Word Problem Solving

Thesis submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy
in
Computer Science and Engineering

by

Pruthwik Mishra
201307577
pruthwik.mishra@research.iiit.ac.in

International Institute of Information Technology
Hyderabad - 500 032, INDIA
December 2023

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "Word Problem Solving" by Pruthwik Mishra, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date	Advisor: Prof. Dipti Misra Sharma

To my family

Acknowledgments

This has been a long, tiring, but a fulfilling journey which is coming to an end. I would like to thank everyone who supported me throughout this journey. It would not have been possible without your help.

First of all, I would thank my advisor Prof. Dipti Misra Sharma who has been very supportive and patient with me in all these years. She has been an amazing guide and person who taught me a lot of things starting from natural language processing concepts to very defining life lessons. I cannot remember a single time when I wanted to discuss something with her and she had not spared time. Thank you madam for this. I thank all the LTRC faculty Dr. Radhika, Dr. Manish, Dr. Parameswari, Prof. Sangal, Dr. Anil, Dr. Chiranjeevi, Prof. Vasudev Varma, Dr. Rajakrishnan, Dr. Rahul for their support in different research areas which I have worked on. I thank all our office and admin staff Praveen, Dhanalaxmi, Laxminarayan Sir, Namratha, Sammaiah Sir, Rambabu Sir, Mahendra sir, Murthy sir, Satish Gatla Sir, Pushpalatha madam, Prathima madam, Kumaraswamy Sir, Srikanth Sir, Prabhakar Sir, Krishna Kishore Sir, Nadeem Sir, Srinivas Rao Sir, Saidulu Sir, and all the house-keeping, mess, and security staff.

I would thank all my friends and research colleagues who have given their invaluable suggestions for my research. This is a going to be a long list. I thank my friends Kunal, Chandan, Vinitha, Anwesha, Ravi, Sai Krishna, Divya Sai, Yashwanth, Ratish, Sushant Bhai, Kishori, Shastri, Nirmal, Sai Ganesh, Mounika. I am grateful that I got the opportunity to work with a group of language experts who have contributed to my progress. I thank all of them: Alpana madam, Anita madam, Preeti madam, Nandini madam, Mithu madam, Kaberi madam, Krithika madam, Sameena madam, Sarita madam, Younus ji, Noman ji, Vaibhavi, Srivani madam, Sarala madam, Shailaja madam, and Avinash sir. I thank Litton for introducing me to the world of word problems and being a part of my initial days in researching this field. I would like to especially thank Arpit, Litton, Maaz, Krishnakant, Vighnesh, Nikhilesh, Ashok, Vandan, Saumitra, Prathyusha, Pranav, Arafat, Ganesh for sharing a great bond of friendship which made this journey more memorable. I thank my fellow students Harshita, Ayush, Sankalp, Anvishka, Kriti, Ketan with whom I got the opportunity to research and mentor them. I want to express my deep gratitude to Vandan with whom I share a great camaraderie culminating

in many research outcomes in terms of tools and research papers. I used his developed tools in many of the works presented in this thesis.

Lastly, I thank the most crucial part of my life, my family. I dedicate this thesis to my mother who has been a pillar of strength for me in difficult times. During all these years, she has always given me courage and taught me perseverance to keep going. It would not have been possible without you, Bou. The second person who has always kept a constant watch on my progress is my elder sister. Although she is a hard task master, she always egged me on to reach my goals. I thank my wife for providing unwavering support to me. I was not present in many of the important phases of her life, but she never complained. I convey my deep gratitude to all these three powerful individuals.

I thank my thesis review panel, Prof. Amba Kulkarni, Prof. Sivaji Bandyopadhyay, and Dr. Manish Shrivastava, for their insightful comments and guidance.

Abstract

Education plays a vital role in shaping up one's life. Education in early childhood includes learning from different activities where counting and other concepts of mathematics act as major building blocks. Mathematics is not just a subject to be taught in schools, but also it has myriad applications in our daily lives such as counting the total number of stationery items, calculating their individual prices, and adding them up for a purchase made in a grocery shop. This kind of real world situations are posited in mathematical word problems which are an essential part of a child's learning that requires natural language understanding (NLU) as well as knowledge of mathematical operations. Mathematical Word Problem Solvers can assist both students and teachers. It is a challenging field and this thesis attempts to provide NLP solutions for math word problems in English and Indian languages.

Our approach for developing word problem solvers follows a pipeline. For a word problem, first, we identify the relevant operands and required operations. In the second step, we form an equation from these identified components. The first two stages can be combined to generate equations at once using neural network based approaches. At the last stage, the equation is solved by a mathematical solver to get the final solution. We focus primarily on the first two stages of this pipeline. We developed solvers using three kinds of approaches: frame based, composition of classifiers based, and end-to-end neural based.

We also shed light on the limitations of the current automatic solvers with respect to the data. We designed different data augmentation techniques to overcome the data scarcity problem. As a part of resource building, we developed word problem datasets and solvers for Indian Languages. In addition to this, we compared different models related to our developed approaches. We empirically show the difference in generation of various equation notation types. For this study, we present the results of equations in infix, postfix, and prefix notations. We also show two natural language processing tasks where components of word problem solvers can be utilized. In the first task, we studied the impact of simple number based pre-processing on the performance of machine translation systems. In the second task, we analyze speech transcript texts to extract equation spans. This kind of text is present mainly in transcripts from mathematical domain. For achieving this, we develop an equation identifier and convertor involving math notations for transcripts. This can make the transcripts easily readable for transcripts which have wide usage of mathematical terms.

