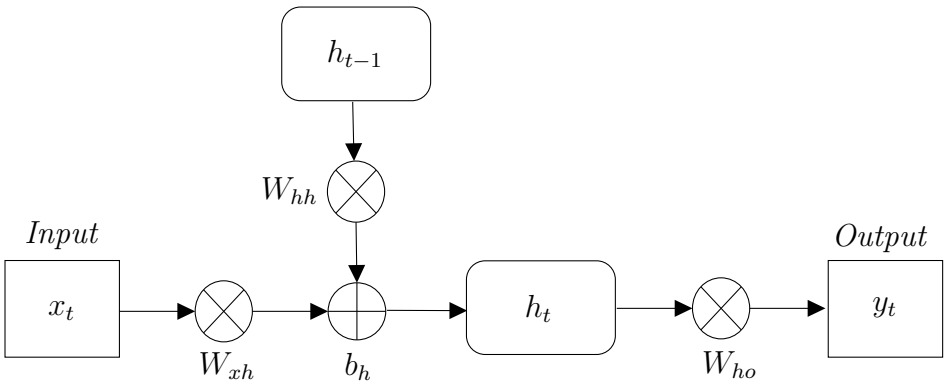


Figure 3.1. Architecture of RNN

RNNs are capable of constructing sentence vectors by making use of word vectors. In this scenario, the output layer at each time step is not necessary, and the sentence vector is generally the last hidden state vector h_t. Figure 3.1 demonstrates the architecture of RNNs.

3.1.1. Deep Recurrent Neural Networks

Compared to shallow RNNs, deep recurrent neural networks (DRNN) have showed the ability to learn feature representations at increasingly higher levels of abstractions and capture more non-linearities within the data in a variety of applications, including character-level and word-level language modeling [33]. DRNNs leverage previous output or historical information as inputs in addition to the input at the current time step. This makes them useful for recognizing sequential data features and using sequential patterns to forecast the next likely scenario [34]. The hidden state in the intermediate levels is made up of two states: one from the previous layer and one from the same layer.

$$h_t^l = \phi(W_{h'h}^l h_t^{l-1} + W_{hh}^l h_{t-1}^l + b_h) \tag{3.3}$$

In Equation 3.3, h_t^l indicates the hidden state in the l^{th} layer at time step t, $W_{h'h}^l$ specifies a weight matrix between the hidden state in the previous layer and the hidden

state in the current layer, and W_{hh}^{l} is also a weight matrix between the hidden states in the same layer.

3.1.2. Bidirectional Recurrent Neural Networks

One of the constraints of standard RNNs is that the information gathered at time step t only encodes information from the previous parts in the sequence [35]. Bidirectional recurrent neural networks (BiRNN) combine two separate RNNs. In one of the networks, the input sequence is fed in regular time order, while in the other the input is in reverse time order. At each time step, the hidden states of the two networks are typically concatenated, as shown in Figure 3.2. This structure allows the network to obtain a knowledge about the sequence in both the backward and forward directions.

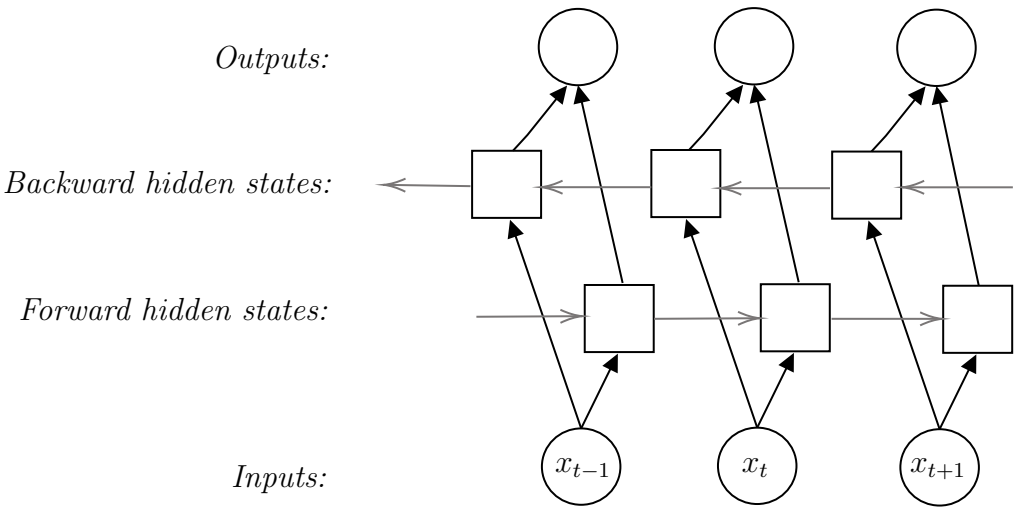


Figure 3.2. Architecture of BiRNN

In attentional seq2seq models, encoding input sequences with BiRNNs is also particularly successful since it helps the model attend to the context around a specific input token.

3.1.3. Long Short-Term Memory

LSTM is a kind of RNN that can learn and remember through long sequences of input data using gating units that govern the information about the network [36]. LSTM solves the following problems of RNNs.

- *Vanishing gradient*: The weights of the neural networks are updated using a gradient value. When the gradient is extremely small, it no longer contributes to the training. In the vanishing gradient problem, the gradient gets smaller as it propagates back over time.
- *Exploding gradient*: It is the case where gradients increase exponentially during backpropagation and cause learning divergence.
- *Short-term memory*: In the next time steps, information from distant previous time steps is ignored. It causes essential information to be lost.

Solving exploding gradients is reasonably straightforward using gradient clipping and truncated backpropagation [37]. However, vanishing gradient is a slightly more challenging problem due to adding skip connections through time. The main idea of LSTM used to solve these problems is replacing each regular node in the hidden layer with a memory cell [32]. Memory cells contain a node with a self-recurrent connection with a fixed weight. It allows gradients to update lightly through time.

Figure 3.3 visualizes a single LSTM memory cell. The input node accepts x_t and h_{t-1} as inputs and computes the activation [28]. The gates are the sigmoidal units that regulate data flow from one node to another and control the reading, writing, and resetting of the cells. The input to the memory cell c_t is multiplied by the activation of the input gate. The previous cell values are multiplied by the forget gate. The activated output of the memory cell is multiplied by that of the output gate, and h_t is obtained. The internal state of the cell is maintained with a recurrent connection.

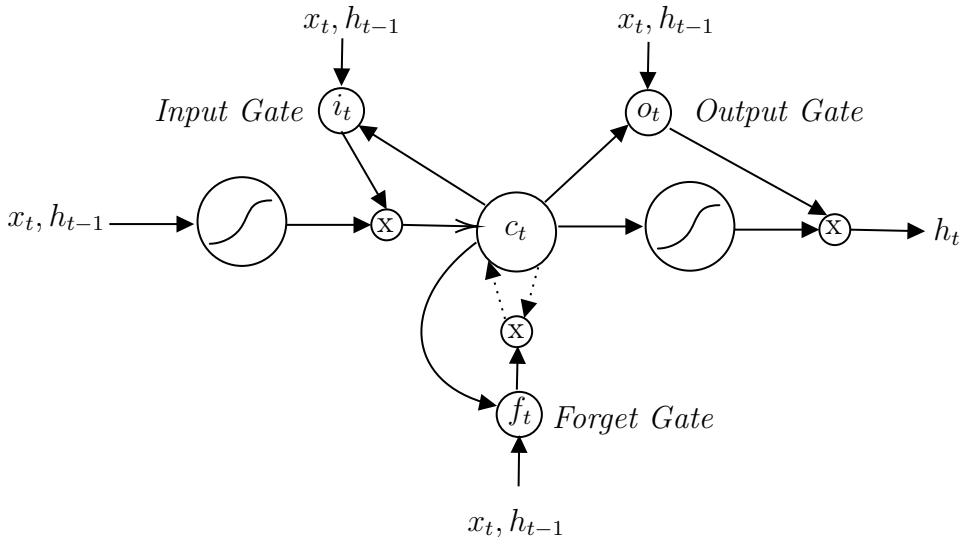


Figure 3.3. LSTM memory cell

3.1.4. Gated Recurrent Units

GRUs are popular alternatives to LSTM [38]. They adaptively reset or update the memory content. Similar to the forget and input gates of LSTM, GRUs have reset gate r_t and update gate z_t. The reset gate determines how much of the past information to forget. It is equivalent to combining the input gate and the forget gate in the LSTM. In other words, the reset gate is responsible for the short-term memory of the network. The update gate reflects how much previous information must be transferred to the next. It corresponds to the output gate in the LSTM. In contrast to LSTM, GRUs do not have a separate memory cell state c_t and they only have a hidden state h_t [30].

At time step t, the hidden state h_t of the GRU is calculated by Equation 3.4, where h_{t-1} is the previous activation, $\tilde{h}_t$ is the new candidate activation, z_t is the update gate and governs how much of the previous memory content is to be forgotten and how much new memory content is to be inserted. The update gate is derived from Equation 3.5 and based on the previous hidden state. x_t is the input vector, U_z denotes the state-to-state recurrent weight matrix, W_z is the update gate weight matrix, and σ is the logistic sigmoid function. This is a linear sum procedure between the current

state and the newly calculated state.

$$h_t = (1 - z_t)h_{t-1} + z_t\tilde{h}_t \tag{3.4}$$

$$z_t = \sigma(W_z x_t + U_z h_{t-1}) \tag{3.5}$$

$$\tilde{h}_t = tanh(W x_t + r_t \odot U h_{t-1}) \tag{3.6}$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1}) \tag{3.7}$$

The candidate activation $\tilde{h}_t$ can be obtained by Equation 3.6, where W and U are weight matrices, r_t is the reset gate, and $\odot$ is the element-wise multiplication. The reset gate r_t modifies the previous hidden state h_{t-1}. It enables GRUs to discard the previous hidden states if required, taking into account the previous hidden states and the current input. The reset mechanism makes optimal use of model capacity by permitting it to reset anytime the identified feature that is no longer required. The reset gate r_t is calculated as shown in Equation 3.7, where W_r and U_r are weight matrices.

3.2. Sequence to Sequence Model

Once the alignment between inputs and outputs is known in advance, RNNs can efficiently map sequences to sequences [13]. However, it is not clear how to apply RNNs to problems with input and output sequences of different lengths. The seq2seq model is an RNN architecture used in tasks such as machine translation, image captioning, and speech recognition. It solves sequence-based issues, particularly changeable input and output sizes. We can examine seq2seq models in two different backgrounds, the vanilla seq2seq model, which corresponds to the encoder-decoder and the seq2seq model with attention.

3.2.1. Encoder-Decoder Architecture

The vanilla seq2seq model combines an encoder and a decoder, whose roles are to encode the input into a state and decode this state to a target output, respectively [39]. A state is either a vector or a tensor. One RNN (usually LSTM or GRU) can be used as an encoder, and another as a decoder.

The encoder is a set of multiple recurrent units, each of which receives a single input sequence element, stores information for that element, and propagates it forward. For example, the input sequence in a question-answering problem is a collection of all words in the question. The encoder vector contains information for all input elements so that the decoder can provide correct predictions. In Equation 3.8, $h_t(encoder)$ represents the hidden states in the encoder. The final hidden state obtained from the encoder is the beginning hidden state of the decoder.

$$h_t(encoder) = \phi(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \tag{3.8}$$

$$h_t(decoder) = \phi(W_{hh}h_{t-1} + b_h) \tag{3.9}$$

$$y_i = h_t(decoder) * W_{hy} \tag{3.10}$$

The decoder is also a set of several recurrent units and predicts an output at time step t. Each recurrent decoder unit takes a hidden state from the previous unit and generates both an output and its hidden state [40]. In the same question-answering example, the output sequence consists of all of the words in the answer. Each word is represented by y_i, where i indicates the order of the word. The hidden states of the decoder are obtained by Equation 3.9. The output generated by the decoder is shown in Equation 3.10, where W_{hy} is the weights matrix from the hidden state to the decoder output. Figure 3.4 represents the proposed architecture.