Contents

List of Figures

List of Tables

List of Abbreviations

BiGRU	Bidirectional Gated Recurrent Unit
BiLSTM	Bidirectional Long Short Term Memory
CLD	Corpus Lexicon Diversity
CNF	Chomsky Normal Form
EDA	Easy Data Augmentation
GRU	Gated Recurrent Unit
ILs	Indian Languages
LCA	Least Common Ancestor
LSTM	Long Short Term Memory
NLP	Natural Language Processing
SAT	Scholastic Assessment Test
SNI	Significant Number Identification
WPS	Word Problem Solving

Chapter 1

Introduction

Natural Language Understanding (NLU) is the most challenging aspect of Natural Language Processing (NLP). NLU systems are essential for performing a gamut of tasks ranging from simpler tasks, such as designing chatbots based on short and simple instructions, to relatively complex tasks, such as reading comprehension and question answering involving different forms of natural language (NL) inputs. Word Problem Solving (WPS) falls in the category of complex NLU tasks. Solving a word problem involves three subtasks:

- Understanding the problem (NLU)
- Forming equations
- Solving the equations

Our research mainly focuses on the first two subtasks. Word problems come in different flavors that require varying levels of NLU. Some word problems are very easy to solve where cues for the operands and operation to be performed are explicitly given, such as "Find the result when 5 is added to 3." Here, "added" refers to the addition operation and the operands are directly mentioned. But explicit cues are often missing in word problems where it becomes very challenging to interpret the meaning of the sentences. In addition to this, it is difficult to figure out which parts of the word problem text are relevant and which parts are to be ignored. Let us look at a few examples below and analyze these issues which make the NLU task much harder.

Example 1:

- Problem Text: "*At a function held in Bhubaneswar's Kalinga Stadium, Naveen Patnaik gave cheques of Rs 2.5 crore each to the Indian men's hockey team vice-captain Birendra Lakra and defender Amit Rohidas and Rs 50 lakh each to Deep Grace Ekka and Namita Toppo of the women's hockey team.*" [1]

[1] `https://www.hindustantimes.com/cities/others/odisha-cm-gives-cash-awards-to-\olympic-hockey-players-101628683292019.html`

- Question: "*How much money was awarded to the hockey players in total?*"
- Steps required for Solving the problem:
  1. NLU:
     - Identification of two sets of hockey players: a. Men b. Women
     - Identification of explicit operands 2.5 crore, 50 lakhs
     - Identification of implicit operands: total number of players in each set is missing; only the names of the players are mentioned in the problem text
     - Identification of operations: *each* for multiplication, *total* for addition
  2. Equation Generation:
     - *Equation 1:* $Total\ Money\ Awarded\ to\ Players\ from\ Men's\ Team\ =$ $Total\ Players\ from\ Men's\ Team\ *\ Money\ Awarded\ to\ Each\ Man$
     - *Equation 2:* $Total\ Money\ Awarded\ to\ Players\ from\ Women's\ Team\ =$ $Total\ Players\ from\ Women's\ Team\ *\ Money\ Awarded\ to\ Each\ Woman$
     - *Equation 3:* $Total\ Prize\ Money\ =$ $Total\ Money\ Awarded\ to\ Players\ from\ Men's\ Team$ $+\ Total\ Money\ Awarded\ to\ Players\ from\ Women's\ Team$
  3. Solving the Equations: Instantiation of variables and solving using a mathematical solver
     $Total\ Players\ from\ Men's\ Team\ =\ 2,$
     $Money\ Awarded\ to\ Each\ Man\ =\ 2.5\ crore,$
     $Total\ Players\ from\ Women's\ Team\ =\ 2,$
     $Money\ Awarded\ to\ Each\ Woman\ =\ 50\ lakh$

Example 2:

- Problem Text: "*Each medal winner at the Tokyo Olympics will be awarded a cash prize of Rs 1 Cr by the Union government.*"
- Question: "*What is the total prize money given to the Olympic medal winners?*"
- Steps required for Solving the problem:
  1. NLU:
     - Identifying how many medals Indian players won at the Tokyo Olympics: requires world knowledge which is missing from the problem text
     - Identification of explicit operands 1 Cr
     - Identification of operations: *each* for multiplication
  2. Equation Generation:

- *Equation 1: Total Money Awarded to Players* $=$ *Total Number of Olympic Medal Winners* $*$ *Money Awarded to Each Winner*

3. Solving the Equations: Instantiation of variables and solving using a mathematical solver
   $Total\ Number\ of\ Olympic\ Medal\ Winners\ =\ 7\ (from\ world\ knowledge)$,
   $Money\ Awarded\ to\ Each\ Player\ =\ 1\ Cr$

Example 3:

- Problem Text: "*GST collections recover to 1.16 lakh crore in July. The July 2021 collections were 33% higher than a year ago, with GST collected on the import of goods rising 36% and domestic transactions (including import of services) growing by 32%.*" [2]
- Question: "*What were the GST collections in July 2020?*"
- Steps required for Solving the problem:

1. NLU:
   - Identification of GST collections in 2021: 1.16 lakh crore
   - Percentage of growth: 33%
   - Identification of irrelevant quantities
     * *2021* in July 2021
     * *36* in GST collected on the import of goods rising 36%
     * *32* in domestic transactions (including import of services) growing by 32%
   - Introduction of Implicit Constants: 0.01 is introduced when dealing with percentages
   - Identification of operations: *higher* for addition, % for multiplication with 0.01
2. Equation Generation:
   - *Equation 1: Growth in rupees in* 2021 *compared to* 2020 $=$ *GST collections in* 2020 $*$ *growth_percentage* $*$ 0.01
   - *Equation 2: GST collections in* 2021 $=$ *GST collections in* 2020 $+$ *Growth in rupees in* 2021 *compared to* 2020
3. Solving the Equations: Instantiation of variables, conversion of numbers in words into numeric equivalents and solving using a mathematical solver
   GST collections in 2021 $=$ 1.16 *lakh crore* $=$ 1160000000000,
   growth_percentage $=$ 33

[2] https://www.thehindu.com/business/Economy/gst-collections-recover-to-116-lakh-\crore-in-july/article35668958.ece

These examples shed light on various challenges with respect to varying degrees of complexitities in NLU and equation generation while solving arithmetic word problems.

1.1 Motivation

Humans employ domain knowledge and quantitative reasoning for understanding word problems. Imparting this kind of world knowledge is difficult for any automatic system. Along with all these, scarcity of resources is a major bottleneck while designing robust word problem solvers. Since word problems involve NLU, they can be used in several NLP applications, for example, machine translation and equation identification in English transcripts. Word problem solvers can also act as teaching assistants. Frame based solvers can empower the teachers to explain arithmetic word problem solving to students with the help of frames and easily understandable frame slots. A solver should have the capability of solving diverse word problems as well as robustness. In this work, we attempt to address these challenges by developing techniques using frame based, machine learning, and neural models.

1.2 Definition

We will first define a word problem. A word problem consists of a real world narrative consisting of 2 parts: problem text and question text as shown in the example below.

- Word Problem: *Ramesh had 10 pencils. He gave 3 to Suresh. How many pencils does Ramesh have now?*
- Problem Text: *Ramesh had 10 pencils. He gave 3 to Suresh.*
- Question Text: *How many pencils does Ramesh have now?*
- Equation: $x = 10 - 3$
- Solution: 7

The problem text consists of 2-3 sentences and describes a real world scenario involving different entities, quantities, and their associated units. These sentences include different operands and operations to be carried out on them. The question text queries an unknown quantity. So, identification of the relevant operands and operations becomes quintessential in solving word problems. In this thesis, we also develop different representation schemes and show their efficiency in problem solving.

1.3 Approaches for Solving Word Problems

Earlier systems were designed to solve word problems which can only operate on a limited set of inputs. Most of the techniques were schema based and handled only addition-subtraction type of problems. Statistical models using verb categorization, expression trees, linear log models using various features of different word problem types were also explored. In recent years, there has been a reinvigorated interest in solving word problems of different kinds - arithmetic questions, probability questions, SAT questions, science questions, questions from other domains as well as word problems with visual cues using neural networks. Most of the current approaches are neural and they treat word problem solving as a sequence to sequence learning task. The whole problem text is considered as the input to an encoder and the corresponding equation to solve the problem is the desired output to be generated by a decoder. Lately, graph transformer based encoder and tree based decoder has been reported as the state-of-the-art technique.

In this thesis, our primary focus is on solving arithmetic word problems using neural approaches.

1.4 Our Contribution

The main contributions of our thesis can be summarized below:

1. We propose a frame identification based word problem solver for English where each frame is represented in terms of pre-defined slots.

2. We develop a GRU based operator identifier and similarity based operand detection model and generate equations that are composed of these identified components for English.

3. We develop a transformer based operand identification model and a transformer based operator identification model for English.

4. We develop an end-to-end equation generation using memory network based encoder and LSTM based decoder for English.

5. We develop different neural based end-to-end generation models including attention based sequence-to-sequence models, transformer based models from scratch, and fine tuning pre-trained transformer models. All these models are designed to solve word problems in English.

6. We present a word embedding and paraphrase based data augmentation technique to tackle the problem of data sparsity.

7. We develop word problem solvers for Hindi and Telugu using current state-of-the-art pre-trained multilingual transformer models.

8. We create and release benchmark word problem datasets conisting of more than 1000 word problems for English, Hindi, and Telugu.
9. We propose a new evaluation technique for equation accuracy centered on equation or expression equivalence.
10. We also introduce an equation identifier and converter for English transcripts.
11. We develop conversion tools for converting numbers written in words into their numeric equivalents. These tools are developed for English, Hindi, Telugu, Gujarati, and Odia.

Our approaches for word problem solving can be broadly summarized as shown in figure 1.1.