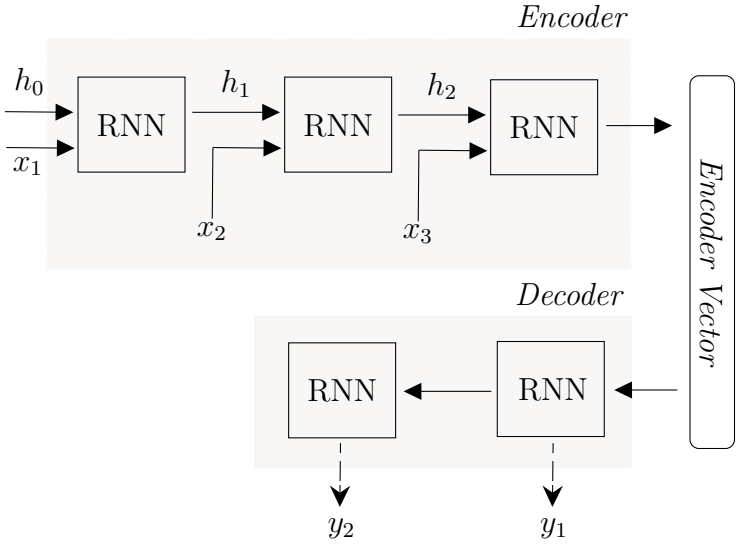


Figure 3.4. Visualization of the encoder-decoder seq2seq model

3.2.2. Attention Mechanism

According to Cho et al. [41], in the vanilla seq2seq models, the encoder generates a fixed length representation from a variable length input sentence, and the decoder creates a proper output based only on this representation. This model works reasonably well on short sentences with few unseen words, but its performance decreases dramatically as the sentence length and number of unseen words increase. In opposition, the attention mechanism handles this bottleneck directly by storing and using all hidden states of the input sequence during decoding. It accomplishes this by establishing a different mapping between each time step of the decoder output and all of the encoder hidden states. The attention mechanism implies that the decoder has access to the complete input sequence and may attend particular components from the sequence to generate decoder outputs [42]. There are two types of the attention mechanisms that have the same basic principles but differ in terms of architecture and computation, as explained in Section 3.2.2.1 and Section 3.2.2.2.

3.2.2.1. Bahdanau Attention Mechanism. As the first approach, Bahdanau et al. [43] suggest to align the decoder with the relevant input positions and implement the attention mechanism. The suggested method can also be described as the *additive attention*

since it conducts a linear combination of the encoder and decoder states. The steps applied in Bahdanau's attention mechanism are as follows:

- In contrast to the vanilla seq2seq model, which uses only the latest encoder hidden state, all hidden states of the encoder and decoder are utilized to build the context vector [44].
- The attention mechanism aligns the input and output sequences using a feed forward network to calculate an alignment score. It is beneficial to focus on the most important information in the source sequence.
- Depending on the context vectors related to the source positions and the previously created target words, the model predicts a target word.

Figure 3.5 shows the attentional seq2seq model proposed by Bahdanau et al. [43]. The encoder is formed of BiRNNs with two hidden state vector sequences [45]. A forward RNN reads the input sequence $(x_1, x_2, ..., x_{T_x})$ and generates the forward hidden states $\left(\overrightarrow{h_1},\ \overrightarrow{h_2},\ ...,\ \overrightarrow{h_{T_x}}\right)$. A backward RNN reads the input sequence in the opposite order $(x_{T_x}, x_{T_{x-1}}, ..., x_1)$ and produces the backward hidden state vector sequence $\left(\overleftarrow{h_1},\ \overleftarrow{h_2},\ ...,\ \overleftarrow{h_{T_x}}\right)$, where T is the total number of words. At each time step, the hidden state vectors are appended from BiRNNs to create a sequence of annotation vectors $(h_1,\ h_2,\ ...,\ h_T)$. For each word x_j, an annotation h_j is created by concatenating the $\overrightarrow{h}_j$ forward and $\overleftarrow{h}_j$ backward hidden states, where j is the order of the word in the sequence. The annotation of word x_j is shown in Equation 3.11. Consequently, the annotation includes representations of both the previous and following words.

The decoder is a unidirectional RNN that measures the impact of each annotation vector to the output prediction [31]. It receives each annotation and forwards the annotation and the previous hidden state of the decoder s_{t-1} to an alignment model a. After that, it creates an attention score $e_{t,j}$ representing how closely the inputs at the j^{th} position fit the output at the t^{th} position.The alignment model a combines s_{t-1} and h_j with an addition operation and calculates the attention score $e_{t,j}$ as indicated in Equation 3.12. Then, each attention score is put into a softmax function to get the

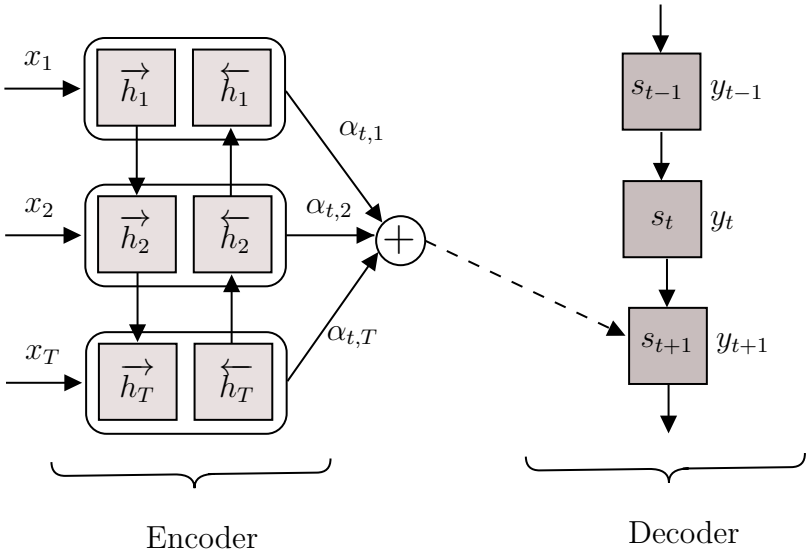


Figure 3.5. Bahdanau's attention mechanism

corresponding attention weight $\alpha_{t,j}$ by Equation 3.13.

$$h_j = [\overrightarrow{h_j^T}; \overleftarrow{h_j^T}]^T \tag{3.11}$$

$$e_{t,j} = a(s_{t-1}, h_j) \tag{3.12}$$

$$\alpha_{t,j} = exp(e_{tj}) / \sum_{j'=1}^{I} exp(e_{t'j}) \tag{3.13}$$

The context vector c_t is the weighted sum of the attention weight $\alpha_{t,j}$ and the hidden state h_j of the encoder according to Equation 3.14. Given the context vector c_t, previously predicted word y_{t-1}, and previous hidden state s_{t-1} of the decoder utilize the next hidden state s_t in Equation 3.15. For each target word y_t, a probability is conditioned, where g is a nonlinear function, as shown in Equation 3.16.

$$c_t = \sum_{j=1}^{T} \alpha_{t,j} h_j \tag{3.14}$$

$$s_t = f(s_{t-1}, c_t, y_{t-1}) \tag{3.15}$$

$$p(y_t | \{y_1, ..., y_{t-1}\}, x) = g(s_t, c_t, y_{t-1}) \tag{3.16}$$

3.2.2.2. Luong Attention Mechanism. An alternative attention mechanism, which we also used in our study, is proposed by Luong et al. [46]. Unlike Bahdanau, this approach is referred as the multiplicative attention [44]. It uses simple matrix multiplications, which are quicker and save space, to convert encoder and decoder states into the attention scores. Depending on where the attention is positioned in the source sequence, Luong suggests two types of attention approaches: *global* and *local*. The common point of these approaches is that they receive the hidden state h_t at the top layer of the LSTM as input at each time step in the decoder. On the other hand, they differ in the derivation of the context vector c_t.

A simple concatenation layer combines the information from the target hidden state h_t and context vector c_t to create an attention vector $\widetilde{h_t}$ as shown in Equation 3.17 [47]. The attention vector $\widetilde{h_t}$ is given to the softmax layer to obtain the predictive distribution defined in Equation 3.18 and current target word y_t.

$$\widetilde{h_t} = tanh(W_c[c_t; h_t]) \tag{3.17}$$

$$p(y_t|y_{<t}, x) = softmax(W_s\widetilde{h_t}) \tag{3.18}$$

The global attention approach pays attention to all the hidden states of the encoder as it derives the alignment score and the context vector. The global attention forms a variable length alignment weight vector a_t depending on the current target hidden state h_t and all source hidden states $\overline{h_s}$ at time step t as calculated in Equation 3.19.

$$a_t(s) = \frac{exp(alignmentScore(h_t, \overline{h_s}))}{\sum_{s'} exp(alignmentScore(h_t, \overline{h_{s'}}))} \tag{3.19}$$

The alignment score is a content-based function for which three possible possibilities are considered and can be examined in Equation 3.20. The first approach is similar to Bahdanau's model and concatenates the h_t and $\overline{h_s}$. Here, W_a is a weight matrix,

and v_a is a weight vector. Other two approaches apply the multiplicative attentions.

$$alignmentScore\ \left(h_t,\ \overline{h_s}\right)\ =\ \begin{cases} v_a^T\ tanh\left(W_a\left[h_t;\ \overline{h_s}\right]\right) & concat \\ h_t^T\ \overline{h_s} & dot \\ h_t^T\ W_a\overline{h_s} & general \end{cases} \tag{3.20}$$

The global context vector c_t is calculated as the weighted average of all the source hidden states $\overline{h_s}$ with the alignment weight vector a_t. The general architecture of the global attention approach is demonstrated in Figure 3.6.

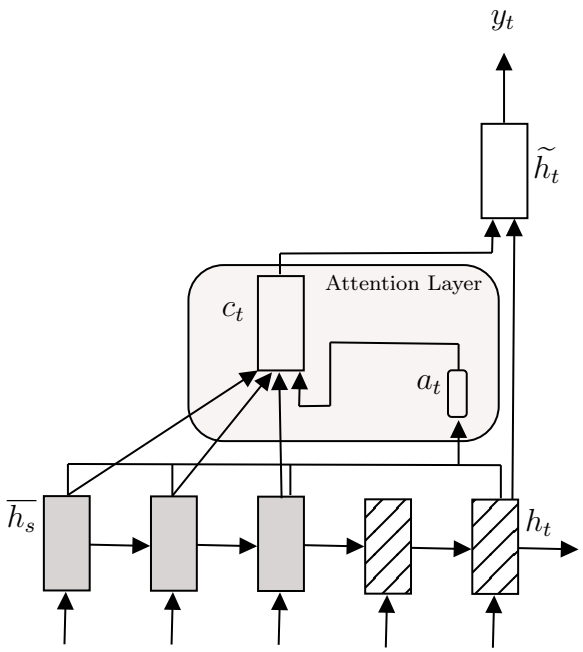


Figure 3.6. Luong global attention mechanism

The global attention approach is computationally expensive since it attends to all source words and may become unfeasible to process long sentences. In order to solve this shortcoming, a local attention approach, which relies on a small subset of source positions for each target word, is recommended. The model generates the aligned position p_t for each target word at time step t. The context vector c_t is obtained as a weighted average of the source hidden states inside the window $[p_t - D, p_t + D]$, where D is chosen experimentally. The aligned position p_t can be computed with two methods: *monotonic alignment* or *predictive alignment*. In the monotonic alignment, it

is considered that the source and target sentences are monotonically aligned. Therefore, p_t equals to t, and the alignment vector a_t is calculated with Equation 3.19.

On the other hand, the predictive alignment predicts an aligned position p_t by using the model parameters, which are v_p and W_p, and multiplying them with the length of the source sentence S. The aligned position p_t is computed by the predictive alignment as shown in Equation 3.21.

$$p_t = S.sigmoid(v_p^T tanh(W_p h_t)) \tag{3.21}$$

In contrast to the global approach, the local alignment vector a_t has a fix dimension $\in \mathbb{R}^{2D+1}$. Figure 3.7 illustrates the local attention model. As a summary, the model tries to predict the aligned position p_t of the current target word. The window $[p_t - D, p_t + D]$ for the source position p_t helps to determine the context vector c_t. The alignment vector a_t (as the local weight) is formed by the source states $\overline{h_s}$ within the window and the current target state h_t.

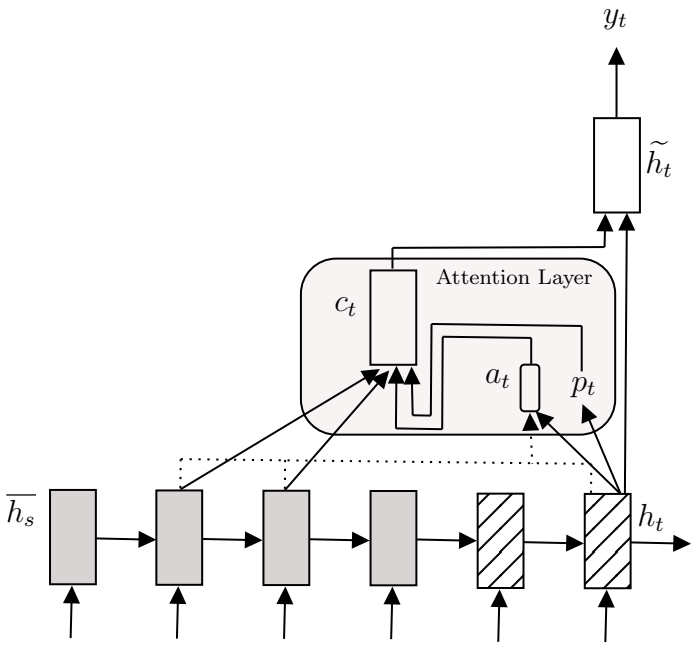


Figure 3.7. Luong local attention mechanism

3.3. Word Embeddings & Pre-trained Language Models

It is a well-known fact that computers are not competent in understanding texts. Nevertheless, thanks to the ability of computers to process numerical data, *word embedding* models have emerged that convert textual data into numerical form. These models are word vector representation techniques in a lower-dimensional space. They can capture the context of a particular word, the syntactic and semantic similarity of that word and its relation to other words [48]. These word embeddings are vital as they are used as the input to machine learning and deep learning models.

The classical word representation methods such as bag-of-words, countVectorizer, and term frequency-inverse document frequency (TF-IDF) depend on the word count in the sentence. The vector size in these methods is equal to the number of elements in the vocabulary. A sparse matrix is occured when the majority of the elements are zero. Also, these methods do not preserve any syntactical or semantic information. For the learning algorithms, the large input vectors require a large number of weights, resulting in high computing requirements. The state-of-the-art word embedding models, which will be discussed in the next sub-sections, solve these issues.

3.3.1. Word2vec

Word-to-vectors (word2Vec) is a common approach for learning word embeddings with a shallow neural network and was proposed by Mikolov et al. [49]. Word2Vec employs two iteration-based models to generate the vector representations of the words.

The first model is *continuous bag-of-words* (CBOW), which predicts the current word relying on the context [50]. It estimates the center word from its surrounding context. This approach maximizes the likelihood of a word existing in a given context via Equation 3.22, where w_i is the word at i^{th} position in the sentence and c is the

window size.

$$P(w_i|w_{i-c}, w_{i-c+1}, ..., w_{i-1}, w_{i+1}, ..., w_{i+c-1}, w_{i+c}) \tag{3.22}$$

It produces a model that depends on the continuous distributed representation of the context. Assume that W is a vocabulary containing all the words. The CBOW model creates a randomly initialized input word matrix denoted by $V \in \mathbb{R}^{N \times |W|}$. For the input word v_i, the i^{th} column of the matrix V indicates an N-dimensional embedding vector. The CBOW model also constructs a randomly initialized output word matrix called as $U \in \mathbb{R}^{|W| \times N}$. The j^{th} row of the output matrix U has an N-dimensional embedding vector for the output word u_j. The one-hot-encoding for each word is preferred to create word vector embeddings. The V^T is used to handle the relevant word vector embeddings in the N-dimensional space. The U^T is adopted to an input word vector to produce a score vector. It is then added to a softmax function to transform the score vector into a probability vector in the W-dimension. The goal of this operation is to produce a probability vector that corresponds to the vector representation of the output word.

The second model of the word2vec is *skip-gram* [51]. Unlike the CBOW, the skip-gram model tries to estimate the surrounding context words given a center word. It is beneficial to develop high-quality distributed vector representations which contain a massive number of syntactic and semantic words. According to Equation 3.23, the model maximizes the likelihood of the context words for a given center word.

$$P(w_{i-c}, w_{i-c+1}, ..., w_{i-1}, w_{i+1}, ..., w_{i+c-1}, w_{i+c}|w_i) \tag{3.23}$$

The optimization process is identical to the CBOW model, but the order is reversed for the context and center words. Figure 3.8 clearly shows the primary difference between the CBOW and skip-gram models.

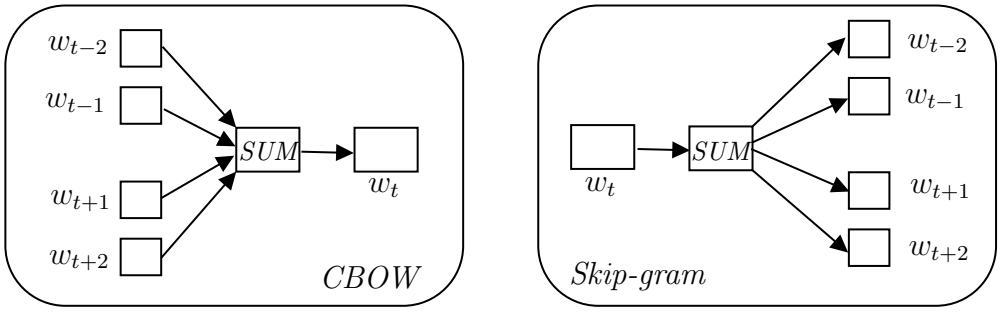


Figure 3.8. CBOW and skip-gram architectures

3.3.2. GloVe

GloVe is an unsupervised learning algorithm developed by Pennington et al. [52] to generate word embeddings by aggregating global word co-occurrence matrices. The primary idea of the GloVe model is to use statistics to identify the relationship between words. Unlike the occurrence matrix, the co-occurrence matrix indicates how frequently a particular word pair appears together. GloVe merges two approaches to obtain the word vectors. The first approach is *global matrix factorization* for representing the classical vector space models. On the other hand, the *local context window* is for the local context information of the words such as word2vec uses.

3.3.3. fastText

fastText makes explicit use of subword information and allows the embeddings for rare words to be represented appropriately [53]. It uses the skip-gram model, in which each word is denoted as a bag of character n-grams or subword units. The character n-grams are assigned to the vector representations, and the average of these vectors yields the final expression of the word. According to Wang et al. [50], the fastText model considerably increases the performance of the syntactic tasks, but not remarkably on the semantic problems.

3.3.4. BERT

BERT is a pre-trained language representation model, published by Devlin et al. [54]. BERT utilizes a transformer model and an attention model in which each output element is linked to each input element, and the weights between them are dynamically computed according to the connections [55]. The most straightforward architecture of the transformer consists of two distinct mechanisms: an encoder that reads the input text and a decoder that generates a prediction for the input. Since the purpose of BERT is to build a language model, only the encoder mechanism is sufficient. The encoder reads the complete sequence of words at once, in contrast to the directional models, which read the text input sequentially from left to right or right to left, but could not do both at the same time. As a result, the transformer model is regarded bi-directional. It enables the BERT model to learn the word's context depending on its surroundings.

Thanks to bidirectional ability, BERT is pre-trained on two NLP tasks: *masked language modeling* and *next sentence prediction*. The goal of the masked language model is to conceal a word in a sentence and then have the system guess what word is hidden based on its context. The next sentence prediction estimates whether two provided sentences have a logical and sequential link or whether their relationship is purely random. BERT outperforms previous embedding models such as word2vec and GloVe, which are limited in interpreting context and ambiguous words. BERT successfully solves ambiguity, which is one of the most difficult problems in NLP.

Given a set of tokens $X = (x_1, x_2, ..., x_n)$, BERT model pads the input with the classification *[CLS]* and separating *[SEP]* tokens [56]. A single vector containing the whole input sentence must be supplied to the classifier of the BERT for the classification tasks. In BERT, the hidden state of the first token is used to cover the entire sentence. In order to provide this, the *[CLS]* token is manually inserted at the beginning of the input sentence. In the next sentence prediction task, the *[SEP]* token is used to inform the model where the first sentence ends and the second sentence begins. When it is a

classification task, the *[SEP]* token is added to the end of the input sentence, as each input sample contains a single text sentence. The BERT model takes fixed length of sentences as inputs. For sentences shorter than a certain threshold, the padding token *[PAD]* is added to provide the same length.

In a pre-trained model, each token (word) has a unique identifier. Therefore, it is necessary to convert each token in the input sentence to its associated unique identifier in the fixed vocabulary of the pre-trained model. The tokens that do not have a unique identifier in the vocabulary are mapped with the identifier of the unkown token *[UNK]*. However, transforming all unseen words to the *[UNK]* token destroys information from the input text. To solve this problem, BERT employs the WordPiece tokenization method, which divides a word into several subwords, allowing the model to better represent both the frequently encountered tokens and also the unseen tokens.

3.3.5. ConvBERT

BERT depends on the global self-attention block, resulting in a huge memory footprint and computation cost [57]. It is observed that although all attention heads query the entire input sequence to construct the attention map from a global perspective, some heads are only required to learn local dependencies. Therefore, ConvBERT, a new span-based dynamic convolution model, is recommended to convert the self-attention heads to the local model dependencies. The state-of-the-art convolution heads and the remaining self-attention heads create a new mixed-attention block that is more efficient at learning both global and local contexts. The ConvBERT model is created by equipping the BERT model with the new mixed-attention block.

3.3.6. ELECTRA

ELECTRA is a self-supervised language representation model that trains two transformer models, the generator and the discriminator [58]. The generator replaces tokens in a sequence and is trained as a masked language model. The discriminator

attempts to determine which tokens in the sequence are replaced by the generator. Since ELECTRA is a pre-training approach, no modifications are made to the fundamental BERT model. The only manipulation is the separation between the hidden and embedding dimensions. An extra linear layer is utilized to project embeddings from the embedding dimensions to the hidden dimension. The linear layer is not used if the embedding dimension is the same as the hidden dimension.

4. DATASETS

Due to the rise of the use of neural models in the MWP task, the need for datasets containing more data and comprehensive equations has increased. The simplest MWPs have a text question, a mathematical equation that includes numbers and one or more basic arithmetic operators, which are (+, −, /, *). More complex MWPs have systems of equations as output, include other operators like square root and exponential expression, or contain more advanced topics and specialized information such as volume calculation and physics knowledge. Recent studies seem to focus on solving complex and challenging MWPs, such as geometry problems [59, 60], multiple unknown linear problems [61], and IQ problems [62].

4.1. Available Datasets

As we mentioned in the previous sections, mostly English and some Chinese studies are pioneers in the MWP task. Therefore, almost all of the available datasets are prepared in these languages. A list of prior benchmark datasets for MWP solving is provided below.

- *Alg514*: It is published by Kushman et al. [16] and consists of 514 algebra word problems annotated with linear equations. The problems with lack of knowledge or requiring explicit background knowledge are removed.
- *Verb395*: Hosseini et al. [15] created it by combining three diverse datasets. The problems include combinations of subtractions and additions, money word problems, and one-unknown equations. In total, there are 395 word problems.
- *Dolphin1878*: The data collection is built by Shi et al. [63] and includes 1878 problems crawled from algebra.com and Yahoo!.
- *Dolphin18K*: It is a large-scale and challenging dataset with 18460 problems, and 5871 equation templates [61]. Since the dataset was created from online forums, there may be errors in the annotations and answers.

- *DRAW1K*: It is a dataset consisting of 1000 algebra word problems [64]. It ensures the alignment between numerical coefficients in the equations and the numbers in the text.
- *SingleEQ*: There are 508 grade-school algebra word problems, including addition, subtraction, multiplication, division, and non-negative rational numbers [65].
- *AQuA*: It is another large-scale dataset with 100000 algebraic word problems and is published by Ling et al. (DeepMind) [66]. Each problem consists of four parts, which are question, answer options (A, B, C, D, and E), rationale (the natural language description of the solution), and correct answer option.
- *Math23K*: The dataset is composed of 23161 Chinese MWPs and 2187 templates which are suitable for elementary schools. These are all linear algebra problems with only one unknown variable [23].
- *MAWPS*: It is a frequently used English benchmark dataset containing equation templates and 3320 questions [67]. However, in some research [68], a simplified version with only 1920 questions is used.
- *ASDiv-A*: It is a diverse corpus in terms of lexicon patterns and wide problem types. Miao et al. [69] suggest that the dataset diversity is more crucial than the size to measure the actual performance of an MWP solver. They compute the diversity of ASDiv-A with a lexicon usage diversity metric that uses BLEU. It is an arithmetic subset of original ASDiv dataset, which requires also additional domain knowledge.
- *SVAMP*: It is a challenging and newly introduced dataset by Patel et al. [68]. It is created by injecting several types of modifications into a set of seed problems derived from the ASDiv-A dataset. Each equation template is converted to prefix form and all numbers are masked with a meta symbol.
- *MathQA*: This is a large-scale dataset of 37200 English multiple-choice MWPs from geometry, physics, probability, and gain-loss domains and improves the AQuA dataset by annotating the rationales, which means explanatory solutions [70].

The general statistics of the MWP datasets are summarized in Table 4.1, and

samples are given in Table A.1.

Table 4.1. Statistics of MWP datasets.