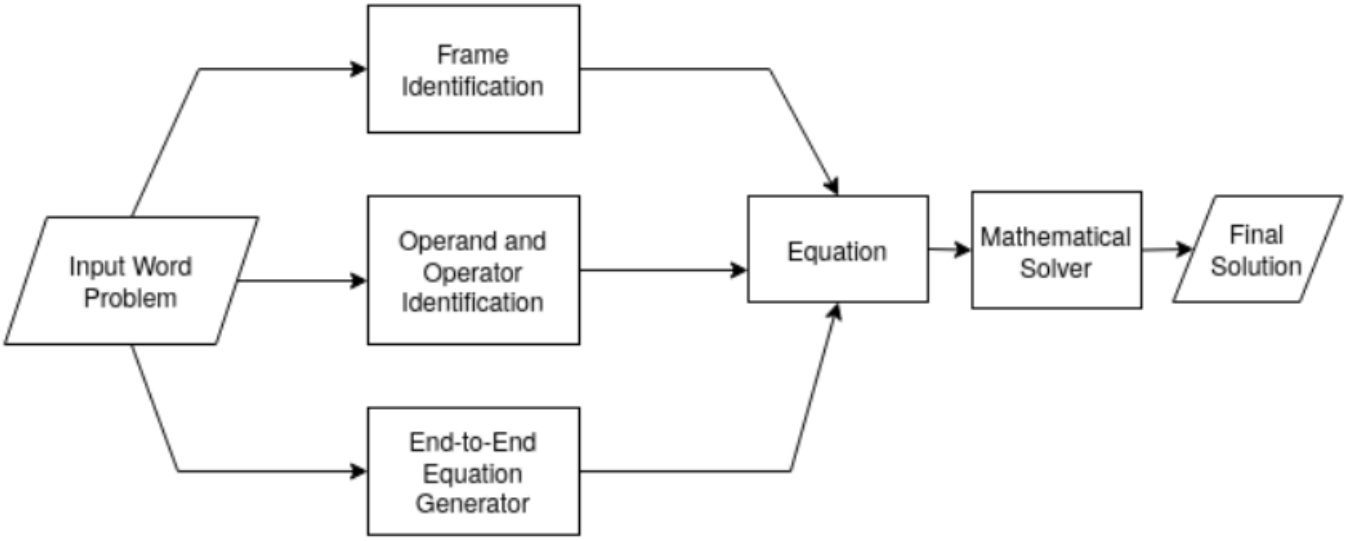

Figure 1.1: Flowchart for Our Approaches

1.5 Organization of Thesis

The rest of the thesis is organized as per the following:

- **Chapter 2 - Related Work:** In this chapter, we briefly outline the previous approaches followed for word problem solving with focus on the recent neural network based approaches.
- **Chapter 3 - Evaluation Metric and Benchmark Datasets:** This chapter sheds light on various evaluation metrics and benchmark datasets available for word problem solving. Different properties of these datasets and the need for diverse and rich datasets for evaluating the robustness of the solvers are also detailed in the chapter.

- **Chapter 4 - Word Problem Solving Using Frame Identification:** This chapter details our initial efforts to build a word problem solver by introducing the concept of frames. This work is inspired from the erstwhile works of schema and verb categories.
- **Chapter 5 - Equation Generation by Composition of Classifiers:** In chapters 5 and 6, we move from single sentence to multi-sentence representations for solving word problems using neural networks. Two approaches are introduced in this chapter. In the first approach, we predict the operator for single operator word problems using a biLSTM/biGRU network. Then, we identify the relevant operands using context similarities between the context around quantities in the problem text and the question text. The final stage generates an equation by composing these identified equation components. In the second approach, we present transformer based operand and operation identification models.
- **Chapter 6 - End-to-End Equation Generation:** This chapter explains our end-to-end model of equation generation using memory network based encoder and LSTM based decoder. Memory network based encoder learns a dense representation of the problem text conditioned on the question for any word problem. This chapter also details other neural approaches including current state-of-the-art based transformer models.
- **Chapter 7 - Data Augmentation Techniques:** We discuss different data augmentation techniques that are used for creating larger datasets containing word problems. In this chapter, we propose a data augmentation technique exploiting the distributional and paraphrase based similarities of content words appearing in the word problems with experimental details of performance improvement. Another augmentation technique through translation is also explored.
- **Chapter 8 - Word Problem Solving in Indian Languages:** Here, we present the current state of word problem solving in Indian Languages. In this chapter, datasets are created containing word problems in Indian languages and solvers for them are developed. Additionaly, we present different data augmentation techniques for Indian languages and empirically show their efficacies.
- **Chapter 9 - Use of WPS Components in NLP Applications:** In this chapter, we show how different components of WPS can be utilized in two different NLP tasks. The first task tackles the problem of incorrect machine translation outputs of numbers written in words. The second task deals with the identification and conversion of mathematical terms in English transcripts where they are represented in terms of plain words. This will make transcripts from this domain more readable.
- **Chapter 11 - Summary, Conclusion, and Future Work:** In this chapter, we summarize our work and provide pointers for future work.

Chapter 2

Word Problem Solving Approaches

Attempts to automatically solve arithmetic word problems started as early as the 1960s. Over the years, several rule-based and statistical word problem solvers have been the go-to models for the task with carefully crafted rules or hand curated features. Recently, with the advancement of deep learning, current systems are moving towards neural models with better text representations across larger texts achieving improved performance on several benchmark datasets. In this chapter, we present an overview of different strategies followed for word problem solving.