	# Problems	**Operators**	**Problems Types**	**Language**
Alg514	514	+, −, /, *	algebra, linear	English
Verb395	395	+, −	addition, subtraction	English
Dolphin1878	1878	+, −, /, *	number word	English
Dolphin18K	18460	+, −, /, *	linear, nonlinear	English
DRAW1K	1000	+, −, /, *	algebra, linear	English
SingleEQ	508	+, −, /, *	linear	English
AQuA	100000	+, −, /, *	algebra	English
Math23K	23161	+, −, /, *	algebra, linear	Chinese
MAWPS	1920	+, −, /, *	algebra	English
ASDiv-A	1218	+, −, /, *	algebra, linear	English
SVAMP	1000	+, −, /, *	algebra, linear	English
MathQA	37200	+, −, /, *	probability, physics, geometry, gain-loss	English

4.2. Turkish MWP Benchmark Datasets

In order to eliminate the lack of MWP studies in Turkish, we converted four English datasets, which are MAWPS, ASDiv-A, SVAMP, and MathQA, into Turkish using a machine translation system in this thesis. To create an MWP dataset from scratch, normally a significant number of questions are collected from websites or textbooks, the equations are created for the questions, and data is served in an acceptable format to train the neural model. Instead of this process, we preferred to translate existing English benchmark MWP datasets into Turkish to compare the performance of the models using the questions of the same difficulty but in different languages. Using an existing dataset translated from one language to another saves time and provides

the opportunity for multi-lingual question-answering systems.

We generated two distinct datasets which differ in their complexity levels. The details of the relevant datasets are explained in Section 4.2.1 and Section 4.2.2. Within the scope of the MWP task, our datasets are the first comprehensive benchmarks datasets created in Turkish and will contribute to future works.

4.2.1. Combined Dataset from MAWPS, ASDiv-A, and SVAMP

The first dataset is formed by combining the MAWPS, ASDiv-A, and SVAMP datasets, all containing elementary school arithmetic questions. These three datasets were chosen to merge since the problem sentences do not require extensive handling, the question types are comparable, and the mathematical equations can be interpreted. In total, 4164 MWP data are provided by adding a few manual questions.

Only the problem texts in the datasets have been translated. In the case of ASDiv-A and SVAMP, the problem text was created by combining the body and question fields. The SVAMP is required the most preprocessing operation before the translation out of the three. As can be seen in Table A.1, in the problem texts of the SVAMP, the punctuation marks should be appropriately placed, and the extra spaces or foreign characters should be eliminated for a good translation. The numbers in the questions of the original version of the SVAMP were replaced with the *numberX* tags, where X is the rank of that number among all the numbers in the sentence. As a result of the tests, the quality of the translations with tags is not satisfactory. For this reason, the numbers field in the dataset and the numberX tags in the sentences have been substituted based on their indices. The arranged problem texts were extracted from the dataset, divided into chunks of 300 due to the restrictions of the application programming interface (API), and translated into Turkish by using Googletrans, a Python library that implements Google Translate API [71]. We examined different machine translation systems and observed that Google produces the best results.

As used in the original form of SVAMP, tagging the numbers with numberX tags according to the order in which they are mentioned provides a generic structure. Since each different number receives a different token in the embedding model, the learning success of the model is negatively affected. However, when the same tokens are included in each problem text thanks to tags (e.g., number0, number1), the attention mechanism tends to pay attention to these tokens as crucial points of the sentence. Therefore, the numbers in each problem text were extracted and stored as a new field, then changed with new tags according to their order. On the other hand, a space has been added before and after all punctuation marks in the sentence so that each punction mark is considered as an embedding token, and all letters are converted into lowercase. The conversion process can be examined in Table 4.2.

Table 4.2. Number tagging in the problem text.

Original Data Sample	**Data Sample After Tagging**
question: Lana'nın en sevdiği grup, biletlerin her birinin 6 dolar olduğu bir konser veriyordu. Lana kendisi ve arkadaşları için 8 bilet ve başka biri gitmek isterse diye ekstra 2 bilet aldı. Toplam kaç dolar harcadı?	*question*: lana ' nın en sevdiği grup , biletlerin her birinin number0 dolar olduğu bir konser veriyordu . lana kendisi ve arkadaşları için number1 bilet ve başka biri gitmek isterse diye ekstra number2 bilet aldı . toplam kaç dolar harcadı ? *numbers*: [6, 8, 2]

In order to generate the equations more consistently and accurately, a common template structure is applied to the equations in the dataset. This structure is already used in SVAMP. Conversion to the template structure has been made for MAWPS, ASDiv-A, and then MathQA. The original versions of the equations are referred as the *infix notation*, where the mathematical operators are between the numbers (operands) in the expression such as (A+B)*(C-D). In our equation template, the operator to be applied to two operands is declared just before these operands. This type of notation

is called the *prefix notation* (e.g. *+AB-CD). The operators shift to the left while the operands remain in their current positions. However, this logic does not work correctly in parenthetical equations. For this reason, the stack data structure is used in the conversion from the infix to the prefix notation. The algorithm employed is as follows:

(i) Create an empty operator stack for the operators, and an empty operand stack for the numbers.
(ii) Determine whether a particular element is an operator, an operand, or a parenthesis.
(iii) If it is an operand, push it into the operand stack.
(iv) If it is a left parenthesis, push it into the operator stack.
(v) If it is a right parenthesis, pop the operator stack, and pop two operands from operand stack. Merge them as a string and push into the operand stack.
(vi) If it is an operator, check the precedence of the current operator and the operator at the top of the operator stack. The precedence of the operators from low to high is as follows: "-+", "*/".
    (a) If the current operator has a higher or equal precedence than the top of the operator stack, push the current operator into the operator stack.
    (b) If the current operator has a lower precedence, hold the current operator, pop the operator stack, and pop two operands from the operand stack. Merge them as a string and push it into the operand stack. Continue this loop until the held operator precedence is greater than the top of the operator stack.
(vii) When the infix equation is completely processed, pop the operator stack, pop two operands from the operand stack, merge them as a string and push it into the operand stack until the operation stack is empty.

4.2.2. MathQA Dataset

The MathQA benchmark dataset consisting of 37200 data was employed as the second dataset. This dataset was adopted because it is one of the most challenging datasets, the amount of data is satisfactory, and it covers a variety of questions

from many aspects. The dataset requires much more preprocessing than the MAWPS, ASDiv-A, and SVAMP datasets, as the questions and solutions are collected from websites. An annotated equation template is used to formulate the absolute equations. However, there are unknowns, constant terms, and complex mathematical operations even in the annotated equations. The numerical answer in the dataset is not always the same as the result of the annotated formula. Despite all the difficulties, many operations were carried out to translate MathQA into Turkish, and it was evaluated in the model training.

Following visual inspections of 37200 data points, the dataset was reduced to 19555 data samples. Physics, geometry, and some of the probability, economics, and interest problems that require knowledge of formulas and equations with many unknowns have been eliminated. Apart from these, the samples that have unknown characters in the problem text and may affect the translation performance have also been removed. Outlier questions with more than 15 tokens in their equations were also deleted.

Several processing operations were carried out on the remaining data before and after the translation. Although there are a great number of operations we performed, we include here the important ones to serve as a guideline for future studies to translate the datasets into different languages. First, some of the pre-translation processes are listed below.

- English abbreviations in the problem texts were written in their long forms. The Googletrans API can not translate abbreviations properly. The dots and spaces of the abbreviations also vary. There are multiple versions of an abbreviation. Some example abbreviations are:
  p.c. → percent, c.i. → compound interest, s.i. → simple interest, p.a. → per annum, h.c.f. → highest common factor, l.c.m. → least common multiple, @ → at, a.p. → arithmetic progression, avg → average, rs → $, no → number, m → meter, m^2 → square meter, cm^3 → cubic centimeter.

- It is critical in our problem that numeric values are expressed in numbers rather than in written form. For this reason, the values such as “one”, “two”, “a hundred”, “million” were translated into numbers.
- Sentences starting with find/solve/calculate keywords but ending with a question mark, and sentences starting with what/how keywords but ending with a dot were edited.
- A caret (^) sign was used to prevent the incorrect translation of the exponential expressions (e.g. “5 power 2” → “5^2”).
- The commas used as the thousands of separators in English are removed. The reason for this removing is that Googletrans API translates the comma, which is the thousands separator, and the period, which is the decimal separator, as a period.
- The decimal expressions that continue as zero are removed (e.g. “52.00” → “52”).
- The mixed numbers in the question were updated with the addition operation due to the errors in the translation (e.g. $3\frac{2}{5} \rightarrow 3 + \frac{2}{5}$).
- The expressions such as “×”, “x”, “÷” were converted to “*” and “/”.
- The different Unicode characters used in place of apostrophes, double quotes, and dashes were adjusted.
- The sequential expressions were shown in written form (e.g. “1st” → “first”, “2^{nd}” → “second”).
- The expressions such as vulgar fraction one half (½) and vulgar fraction three quarters (¾) are written in the division forms.

After translation, the corrections made in the problem text are as follows.

- All letters were converted to lowercase.
- The translation errors in problem texts containing equations were fixed as a result of visual inspections.
- A space was added between the numbers and operators of the equations in the problem text. Thus, the numbers are masked and each operator is treated as a separate token.

Problem text: (18^a)*9^(3a-1)=(2^2)(3^b) için a ve b pozitif tam sayılarsa, a'nın değeri nedir?
Processed text: (number0 ^ a) * number1 ^ (number2 a - number3) = (number4 ^ number5) (number6 ^ b) için a ve b pozitif tam sayılarsa , a ' nın değeri nedir ?

Following that we turned the annotated formulas into the prefix templates as we used in the previous dataset. In the annotated formula, there were constant numbers in the form of *const_X*, for example, const_100 for the percentile problems. We used these constants in their numerical forms. On the other hand, apart from the four basic mathematical operations (add, subtract, divide, and multiply) that we considered, there are some advanced mathematical operations such as *round*, *log*, and *cosine*. Since all operations are displayed in written form rather than signs, these operations are easily extracted, and related data samples are removed from the dataset. We deleted the samples with more than 15 tokens in the equation that reduces the success of the model. It can be said that the formula is generally suitable for the prefix structure. After the written operators were translated into the signs, we obtained our final equations. An example is shown below:

"multiply(divide(80, 160), const_100)" → "* / 80 160 100"

Eventually, we constructed the final version of the dataset in JSON data format, including the "Question", "Equation", "Numbers", and "Answer" fields.

5. METHODOLOGY

In this thesis, the steps of the MWP solving system are summarized below, and detailed in the following sections.

- Process the data as the input.
- Feed the input to the embedding model.
- Encode the embedded question text in the encoder.
- Generate the equation with the Luong attention mechanism in the decoder.
- Evaluate the generated equations with the accuracy and BLEU score metrics.

5.1. Data Processing

Until now, we thought that the first step of the system is the embedding model. However, the first thing to do in a neural model is to read the data and make other adjustments rather than feature engineering. As soon as our bulk data is read in JSON format, it is shuffled, split into the train and test sets, grouped to contain a specified number of data, and each group is called a *batch*. In order not to increase the computation time by doing the same operations each time, we stored the train and test batches in the DataLoader objects belonging to the PyTorch library, and we saved them to the local system. In case of data or batch size change, the shuffling, splitting, and grouping procedures are initiated, and the DataLoader is updated.

Suppose that each space-separated word, number, or operator in the problem texts and equations is a token. We recorded each distinct token in the problems to a word vocabulary, and each distinct token in the equations to an equation vocabulary. They are simply mapping files. Both vocabularies are used to create the tensor arrays for the embedding. The word vocabulary contains the unique identifiers (ID) corresponding to each token in the questions, the tokens corresponding to the relevant unique IDs, the number of times each token occurs, and the total number of tokens in

both train and test sets. In the following steps, the <s> tag is used to indicate the beginnings of the questions, </s> tag implies the endings, and <unk> is the dummy tag. Since these tags can also be considered as tokens, they were added to the word vocabulary. The fields of the word vocabulary are demonstrated in Table 5.1. The idea of the equation vocabulary is the same as that of the word vocabulary, as shown in Table 5.2. It takes into account all operators, constant numbers and numberX tags in the equations, in addition to the <unk>, <s>, and </s> tags, where the <unk> tag is a placeholder for the unknown words if needed. We stored the word and equation vocabularies in the local system for reuse, just as we did for the train and test batches.

Table 5.1. Word vocabulary.