2.1 Early Systems

Most of the early systems were rule based that transformed a word problem into an intermediate representation and attempted to solve them through a set of predefined rules or knowledge schemata. *STUDENT* [1] was the first system which could solve arithmetic problems posed in natural language. As an initial task, the word problem was decomposed into simple kernel sentences or propositions. The kernel sentences thus generated had a direct mapping to the information store consisting of simple conversion rules and mathematical formulae. Transformations of English words or phrases into variables, substitutors, and operators were critical to generate the equations. The identification of variables and co-referents were based on simple pattern matching. The same variables were detected by the same phrases across sentences. Any variation to the patterns would make the question unsolvable. Inter sentential dependencies like pronouns were not handled by STUDENT.

WORDPRO [2] was the first system to capture word problems through a form of schemata. Each schema consisted of 3 slots - object, quantity, and specification. 3 higher level schemata *Change* , *Combine*, and *Compare* were defined for word problems involving addition and subtraction problems. The working of the system was controlled by an ordered set of production rules. The firing of specific rules were based on the keywords related to that particular problem type. Even though WORDPRO was tested on a very limited dataset containing 14 word prob-

lem types, it opened up a direction of research for the cognitive modeling of problem solving procedures using schemata. **ARITHPRO** [3] was such an attempt to increase the coverage of problem types and number of rules to include better solving strategies.

Bakman [4] proposed a basic arithmetic word problem solver called *ROBUST* that could solve multiplication and division problems additionally as well as operate on variations in natural language input unlike its predecessors. *ROBUST* also introduced the notion of relevancy on quantities appearing in a problem text. *ROBUST* used 8 different change schemata for differentiating between situations involving changes in place, ownership, creation of new objects, and termination of existing objects. *ROBUST* introduced the concept of "formula", which represented a generalized description of a situation related to a schema. For *Transfer-In-Place* schema, the change formula is as follows:

- There was an initial number of objects in a place.
- An additional number of objects was transferred into this place.
- There is a final number of objects in the place.

Sundaram et al. (2015) [5] used the schemata proposed by Bakman to solve problems available in a benchmark dataset AI2 [6] and achieved improved performance. The major limitation of all these systems was their inability to deal with problems that required world knowledge and common sense.

2.2 Statistical Approaches

Statistical approaches [6, 7, 8, 9, 10, 11, 12, 13, 14] extract problem specific features for the representation of the input word problems. We briefly discuss some widely used statistical approaches.

2.2.1 Template Based Solver

KAZB [7] learned the alignment of variables and numbers from the problem text to generate a system of linear equations. As a part of this work, they released a set of 514 word problems, their corresponding solutions, and sets of equations. The proposed system could solve 70% of the questions. They modeled a joint log linear distribution over the full set of equations and the alignments learned between the components of the equations and the problem text. First, a template was chosen for the overall structure of the equation system. Then, in the second step, the instantiation of the template was done by filling the slots from the numbers and nouns in the text. Both the steps constituted a derivation. The parameters of the suggested model were estimated by maximizing the conditional log-likelihood of the data by marginalizing over all valid derivations. L-BFGS [15] method was used for optimizing the parameters. Most

of the features used for modeling were linguistic such as dependency tree, lemma, phrases, unigrams, bigrams, and others dealt with type of the expected answer i.e whether the solution was positive or an integer. Computing the answer required summing over all templates and all possible alignments which made the search space exponential. So during inference, this was approximated employing a beam search. The authors showed that the system's performance was directly proportional to the increase in system templates' frequency. The authors also acknowledged that external knowledge bases may be required to understand the concepts of decrease, profit, loss, and other semantic concepts. For equation template frequencies higher than 20, the system was able to solve 87% of the word problems. Zhou et al. [8] proposed an improved version of KAZB. The model only mapped the numbers in the word problem into the number slots which significantly reduced the search space and made training easier in terms of both time and space complexities. The proposed system registered an improvement of 10% from the Kushman's system by solving 79.7% of the total problems. A robust log-linear model was designed which maximized the margin between correct and incorrect assignments of slots inside the templates. This resulted in a quadratic programming problem. Upadhyay et al. [9] showed that many systems lacked the reasoning ability on how the equations were constructed. The authors shared a dataset containing derivations of **2200** arithmetic word problems in addition to the equations and solutions. Each derivation was composed of an equation template and its corresponding alignments of slots with the problem text. They showed improvement in performance when derivations were included in training the model.

2.2.2 Verb Categorization Based Solver

ARIS [6] relied on the concept of verb categorization to form equations and solve word problems. They categorized each verb appearing in a sentence into one of 7 predefined classes. They represented every sentence as a set of containers, entities, quantities, attributes, and relations. Each word problem represented a partial state of the world. Quantities got updated or created within the containers with the progression of state. The question queried about a quantity in a particular state. ARIS performed the grounding of the required information mentioned above by using different tools under stanford coreNLP suite [16]. The verbs were categorized using an SVM classifier where the features included wordnet [17] features, dependency level relations between a verb, and other words in a sentence, similarity scores between a verb and a set of seed words. The updates of quantities between states are completed by matching the containers and entities between two successive states. The irrelevant quantities can be easily found by looking at the non-matching containers. They released a public benchmark dataset for arithmetic word problems named AI2. This dataset consisted of only addition and subtraction problems. Most of the problems had 2 operands and 1 operation. The system could solve 77.7% of the problems. The errors were attributed to the errors made by the external tools such as

dependency parsers, coreference resolvers, irrelevant information, lack of world knowledge, and missing explicit entities.

2.2.3 Expression Tree Based Solver

Roy et al. (2015) [10] used the concept of expression trees to represent and evaluate arithmetic expressions without the need of additional annotations or predefined templates. The authors uniquely decomposed any arithmetic problem into multiple classification problems, then composed an expression tree through a constrained inference framework. They also introduced the concept of quantity schemas for better extraction of features to be used in the classification tasks. Quantity schema contained all the related information about a quantity. The associated verb, subject, tokens inside the noun phrase containing the quantity known as the 'unit', related noun phrases, whether the quantity referred to rate were constituents of a quantity schema. Two classification tasks were performed: Relevance Classifier for quantities and LCA (Least Common Ancestor) Classifier for identifying the operation between a pair of quantities. Each expression tree was scored based on the confidence scores from the above classifiers. The authors employed beam search strategy for finding the highest scoring constraint satisfying expression as the number of possible expression trees grew exponentially with increase in number of quantities in the text. They showed improvements on all benchmark datasets. Most of their errors occurred as a result of unknown verbs and faulty quantity schema extraction. The authors [11] extented this work by introducing unit dependency graphs. They trained classifiers to identify rate related quantities and compatibility between different units. Roy et al. (2016) [12] introduced the concept of "Equation Parsing". Given a sentence, the equation parser identified the noun phrases representing the variables. It generated a projective equation expressing the relationship between the variables inside a sentence. It also used a high precision lexicon of mathematical expressions and a series of structured classifiers to generate 70% correct equations.

2.3 Neural Approaches

This section focuses on the neural approaches used for word problem solving. Recent literature surveys suggest that neural approaches have transformed this field of research. We briefly discuss some of the seminal works in this area.

2.3.1 Deep Neural Solver

Wang et al. [18] presented a deep neural solver which directly translated math word problems into equation templates using Recurrent Neural Network (RNN) without applying any cumbersome hand-creafted feature engineering techniques. They also designed a hybrid system

combining the mentioned RNN model and similarity-based retrieval model to record an overall improvement in solving word problems. They were the first to explore the effects of seq2seq models on automatic word problem solving. Their proposed two models outperformed all the previous statistical models designed in this field.

Most of the word problem solvers perform poorly on datasets consisting of diverse word problems like the Dolphin18K [19] dataset. Huang et al. [19] showed that a simple similarity based retrieval model outperformed its sophisticated statistical counterparts on large datasets. The significant Number Identification (SNI) module is critical in identifying relevant numbers inside a word problem. The SNI module is an LSTM-based binary classification model. Each training sample for this model consists of a number and its context. The length of the context window for each sample is 3. If a number is significant, then it would be replaced with a number symbol as $n_1, n_2, n_3, ...$ The seq2seq model was 5 layers deep, with a word embedding layer, a 2-layer GRU as the encoder and a 2-layer LSTM as the decoder. The retrieval model found out the lexical similarity between a test problem and all the training problems where each word problem was represented as a TF-IDF vector.

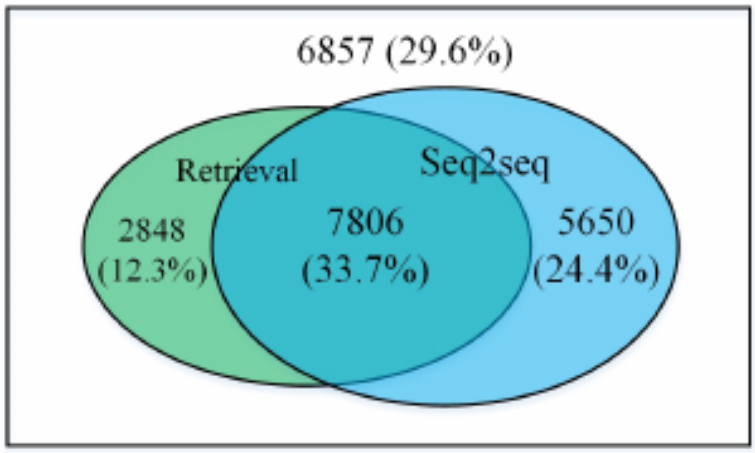

Figure 2.1: Hybrid Model Consisting of Seq2seq and Retrieval Models

The accuracy of the retrieval model was positively correlated with the maximal similarity score between the target problem and the problems in training data. The authors observed that the seq2seq and retrieval models complemented each other which is shown in Figure 2.1.

2.3.2 Goal Driven Tree Structured Neural Solver

Xie et al. [20] modeled word problem solving in a goal driven manner. The proposed solver generates an expression tree for a given word problem using a tree structured neural network. It identifies the goal to achieve and then decomposes it into multiple subgoals recursively based on the operators involved. For encoding the input problem text, the authors used gated recurrent

units on word embeddings and a recursive neural network as a decoder. They also showed that the inclusion of subtree embeddings during the expression tree generation improved the performance of the solver. The solver performed better than the sequence to sequence model based solvers. Using this goal driven approach, they were able to eliminate the generation of invalid mathematical expressions and spurious numbers that did not appear in the input problem.