Key	**Value**
words_voc	{“<unk>”: 0, “<s>”: 1, “</s>”: 2, “ali”: 3, “number0”: 4, “şekeri”: 5, “sınıfındaki”: 6, “number1”: 7, “kişi”: 8, “arasında”: 9, ..., “vanilya”: 7249, “tarifleri”: 7250, “gereklidir”: 7251}
word_ids_voc	{0: “<unk>”, 1: “<s>”, 2: “</s>”, 3: “ali”, 4: “number0”, 5: “şekeri”, 6: “sınıfındaki”, 7: “number1”, 8: “kişi”, 9: “arasında”, ..., 7249: “vanilya”, 7250: “tarifleri”, 7251: “gereklidir”}
word_by_count	{“<unk>”: 1, “<s>”: 1, “</s>”: 1, “ali”: 19, “number0”: 4245, “şekeri”: 37, “sınıfındaki”: 4, “number1”: 4113, “kişi”: 222, “arasında”: 26, ..., “vanilya”: 4, “tarifleri”: 1, “gereklidir”: 1}
number_of_distinct_words	7252

Table 5.2. Equation vocabulary.

Key	**Value**
operators_voc	{“<unk>”: 0, “<s>”: 1, “</s>”: 2, “+”: 3, “number0”: 4, “number1”: 5, “*”: 6, “/”: 7, “number2”: 8, “-”: 9, ..., “5”: 27, ‘“10”: 28}
operator_ids_voc	{0: “<unk>”, 1: “<s>”, 2: “</s>”, 3: “+”, 4: “number0”, 5: “number1”, 6: “*”, 7: “/”, 8: “number2”, 9: “-”, ..., 27: “5”, 28: “10”}
operators_by_count	{“<unk>”: 1, “<s>”: 1, “</s>”: 1, “+”: 1916, “number0”: 3991, “number1”: 3820, “*”: 859, “/”: 795, “number2”: 1618, “-”: 1974, ..., “5”: 2, “10”: 1}
number_of_distinct_operators	29

For some embedding models, the tensor arrays with padding were created for each batch. The tokens of each sample in a batch were encoded by its unique ID in the relevant vocabulary. The unique ID of our end of sentence tag </s> is 2 and is appended to the end of each sample. In order for all the samples of the batch to be in the same size, the </s> tag is padded at the end of the samples, taking into account the max-length sample. Thus, for each batch, we formed a tensor array, which has a shape of the max-sample-length and the batch size. An equation batch and the resulting tensor array are given below as an example.

Equation batch (batch size = 8) → [“/ number1 number2”, “+ number0 number1”, “* / number1 number0 number2”, “- number0 number2”, “/ + number0 number1 number0”, “+ number0 number1”, “* number0 number1”, “- number1 number0”]

Encoded equation batch with padding → [[7, 5, 8, 2, 2, 2], [3, 4, 5, 2, 2, 2], [6, 7, 5, 4, 8, 2], [9, 4, 8, 2, 2, 2], [7, 3, 4, 5, 4, 2], [3, 4, 5, 2, 2, 2], [6, 4, 5, 2, 2, 2], [9, 5, 4, 2, 2, 2]]

```
Equation tensor → tensor([[7, 3, 6, 9, 7, 3, 6, 9],
                          [5, 4, 7, 4, 3, 4, 4, 5],
                          [8, 5, 5, 8, 4, 5, 5, 4],
                          [2, 2, 4, 2, 5, 2, 2, 2],
                          [2, 2, 8, 2, 4, 2, 2, 2],
                          [2, 2, 2, 2, 2, 2, 2, 2]])
```

5.2. Embedding Models

In this thesis, we have integrated many different word embedding models and pre-trained language models into our system to make robust analyses and comparisons. The embeddings are created for both questions and equations. However, the question embeddings are processed by the language models for their contextual representations. In contrast, the equation embeddings are simply constructed for tokenization so that they can be fed into the attention decoder.

5.2.1. Equation Embeddings

The equation embedding model has as many tensors as the number of distinct operators in the equation dictionary. The tensors can be thought of as the embedding vectors, and they are frequently used to store and obtain the word embeddings with indices. The size of each embedding vector, *i.e., embedding dimension*, should be appropriately set so that there is no size mismatch when given as input to the decoder. The weights of the equation embeddings are initialized using uniform distribution.

5.2.2. Question Embeddings

One of the most critical steps of the MWP system is to represent the question tokens accurately and comprehend their contexts. Although the implementation of the embedding models is almost identical, some steps may differ, such as their binaries, input forms, and tokenizers.

5.2.2.1. Word2vec, fastText, and GloVe Embeddings. The use of the word2vec, fastText, and GloVe models in the embedding layer is similar. They employ the pre-trained word vectors, accessible as binary or text forms. For these three embedding models, we used the binary forms of the vectors. The pre-trained word2vec and GloVe models were loaded as the keyed vector, a mapping structure between keys and vectors. Each vector is distinguished by its lookup key, which is often a short string token. On the other hand, the fastText model was obtained as a model object over the binary file with its library.

We generated the tensors filled with random numbers from the standard normal distribution with a zero-mean and unit variance for the three models. The tensor size is related to the number of distinct words in the word vocabulary and the input embedding dimension. The embedding models have different input embedding dimensions. For example, 400-dimensional space is suitable for Turkish pre-trained word2vec, and 300-dimensional space is for the fastText and GloVe models. We obtained the vector representations of each distinct token in the word vocabulary by the pre-trained embedding model and wrote to the relevant index in the tensor array. A random tensor is initialized for the token, which does not exist in the word2vec and GloVe models. In contrast, the fastText model provides the word vectors even for unseen words by aggregating the n-grams included in the word.

The embedding layer stores word embeddings of a fixed dictionary and was constructed from given tensor array. The padded question batch was given to the embedding layer, and the output of the embedding model, in other words, the input of the encoder model was generated.

5.2.2.2. BERT, ELECTRA, and ConvBERT Embeddings. Instead of using any binary file to load the BERT, ELECTRA, and ConvBERT pre-trained language models, it is sufficient to give the model ID of the predefined tokenizer to the function in the Python transformers package. For the BERT model, we used a package with an uncased vocabulary of 128000. In the ELECTRA and ConvBERT implementations, we

preferred the uncased packages trained with the Turkish part of the Multilingual C4 corpus and have a vocabulary of 32000.

In contrast to the usage of padded question batches in the word2vec, fastText, and GloVe models discussed in Section 5.2.2.1, different padding and tokenization procedures were utilized for the BERT, ELECTRA, and ConvBERT models. Firstly, the problem text in an unprocessed question batch was given to the BERT tokenizer as input without any operation. As mentioned in Section 3.3.4, certain tokens mark the beginning and end of the text due to the classification and separating operations of the BERT model. Accordingly, the *[CLS]* token was added to the beginning of the tokenized problem text and the *[SEP]* token to the end. After including the *[CLS]* and *[SEP]* tokens, the tokenized texts were padded with the *[PAD]* token so that all sequences have the same length with the longest sequence in the batch. Then, the tokens were converted to the corresponding IDs in the BERT model and the tensor was created for each problem.

We utilized an attention mask to prevent the model from weighting the *[PAD]* token in the sequence. In order to obtain contextualized representations of each token, the attention mask was fed through the BERT model. The processing stages for ELECTRA and ConvBERT are the same as for BERT. The only difference is in the packages where the models are loaded.

5.3. Encoder Model

In order to encode the input sequence and store the information, we implemented a two-layer bidirectional encoder as a part of the seq2seq model. The encoder retrieves the semantic meanings from the problem text and produces a sequence of hidden states. We employed both GRU and LSTM encoders, which have similar architectures.

When initializing the RNN layers of the encoder, the number of expected features is the same as the embedding size used in the embedding layer. The dimensions must

be consistent so that the output of one layer can be the input of another. Depending on the embedding model used, the number of features in the encoder also changes. The size of the hidden units in each layer is crucial since the hidden states in the encoder will be used to calculate the attention which generates the output in the decoder. The number of recurrent layers is another initialization parameter. Building a two-layer encoder model entails creating a *stacked RNN model* which feeds the outputs of the first layer as inputs to the second layer and calculates the final results. On the other hand, a dropout layer was added to the outputs of all RNN layers except the last layer.

As mentioned earlier, the embedding model produces a padded contextualized question tensor per batch. Even though the lengths of the sequences in the padded question tensor are the same, many redundant values are appended to the sequences. Feeding the padded tensor to the RNN layer wastes computing resources and causes output errors for the forward computations. Therefore, before passing the tensor to the RNN layer, we used the *packing padded sequence* operation, which compresses the redundant padded values in order for the RNN model to read data during the training accurately. Before the packing operation, the padded question tensor should be sorted in descending order by the length of its non-padded question tokens. Following these steps, the padded tensor was fed to the RNN layer.

After the RNN layer, the padded question tensor was still compressed. We refilled the compressed tensor with another operation to perform the subsequent calculations, such as adding the backward and forward hidden states of the RNN. A unidirectional RNN can not address the future context. Therefore, we fed the input vectors into two RNN layers in opposite directions and combined their hidden state vectors. The bidirectional RNN makes use of both past and future information. By combining the bidirectional outputs, our encoder model returns both its outputs and hidden states for use in the decoder.

5.4. Decoder Model

The encoder model is a stack of bidirectional RNNs, whereas the decoder is a recurrent sequence generator that combines a unidirectional RNN with an attention mechanism. The attention mechanism has the capacity to learn the alignment between the input and output sequences. Based on this, we implemented the Luong attention mechanism. As in the encoder, we preferred to use GRU and LSTM models as the RNN in the decoder.

We inserted the <s> tag ID at the beginning of the equation tensor to indicate the start of the sequence and appended the </s> tag ID to signify the end of the sequence. It is crucial to feed this information into the equation tensor since the decoder model runs until it predicts the </s> tag, then the equation generation is completed.

The decoder reads the whole source sequence using the last hidden state and cell state of the encoder and estimates the following word in the target sequence. During training, both the question tensor from the encoder and its equation tensor were inserted into the decoder. In the test case, we only give the question and then expect the equation to be produced. We ran the decoder model at the first time step with the equation tensor and the internal hidden states of the encoder. The output is a token from the equation vocabulary with the highest probability at time step t. We also used this token as the input for the next time step $t+1$, and updated the internal hidden states with that of the decoder. We continue iterating using the hidden state and output of the previous time step until estimating the </s> tag.

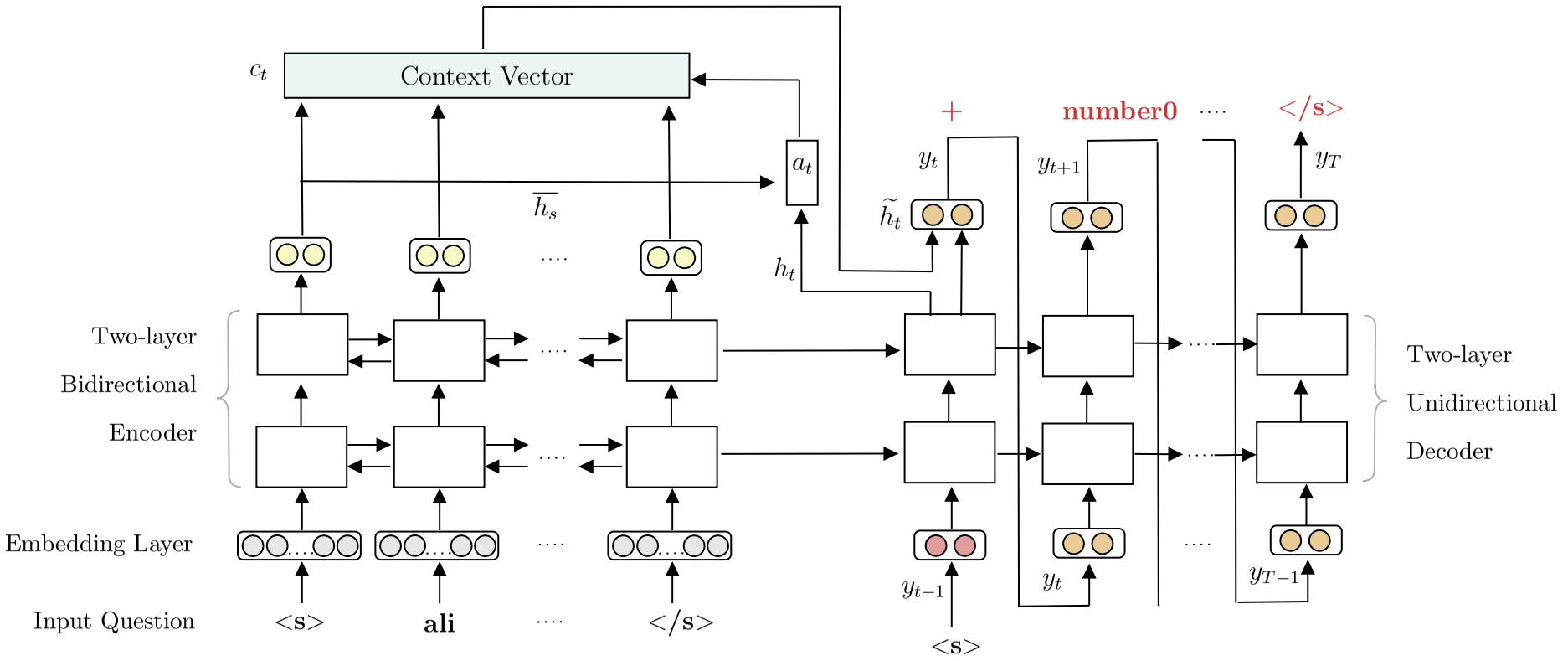
The encoder produces the hidden states in the source sequence at each time step. In the meantime, the decoder releases the hidden state for each step in the target sequence. We calculated an alignment score depending on which word in the source sequence is aligned with the equation token in the target sequence using the score function, as examined in Equation 3.20. The alignment score was determined by combining all the source hidden states (encoder) with the current target hidden

state (decoder). We derived the general alignment score by using the attention weight matrix. The alignment score was then normalized using the softmax function to extract the attention weights.

It is a known fact that both the question sequences as input and the equations as output are not of fixed length. We calculated an intermediate fixed-size context vector to learn about the variable length sequences and relate them to the variable length outputs. On the other hand, the attention mechanism relies on the most relevant parts of the input sequence rather than the complete sequence. Instead of constructing a single context vector from the last hidden state of the encoder, the attention constructs shortcuts between the whole sequence and the context vector. The context vector weights are adjustable for each output. As a result, the context vector establishes an alignment between the source and target sequences.

The context vector was obtained by multiplying the alignment weight vector with the weighted average of all the source hidden states. We performed the unidirectional RNN to produce the tokens of the equation sequence in a one-by-one manner with the context vector. We concatenated the decoder and weighted context vector in the given dimension and added them to the ReLU activation function. A softmax activation function was utilized for the next token prediction over all the tokens in the equation vocabulary.

Figure 5.1 shows the general architecture of our MWP system.



$\overline{h_s}$: *All source hidden states*
h_t : *Current target hidden state*
$\widetilde{h}_t$: *Attention vector*
c_t : *Context vector*
a_t : *Alignment weight vector*
y_t : *Decoder output*

Figure 5.1. Overall architecture

5.5. Evaluation Metrics

The model was evaluated with two different methods, which are explained in detail below.

- *BLEU-4 Score*: Although the equations predicted by our model are not exactly the same as the reference equations in the dataset, it is also a measurable result that several terms overlap with the reference equations. Therefore, instead of just calculating the output accuracy, a BLEU-4 score metric was used, which calculates a score for up to 4-grams using uniform weights. As the BLEU score gets closer to 1, the similarity of the candidate and reference equations increases. The closer to 0, there is no n-gram overlap in any order of n-grams. A smoothing function was applied to handle this severe behavior when no n-gram overlaps are identified. We obtained the BLEU score for each predicted equation and its corresponding reference equation in the batch. We summed up all the BLEU scores in the batch and divided them by the total number of samples in that batch. We averaged the scores of the batches across the entire test dataset. We computed the sentence-level BLEU with a single reference sentence.
- *Accuracy*: It is calculated by counting the cases where the equation produced by the model and the corresponding reference equation are precisely the same. The number of exactly matching samples in the batch is divided by the total number of samples in the batch. By taking the average among the batches, the accuracy across the dataset is found.

6. EXPERIMENTS AND RESULTS

6.1. Dataset Statistics

We split each dataset to use 80% of it for training and the remaining 20% for testing. The number of samples in the train and test sets are shown in Table 6.1.

Table 6.1. Number of samples in train and test sets.

Dataset	**# Samples in Train Set**	**# Samples in Test Set**
Combined	3301	862
MathQA	15651	3904

Figure 6.1 is a histogram plot and shows the distribution of the total number of words in the problem texts based on the train and test sets from both datasets. Although the MathQA set is shuffled before dividing it into train and test sets, there are different distributions between train and test sets according to the length of the problem texts. One reason is that the dataset has a huge number of samples. On the other hand, the train and test sets of the combined data set have a more consistent distribution.

Figure 6.2 visualizes how many equations have a given length in the datasets. In other words, it shows the distribution of the equations in the dataset according to the number of tokens they have. Especially in the MathQA dataset, there were equations with too many tokens that could not be learned and solved. In order to purify the data from this situation, which negatively affects the success of the model, we deleted the samples with more than 15 tokens in the equation. The positive effect of this process on the GRU encoder-decoder model can be seen in Table 6.2.

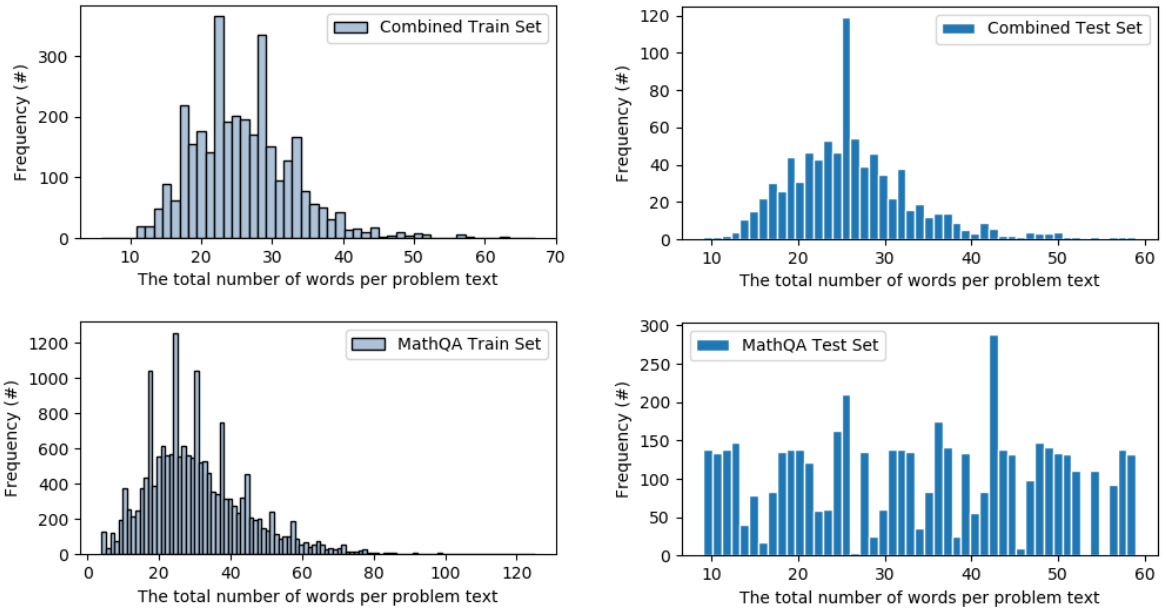


Figure 6.1. Distribution of word counts in the problem texts according to the datasets.

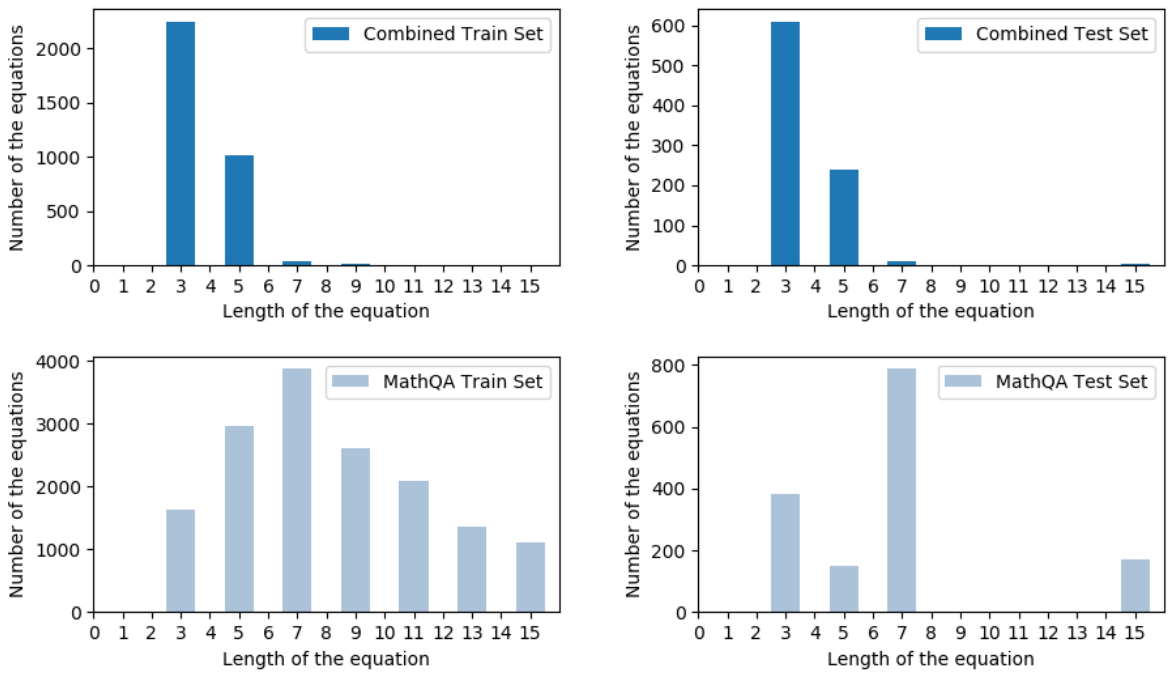


Figure 6.2. Total number of tokens in the equations according to the datasets.

Table 6.2. Effect of removing long equations from the MathQA dataset.

Max. Token Length in the Equation	**Accuracy (%)**	**BLEU (%)**
209	16.10	22.46
15	18.69	32.32

We made simple random sampling from both datasets to measure the success of the GoogleTrans API, which we use to translate the problem texts from English to Turkish. We randomly selected the problem texts from both datasets with a sample size of 20. We evaluated the translation results of the GoogleTrans API with the BLEU score, assuming the versions of the problem texts translated by us without using the API are true. The translation quality of GoogleTrans API can be seen in Table 6.3.

Table 6.3. GoogleTrans API tranlation quality.

Dataset	**BLEU Score(%)**
Combined	91.46
MathQA	94.89

6.2. Word Embedding and Pre-Trained Language Models Results

As previously mentioned, we generated embedding vectors for our problem texts using word2vec, BERT, GloVe, fastText, ConvBERT, and ELECTRA models and fed them into the neural model. The embedding models are critical for exploring the syntactic and semantic relationships between significant contexts in the problem texts and equations. As a result, we conducted an experimental analysis in which we tested several embedding models.

In word embeddings models, dimensionality, i.e., input embedding size, specifies the total number of encoded features. In other words, it refers to the size of word vectors and is distinct from the size of the vocabulary, which is the number of words

for which we preserve. Table 6.4 presents the dimensions of the word vectors used in the embedding models.

Table 6.4. Dimensionality of the word vectors.

Embedding Model	**Dimension of the Word Vector**
Word2vec	400
GloVe	300
fastText	300
BERT	768
ConvBERT	768
ELECTRA	256

For example, if a word in our problem text is an unseen word for the word2vec and GloVe models, we generated a random embedding vector from the standard normal distribution with a zero-mean and unit variance. However, the fastText model can vectorize an unseen word into its n-grams. The results of these two approaches are as in Table 6.5 when all parameters are the same except for the embedding models and their dimensional spaces.

Table 6.5. Test set performance of the word2vec, GloVe, and fastText models.

Model	**Accuracy (%)**	**BLEU (%)**
Word2vec	65.30	62.26
GloVe	60.79	63.39
fastText	62.88	64.50

In the BERT, ELECTRA, and ConvBERT models, we employed an attention mask to prevent the model from weighting the padding token in the sequence. Comparing these three models, we obtained the results in Table 6.6. The ConvBERT can be considered as an enhancement of the ELECTRA. It proposes an orthogonal approach

to augment the attention mechanism with a small parameter size and a faster training. In general, the BERT outperforms all other embedding models.

Table 6.6. Test set performance of the BERT, ELECTRA, and ConvBERT models.

Model	**Accuracy (%)**	**BLEU (%)**
BERT	**71.69**	**72.84**
ELECTRA	63.23	64.26
ConvBERT	69.84	71.55

6.3. Seq2seq Model Results

We constructed the seq2seq models with the attention mechanism based on the Luong's approach to solve MWPs. One of the main reasons to use the seq2seq model is that its input and output length are not fixed and can be of very different sizes, unlike other RNN models. Our seq2seq models consist of a bidirectional RNN encoder and an unidirectional RNN decoder. The datasets were one of the most crucial parts influencing the success of the our model during the tests. Since the MathQA dataset contains problems of various types and difficulties, it achieved much lower success than the combination of other MAWPS, ASDiv-A, and SVAMP datasets.