2.3.3 Other Neural Approaches

Ling et al. [21] introduced a concept of answer rationale for solving word problems. Each answer rationale consists of a sequence of steps required to solve a problem. For this, they use a latent sequence of instructions where the solver first learns to convert an input word problem into an instruction sequence and then learns to generate a solution based on the predicted instructions. Similar to Ling et al. [21], Huang et al. (2018) [22] presented an intermediate representation for generating equations that were motivated by semantic parsing. Another neural approach [23] used tree LSTM decoders for generating equations required to solve a problem. Griffith et al. (2019) [24], Griffith et al. (2021) [25], Wang et al. [26] used transformer [27] based models and experimentally studied the difficulty level in generating equations with different notations (prefix, infix, postfix). Patel et al. [28] empirically showed that the current solvers are only learning shallow features to solve word problems in the benchmark datasets. The authors also pointed out a severe limitation of the solvers where they predicted the equations without looking at the questions. Other studies Sundaram et al. (2022) [29] suggest that deep learning based approaches are inadequate for learning the mapping of linguistic features of word problems and the mathematical concepts.

As a part of this work, we have implemented a frame based approach and neural approaches for word problem solving. For training our models, we have developed datasets. We also highlight the requirement of data augmentation techniques and show their effectiveness in building better and robust models. We tested our systems' performance on benchmark datasets and gave a comparative analysis with the current systems. We briefly discuss the deep learning concepts mainly used in this thesis.

2.3.4 Background for Our Approaches

The input for word problem solving is a sequence of words and the output is an equation conditioned on the input which is again a sequence of symbols. The symbols denote the operands and the operations. In most word problems, the operands are explicit and present in the problem text whereas the operations are implicit and are to be inferred from the input problem. We modeled word problem solving as a sequence transduction task. For this, we focused mainly on neural models and utilized two kinds of architectures for modeling our experiments.

- Sequence-to-Sequence Architecture
- Transformer Architecture

2.3.4.1 Sequence-to-Sequence Architecture

Sequence-to-Sequence [30] or Seq2Seq or encoder-decoder models have been in wide usage for several tasks such as machine translation, summarization, question answering. The efficacy of these model has been further enhanced by the introduction of attention mechanism [31, 32]. While generating an output symbol at each decoding step, the decoder has the information about the whole input sequence as well as the previous output symbols. The hidden state at each decoding step is weighed with every hidden state of the encoder to compute the attention score. The weighted sum of the hidden states of the encoder is often called the context vector and passed to the decoder. We also used a variant of this architecture to design an end-to-end word problem solver in Figure 6.2.

2.3.4.2 Transformer Architecture

Transformer [27] based models are the current best performing models in several NLP tasks. These models have superior computation efficiency than recurrent models due to parallelizability. This enables transformer models to be trained in significantly less time. The architecture is shown in Figure 2.2.

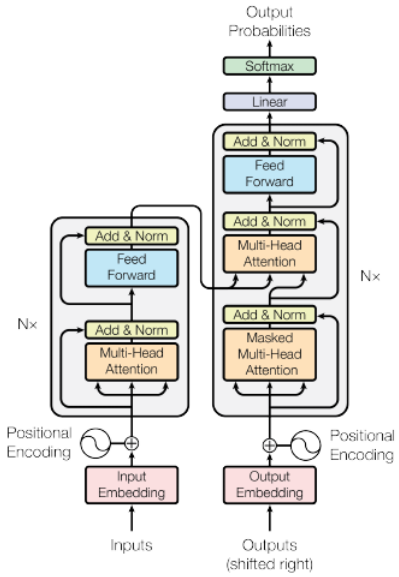

Figure 2.2: Transformer Architecture (the figure is from the original paper)

The architecture is similar to a Seq2Seq [30] encoder-decoder model. The major difference lies in the processing of the input vectors. The vanilla encoder-decoder model [30] processes the inputs in a linear fashion whereas the trasformer can function on the inputs in a parallelized format. For understanding the order of the input sequence, positional encodings are added to provide the information about the positions of the tokens. Self-attention helps the model capture the contextual information in an effective manner by looking at all the words in a sequence while processing a word. The self-attention is applied multiple times in parallel which allow transformers to model the relationships between words in multiple perspectives. Each attention layer in the encoder and decoder is followed by a feed-forward network to learn complex patterns. Layer normalization is applied after each layer during training with residual connections.

Transformers use scaled dot product attention mechanism where there are 3 primary matrices associated with the input sequence: queries (Q), keys (K), and values (V). d_k denotes the dimensions of Q and K and d_v is the dimension of V.

The attention is computed as:

$$Att(Q, K, V) = softmax(QK^T/\sqrt{d_k})V \tag{2.1}$$

This attention is not computed once. Istead it is computed multiple times with linear projections parallely and generating multiple output values. These output values are concatenated and projected to yield the final values. This is called multi head attention and is useful to learn better representations for the input sequence.