We first created the seq2seq model with the GRU encoder and GRU decoder and tested it on the combined dataset. The GRU seq2seq model yielded good results with the BERT embedding model. During the tests with different hyperparameters, we observed the direct effect of the number of layers and hidden units on the model success. Since there is only one layer, no dropout is used at the end of the GRU model.

We also applied the GRU model to our Turkish MathQA dataset. The results are not very satisfactory as the MathQA dataset is challenging and does not have a uniform format and problem distribution as the combined dataset. There are cases in the dataset where the equation for the problem is incorrect, or the answer when the

Table 6.7. GRU seq2seq model results for the combined dataset.

	BERT		**ConvBERT**		**Word2vec**		**GloVe**	
GRU Model	**Acc.**	**BLEU**	**Acc.**	**BLEU**	**Acc.**	**BLEU**	**Acc.**	**BLEU**
[1 layer x 128 hidden units]	67.05	70.20	68.45	69.63	56.96	61.26	54.75	59.22
[1 layer x 256 hidden units]	67.40	69.03	68.33	69.16	59.28	62.12	58.93	63.03
[2 layers x 128 hidden units]	67.87	70.60	68.56	69.24	60.09	62.72	59.51	62.90
[2 layers x 256 hidden units]	**71.69**	**72.84**	69.84	71.55	62.30	65.26	60.79	63.39

equation is solved does not match the labeled answer. All these negatively affected the success of the model.

Our second seq2seq model consists of the LSTM encoder-decoder. When the model was tested on the combined dataset with the parameters that allowed success in the GRU, an accuracy of 70.19% and a BLEU score of 71.82% were achieved.

Experiments on our second dataset MathQA did not produce the desired results. During the tests, the GRU model achieved the highest accuracy of 18.69% and BLEU score of 32.32%. In the meantime, the LSTM model obtained the highest accuracy of 18.29% and BLEU score of 32.20%. We see the reason for the failure as the lack of pattern in the questions, the wide variety of question types, and the existence of the equations unrelated to the questions.

6.4. Comparison with Other MWP Studies

Since there are no Turkish MWP studies with the neural models in the literature, we have compiled the results of the studies conducted with the English versions of our datasets. Amini et al. [70] measure the performance of the English MathQA dataset by using a sequence-to-program model, which provides an additional layer of supervision to limit the impact of statistical bias in the dataset. They performed

Table 6.8. LSTM seq2seq model results for the combined dataset.

BERT	Accuracy	**70.19 (%)**
	BLEU	**71.82 (%)**
ConvBERT	Accuracy	68.33 (%)
	BLEU	70.79 (%)
ELECTRA	Accuracy	63.69 (%)
	BLEU	66.65 (%)
Word2vec	Accuracy	60.79 (%)
	BLEU	65.00 (%)
GloVe	Accuracy	60.56 (%)
	BLEU	65.29 (%)
fastText	Accuracy	59.28 (%)
	BLEU	60.92 (%)

a test accuracy of 51.9%. Patel et al. [68] propose a seq2seq model provided with the RoBERTa pre-trained embeddings. Using the MAWPS and ASDiv-A datasets for training and the SVAMP dataset for testing, they achieved 40.3% accuracy. Since SVAMP is a more challenging dataset than the other two, utilizing it directly in the test decreased accuracy. We combined and then shuffled the three datasets to use more homogeneously. Lan et al. [72] implemented a seq2seq LSTM encoder-decoder model and made separate analyzes on the MAWPS, ASDiv-A, and SVAMP datasets. Through their experiments, they found that the MAWPS has an accuracy of 79.7%, ASDiv-A has 55.5%, and SVAMP has 24.2%. Table 6.9 summarizes the comparison results.

6.5. Hyperparameters and Implementation Details

During the experiments, we constructed the model with many different hyper-parameters. For the Turkish MWP solving system, we provided our most successful

Table 6.9. Comparison with English MWP studies.

Study	**Model**	**Dataset**	**Test Accuracy**
Amini et al. [70]	Seq2prog	English MathQA	51.9%
Lan et al. [72]	Seq2seq	English MAWPS	79.7%
Lan et al. [72]	Seq2seq	English ASDiv-A	55.5%
Lan et al. [72]	Seq2seq	English SVAMP	24.2%
Patel et al. [68]	Seq2seq	English MAWPS and ASDiv-A for training, SVAMP for testing	40.3%
Our study	Seq2seq	Turkish MathQA	18.7%
Our study	Seq2seq	Combined Turkish MAWPS, ASDiv-A, and SVAMP	71.7%

results with the hyperparameters given in Table 6.10. Our system has a BERT embedding model with 768 units, a two-layer bidirectional GRU with 256 hidden units as the encoder, a two-layer unidirectional GRU with 256 hidden units as the decoder, an Adam optimizer with a learning rate of 2e-4 for the word embeddings and a learning rate of 8e-6 for the equations embeddings. We applied standard dropout during training after each RNN layers with a probability of 0.1. We trained the model with a batch size of 8 over 50 epochs. To avoid exploding gradients, we used gradient clipping technique by norm. We calculated a negative log likelihood loss over the decoder outputs. We added the softmax activation function to the output layer of the decoder.

We subscribed to a high-performance computing cluster owned by Boğaziçi University TETAM for training and testing the model. We carried out our developments by using the PyTorch library in Python.

Table 6.10. Hyperparameters of the model with the highest accuracy.

Hyperparameter	**Value**
Dataset	MAWPS + ASDiv-A + SVAMP
Model	GRU encoder + GRU decoder
Epoch	50
Number of layers	2
Batch size	8
Embeddin hidden size	768
Hidden size	256
Dropout	0.1
Uniform distribution bound	0.05
Learning rate	2e-4
Embedding learning rate	8e-6
Embedding dimension for equations	16
Max norm of the gradients	1

7. CONCLUSION AND FUTURE WORK

In the literature, solving math word problems in English plays a dominant role. Although there are a few Turkish studies carried out by creating problem templates with rule-based methods, there are no studies conducted with neural models. To remedy this shortcoming, in this thesis, we introduced a sequence-to-sequence neural model with the attention mechanism and aimed to contribute to machine translation systems by using our model, which solves a given written mathematical problem in Turkish by creating an equation.

We contributed to the literature by publishing new Turkish math word problem solving benchmark datasets. We created our first novel dataset by combining the MAWPS, ASDiv-A, and SVAMP datasets in English and translating them into Turkish. We generated our second dataset by translating another English dataset, MathQA. We provided experimental variation using the word2vec, GloVe, BERT, fastText, ConvBERT, and ELECTRA embedding models. On the other hand, we also employed different RNN models, which are GRU and LSTM, in the encoder and decoder to make comprehensive comparisons.

We achieved our best performance with a test accuracy of 71.69% and a BLEU score of 72.84% on the combined dataset using the BERT embedding model and GRU encoder-decoder model. Despite the relatively low accuracies in the second dataset, we gained a corpus that could be worked on and achieved much better results thanks to a large amount of samples.

A promising future work would be exploring techniques to improve our results in the MathQA dataset, implement different neural models such as transformers, and integrate Bahdanau's attention mechanism. We also plan to provide the commonsense knowledge we need while solving problems through semantic networks. Integration with GPT-3, a state-of-the-art language model, is one of our future goals.

REFERENCES

1. Bobrow, D., "Natural Language Input for a Computer Problem Solving System", *Massachusetts Institute of Technology*, 1964.

2. Mukherjee, A. and U. Garain, "A Review of Methods for Automatic Understanding of Natural Language Mathematical Problems", *Artificial Intelligence Review*, Vol. 29, pp. 93–122, 2008.

3. Bakman, Y., "Robust Understanding of Word Problems with Extraneous Information", *arXiv*, 2007.

4. Liguda, C. and T. Pfeiffer, "Modeling Math Word Problems with Augmented Semantic Networks", *Proceedings of the International Conference on Applications of Natural Language Processing to Information Systems*, Vol. 7337, pp. 247–252, 2012.

5. Yuhui, M., Z. Ying, C. Guangzuo, R. Yun and H. Ronghuai, "Frame-Based Calculus of Solving Arithmetic Multi-Step Addition and Subtraction Word Problems", *Second International Workshop on Education Technology and Computer Science*, Vol. 2, 2010.

6. Roy, S., S. Upadhyay and D. Roth, "Equation Parsing: Mapping Sentences to Grounded Equations", *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 1088–1097, 2016.

7. Dries, A., A. Kimmig, J. Davis, V. Belle and L. De Raedt, "Solving Probability Problems in Natural Language", *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pp. 3981–3987, 2017.

8. Roy, S. and D. Roth, "Mapping to Declarative Knowledge for Word Problem Solving", *Transactions of the Association for Computational Linguistic*, Vol. 6, 2018.

9. Say, C., H. L. Akın and S. Özsoy, "A Program which Solves Arithmetic Problems in Turkish", *Proceedings of the Ninth International Symposium on Computer and Information Sciences (ISCIS-IX)*, pp. 550–557, 1994.

10. Say, C., "Understanding Arithmetic Problems in Turkish", *International Journal of Pattern Recognition and Artificial Intelligence*, Vol. 15(2), pp. 359–374, 2001.

11. Çakıroğlu, Ü., "Can Computer Understand and Solve Turkish Arithmetic Problems?", *World Applied Sciences Journal*, Vol. 2, pp. 196–202, 2009.

12. Eken, S., E. Ekinci and A. Sayar, "Understanding and Solving Turkish Arithmetic Problems via XML Keywords", *Düzce Üniversitesi Bilim ve Teknoloji Dergisi*, Vol. 2, pp. 48–55, 2014.

13. Sutskever, I., O. Vinyals and Q. V. Le, "Sequence to Sequence Learning with Neural Networks", *Proceedings of the 27th International Conference on Neural Information Processing Systems*, Vol. 2, pp. 3104–3112, 2014.

14. Brownlee, J., *What Are Word Embeddings for Text?*, 2017, `https://machinelearningmastery.com/what-are-word-embeddings/`.

15. Hosseini, M. J., H. Hajishirzi, O. Etzioni and N. Kushman, "Learning to Solve Arithmetic Word Problems with Verb Categorization", *Proceedings of the Conference on Empirical Methods in Natural Language Processing EMNLP*, pp. 523–533, 2014.

16. Kushman, N., L. Zettlemoyer, R. Barzilay and Y. Artzi, "Learning to Automatically Solve Algebra Word Problems", *Association for Computational Linguistics*, pp. 271–281, 2014.

17. Upadhyay, S., M.-W. Chang, K.-W. Chang and W.-T. Yih, "Learning from Explicit and Implicit Supervision Jointly For Algebra Word Problems", *Proceedings of the Conference on Empirical Methods in Natural Language Processing EMNLP*, pp.

297–306, 2016.

18. Mitra, A. and C. Baral, "Learning To Use Formulas To Solve Simple Arithmetic Problems", *Association for Computational Linguistics*, Vol. 1, pp. 2144–2153, 2016.

19. Liang, C.-C., K.-Y. Hsu, C.-T. Huang, C.-M. Li, S.-Y. Miao and K.-Y. Su, "A Tag-based English Math Word Problem Solver with Understanding, Reasoning and Explanation", *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pp. 67–71, 2016.

20. Zhou, L., S. Dai and L. Chen, "Learn to Solve Algebra Word Problems Using Quadratic Programming", *Proceedings of the Conference on Empirical Methods in Natural Language Processing EMNLP*, pp. 817–822, 2015.

21. Roy, S. and D. Roth, "Unit Dependency Graph and its Application to Arithmetic Word Problem Solving", *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, pp. 3082–3088, 2017.

22. Zhang, D., L. Wang, N. Xu, B. T. Dai and H. Shen, "The Gap of Semantic Parsing: A Survey on Automatic Math Word Problem Solvers", *arXiv*, 2018.

23. Wang, Y., X. Liu and S. Shi, "Deep Neural Solver for Math Word Problems", *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 845–854, 2017.

24. Robaidek, B., R. Koncel-Kedziorski and H. Hajishirzi, "Data-Driven Methods for Solving Algebra Word Problems", *ArXiv*, 2018.

25. Huang, D., J. Liu, C.-Y. Lin and J. Yin, "Neural Math Word Problem Solver with Reinforcement Learning", *Proceedings of the 27th International Conference on Computational Linguistics*, pp. 213–223, 2018.

26. Huang, D., J.-G. Yao, C.-Y. Lin, Q. Zhou and J. Yin, "Using Intermediate Representations to Solve Math Word Problems", *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pp. 419–428, 2018.

27. Mikolov, T., M. Karafiát, L. Burget, J. Cernocký and S. Khudanpur, "Recurrent Neural Network based Language Model", *Proceedings of the 11th Annual Conference of the International Speech Communication Association*, Vol. 2, pp. 1045–1048, 2010.

28. Graves, A. and J. Schmidhuber, "Framewise Phoneme Classification with Bidirectional LSTM Networks", *Proceedings 2005 IEEE International Joint Conference on Neural Networks*, Vol. 4, pp. 2047–2052, 2005.

29. Huang, Z., W. Xu and K. Yu, "Bidirectional LSTM-CRF Models for Sequence Tagging", *arXiv*, 2015.

30. Chung, J., Ç. Gülçehre, K. Cho and Y. Bengio, "Gated Feedback Recurrent Neural Networks", *Proceedings of the 32nd International Conference on International Conference on Machine Learning*, Vol. 37, pp. 2067–2075, 2015.

31. Schmidt, R. M., "Recurrent Neural Networks (RNNs): A Gentle Introduction and Overview", *arXiv*, 2019.

32. Yu, L., "Tackling Sequence to Sequence Mapping Problems with Neural Networks", *arXiv*, 2018.

33. Pascanu, R., C. Gulcehre, K. Cho and Y. Bengio, "How to Construct Deep Recurrent Neural Networks", *arXiv*, 2013.

34. Viswambaran, R. A., G. Chen, B. Xue and M. Nekooei, "Evolving Deep Recurrent Neural Networks Using A New Variable-Length Genetic Algorithm", *IEEE Congress on Evolutionary Computation*, pp. 1–8, 2020.

35. Schuster, M. and K. Paliwal, "Bidirectional Recurrent Neural Networks", *IEEE Transactions on Signal Processing*, Vol. 45(11), pp. 2673–2681, 1997.

36. Williams, R. and J. Peng, "An Efficient Gradient-Based Algorithm for On-Line Training of Recurrent Network Trajectories", *Neural Computation*, Vol. 2(4), pp. 490–501, 1990.

37. Lin, T., W. Horne, P. Tino and C. Giles, "Learning Long-term Dependencies in NARX Recurrent Neural Networks", *IEEE Transactions on Neural Networks*, Vol. 7(6), pp. 1329–1338, 1996.

38. Chung, J., Ç. Gülçehre, K. Cho and Y. Bengio, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling", *arXiv*, 2014.

39. Alla, S., *Introduction to Encoder-Decoder Sequence-to-Sequence Models (Seq2Seq)*, 2021, `https://blog.paperspace.com/introduction-to-seq2seq-models/`.

40. Kostadinov, S., *Understanding Encoder-Decoder Sequence to Sequence Model*, 2019, `https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04af4346`.

41. Cho, K., B. van Merrienboer, D. Bahdanau and Y. Bengio, "On the Properties of Neural Machine Translation: Encoder-Decoder Approaches", *arXiv*, 2014.

42. Loye, G., *Attention Mechanism*, 2019, `https://blog.floydhub.com/attention-mechanism/`.

43. Bahdanau, D., K. Cho and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate", *Computing Research Repository (CoRR)*, 2015.

44. Khandelwal, R., *Attention: Sequence 2 Sequence model with Attention Mechanism*, 2020, `https://towardsdatascience.com/sequence-2-sequence-model-with-attention-mechanism-9e9ca2a613a`.

45. Cristina, S., *The Bahdanau Attention Mechanism*, 2021, `https://machinelearningmastery.com/the-bahdanau-attention-mechanism/`.

46. Luong, M.-T., H. Pham and C. D. Manning, "Effective Approaches to Attention-based Neural Machine Translation", *Association for Computational Linguistics*, pp. 1412–1421, 2015.

47. Cristina, S., *The Luong Attention Mechanism*, 2021, `https://machinelearningmastery.com/the-luong-attention-mechanism/`.

48. Naseem, U., I. Razzak, S. K. Khan and M. Prasad, "A Comprehensive Survey on Word Representation Models: From Classical to State-of-the-Art Word Representation Language Models", *Association for Computing Machinery*, Vol. 20(5), pp. 1–35, 2021.

49. Mikolov, T., K. Chen, G. Corrado and J. Dean, "Efficient Estimation of Word Representations in Vector Space", *ICLR Workshop*, 2013.

50. Wang, B., A. Wang, F. Chen, Y. Wang and C.-C. J. Kuo, "Evaluating word embedding models: methods and experimental results", *APSIPA Transactions on Signal and Information Processing*, Vol. 8(1), 2019.

51. Mikolov, T., I. Sutskever, K. Chen, G. Corrado and J. Dean, "Distributed Representations of Words and Phrases and Their Compositionality", *Advances in Neural Information Processing Systems*, Vol. 26, 2013.

52. Pennington, J., R. Socher and C. Manning, "GloVe: Global Vectors for Word Representation", *Association for Computational Linguistics*, pp. 1532–1543, 2014.

53. Bojanowski, P., E. Grave, A. Joulin and T. Mikolov, "Enriching Word Vectors with Subword Information", *Transactions of the Association for Computational Linguistics*, Vol. 5, 2016.

54. Devlin, J., M.-W. Chang, K. Lee and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", *arXiv*, 2018.

55. Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, "Attention Is All You Need", *arXiv*, 2017.

56. Yeung, A. A., *BERT - Tokenization and Encoding*, 2020, `https://albertauyeung.github.io/2020/06/19/bert-tokenization.html/`.

57. Jiang, Z., W. Yu, D. Zhou, Y. Chen, J. Feng and S. Yan, "ConvBERT: Improving BERT with Span-based Dynamic Convolution", *arXiv*, 2020.

58. Clark, K., M.-T. Luong, Q. V. Le and C. D. Manning, "ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators", *ICLR*, 2020.

59. Sachan, M. and E. Xing, "Learning to Solve Geometry Problems from Natural Language Demonstrations in Textbooks", *Association for Computational Linguistics*, pp. 251–261, 2017.

60. Seo, M., H. Hajishirzi, A. Farhadi, O. Etzioni and C. Malcolm, "Solving Geometry Problems: Combining Text and Diagram Interpretation", *Association for Computational Linguistics*, pp. 1466–1476, 2015.

61. Huang, D., S. Shi, C.-Y. Lin, J. Yin and W.-Y. Ma, "How well do Computers Solve Math Word Problems? Large-Scale Dataset Construction and Evaluation", *Association for Computational Linguistics*, Vol. 1, pp. 887–896, 2016.

62. Wang, H., F. Tian, B. Gao, C. Zhu, J. Bian and T.-Y. Liu, "Solving Verbal Questions in IQ Test by Knowledge-Powered Word Embedding", *Association for Computational Linguistics*, pp. 541–550, 2016.

63. Shi, S., Y. Wang, C.-Y. Lin, X. Liu and Y. Rui, "Automatically Solving Number Word Problems by Semantic Parsing and Reasoning", *Proceedings of the 2015*

Conference on Empirical Methods in Natural Language Processing, pp. 1132–1142, 2015.

64. Upadhyay, S. and M.-W. Chang, "DRAW: A Challenging and Diverse Algebra Word Problem Set", *Microsoft*.

65. Koncel-Kedziorski, R., H. Hajishirzi, A. Sabharwal, O. Etzioni and S. D. Ang, "Parsing Algebraic Word Problems into Equations", *Transactions of the Association for Computational Linguistics*, Vol. 3, pp. 585–597, 2015.

66. Ling, W., D. Yogatama, C. Dyer and P. Blunsom, "Program Induction by Rationale Generation: Learning to Solve and Explain Algebraic Word Problems", *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, Vol. 1, pp. 158–167, 2017.

67. Koncel-Kedziorski, R., S. Roy, A. Amini, N. Kushman and H. Hajishirzi, "MAWPS: A Math Word Problem Repository", *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1152–1157, 2016.

68. Patel, A., S. Bhattamishra and N. Goyal, "Are NLP Models really able to Solve Simple Math Word Problems?", *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2080–2094, 2021.

69. Miao, S.-y., C.-C. Liang and K.-Y. Su, "A Diverse Corpus for Evaluating and Developing English Math Word Problem Solvers", *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 975–984, 2020.

70. Amini, A., S. Gabriel, P. Lin, R. Koncel-Kedziorski, Y. Choi and H. Hajishirzi, "MathQA: Towards Interpretable Math Word Problem Solving with Operation-Based Formalisms", *Proceedings of the 2019 Conference of the North American*

Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 2357–2367, 2019.

71. Han, S., *Googletrans Library*, 2020, `https://pypi.org/project/googletrans/`.

72. Lan, Y., L. Wang, Q. Zhang, Y. Lan, B. T. Dai, Y. Wang, D. Zhang and E.-P. Lim, "MWPToolkit: An Open-source Framework for Deep Learning-based Math Word Problem Solvers", *arXiv*, 2021.

APPENDIX A: SAMPLES FROM MWP DATASETS

Table A.1. Samples from MWP datasets.

Dataset	**Sample**
Alg514	**sQuestion**: In a chemistry class, 5 liters of 4% silver solution must be mixed with a 10% solution to get a 6% solution. How many liters of the 10% solution are needed?, **lEquations**: .01*4*(5)+.01*10*x=.01*6*(5+x), **lSolutions**: 2.5, **Template**: a * m - b * m = b * c - c * d, **iIndex**: 300319, **Alignment**: [{coeff: d, SentenceId: 0, Value: 4.0, TokenId: 8}, {coeff: c, SentenceId: 0, Value: 5.0, TokenId: 5}, {coeff: a, SentenceId: 0, Value: 10.0, TokenId: 17}, {coeff: b, SentenceId: 0, Value: 6.0, TokenId: 23}], **Equiv**: [[0, 17, 10], [1, 5, 10]]
Dolphin1878	**id**: algebra.com.117395, **index**: 1043, **text**: one number is 11 more than another number. Find the two numbers if three times and the larger exceeds four times the smaller number by 4., **sources**: algebra.com.117395, **equations**: [unkn: x, y, equ: x=y+11, equ: 3*x=4*y+4], **ans**: {40;29}, **ans_simple**: [40, 29]

Table A.1. Samples from MWP datasets. (cont.)

Dataset	**Sample**
Dolphin18K	**original_text**: what is 30 divided by half plus 10? 1st correct answer gets 10 points!!!!!?, **text**: what is 30 divided by half plus 10?, **flag**: 0, **ans**: 70, **equations**: unkn: x, equ: x=30/(1/2)+10, **id**: yahoo.answers.20061213041448AAxoy5z
AQuA	**question**: A grocery sells a bag of ice for $1.25, and makes 20% profit. If it sells 500 bags of ice, how much total profit does it make?, **options**: [A)125, B)150, C)225, D)250, E)275], **rationale**: Profit per bag = 1.25 * 0.20 = 0.25, Total profit = 500 * 0.25 = 125. Answer is A., **correct**: A
MathQA	**question**: A train running at the speed of 48 km/hr crosses a pole in 9 seconds. what is the length of the train?, **options**: a) 140 , b) 130 , c) 120 , d) 170 , e) 160, **rationale**: Speed = (48*5/18) m/sec = (40/3) m/sec. length of the train = (speed*time). length of the train = (40/3*9) m = 120 m. answer is c., **correct**: C, **annotated_formula**: multiply(divide(multiply(48, const_1000), const_3600), 9)
MAWPS	**iIndex**: 1, **sQuestion**: Joan found 70.0 seashells on the beach . She gave Sam some of her seashells . She has 27.0 seashells . How many seashells did she give to Sam ?, **lEquations**: X=(70.0-27.0), **lSolutions**: 43.0

Table A.1. Samples from MWP datasets. (cont.)

Dataset	**Sample**
ASDiv-A	**id**: nluds-0064, **grade**: 2, **source**: http://www.k5learning.com, **body**: Gino has 63 popsicle sticks. I have 50 popsicle sticks., **question**: What is the sum of our popsicle sticks?, **solution-type**: addition, **answer**: 113 (popsicle sticks), **formula**: 63+50=113
SVAMP	**question**: julia played tag with number0 kids on monday . she played tag with number1 kids on tuesday . how many more kids did she play with on monday than on tuesday ?, **numbers**: 18 10, **equation**: - number0 number1, **answer**: 8.0, **group_nums**: [1, 2, 3, 4, 5, 6, 12, 13, 14, 28, 29, 30], **type**: Subtraction, **variation_type**: 11, **body**: julia played tag with number0 kids on monday . she played tag with number1 kids on tuesday ., **ques**: how many more kids did she play with on monday than on tuesday ?