Chapter 3

Benchmark Datasets and Evaluation Strategies

Chapter 2 gives an overview of the approaches followed for word problem solving. To ascertain which approach is more accurate, evaluation becomes an essential step. The quality of benchmark data and the metrics for evaluating models are important in presenting a more accurate picture of which model is better in terms of solving a wider range of word problems. In this chapter, we will also shed light on the evaluation strategies used for measuring the performance of the Word Problem Solving (WPS) approaches.

3.1 Evaluation Metrics

Let us understand different evaluation metrics with the following example.

- Question: *John has 2 pens and 4 pencils. Robert has 4 pens. How many pens are there?.*
- Solution: 6
- Equation: $X = 2 + 4$
- Derivation: $X = a + b; a = 2\ pens; b = 4\ pens$
- Solution with associated entity: 6 *pens*

3.1.1 Solution Accuracy

Solution accuracy is calculated as the fraction of correctly solved problems with the total number of word problems. Most systems are evaluated in terms of solution accuracy. They do not take into account how the answer is arrived at. If a solver predicts the answer to the above question as 6, then it will be counted as a correct answer.

3.1.2 Equation Accuracy

Equation accuracy is the fraction of the total number of correctly identified equations with the total number of word problems. For the above question, a solver just needs to identify the equation as $X = 2 + 4$. The question contains two mentions of the number '4'. If a solver is evaluated in terms of equation accuracy, it does not need to align the exact mention of the number. Most of the systems are evaluated in terms of equation accuracy. In the above example, the equation accuracy will label an equation incorrect even if the predicted equation is equivalent to the given equation. A predicted equation $X = 4 + 2$ for the given example would be deemed incorrect.

Equation accuracy used in many previous works ignored the equation or expression equivalence property shown below.

- Equivalent Expressions: $(17 - 5) + 3 = (17 + 3) - 5$

This becomes very crucial while evaluating the performance of solvers that can handle multistep arithmetic word problems. We leveraged this concept and proposed a new evaluation method [33] for equation accuracy.

3.1.3 Derivation Accuracy

[9] devised a new evaluation strategy of comparison of derivations. A derivation consists of an equation template and an alignment of text to coefficients. They showed that when a word problem solver was trained on derivations, it performed significantly better than when trained on only equations. This is a new direction of evaluation that future systems can use to show their system's efficiency. This metric will help solvers design better generalizable systems.

3.1.4 Associated Entity Accuracy

An operation between two quantities can only be performed when they share the same unit. In the above word problem, '4' is associated with two entities in two mentions - *pencils*, *pens*. The first mention is irrelevant for answering this question as units are different. So, a solution can be embedded with the associated entity for a better evaluation metric. The solver is required to output a numeric quantity as well as the associated entity with the quantity.

3.2 Benchmark Datasets

Different benchmark datasets of varying sizes have been developed to train and test word problem solvers. Most of them were constructed by crawling popular math tutoring websites like `www.algebra.com`, `www.mathtutor.com`, `www.math-aids.com`. AI2 dataset [6] was a curated

version of such problems with a very limited vocabulary. [34] showed that many benchmark datasets were composed of problems with high lexical overlap and equation template overlap which will be studied in detail in the following sections. [34] proposed a greedy search technique to eliminate such frailties of datasets and showed 20-30% reduction in the size of all the datasets. They created a repository called "MAWPS" collecting all word problem datasets released by different authors. [35] showed that many datasets had vocabulary biases. In AI2 dataset, the verb 'give' was only associated with subtraction operation, it completely ignored the addition aspect of the verb. ALGES514 [7] contains 514 problems constructed from only 28 equation templates. Similarly, [35] empirically showed that current solvers perform poorly on an unbiased dataset and they suffered from non generalization. They released a dataset comprising 1492 problems by eliminating the existing biases. Dolphin18K [19] is one of the biggest datasets in terms of size and variety of problems. It has 18460 word problems and 5871 equation templates. They showed that a simple similarity match between a problem and all the training problems yielded better results than other models built using more sophisticated features on this dataset. AQuA [21] is the biggest dataset for word problem solving. It contains 101449 triples of algebraic questions, answers, and rationales. All these datasets are available in English. Math23k [18] and Ape210k [36] are two large datasets available in Chinese. Both the datasets contain elementary word problems needing only one unknown variable to solve them. But Ape210k dataset is more diverse in terms of questions and equation templates. ASDiv [37] is the most diverse English dataset for single variable word problems covering a large number of text patterns. It also contains annotations of problem type and grade level.

3.2.1 Issues in Current Benchmark Datasets

This section discusses the issues in current benchmark datasets.

Lexical Overlap

As mentioned earlier, many benchmark datasets consist of similar types of word problems with variation of only a few words. The following example demonstrates the nature of lexical overlap. As the number of equation templates for single variable word problems containing one or two operators is small, we do not discuss the template overlap here.

1. *Joan went to 4 football games this year . She went to 9 games last year . How many football games did Joan go to in all ?*

2. *John went to 5 football games this year . He went to 6 games last year . How many football games did John go to in all ?*

3. *Joan went to 3 baseball games this year . She went to 8 games last year . How many baseball games did Joan go to in all ?*

Questions 1 and 2 differ only in terms of the named entities specifically the person names and pronouns associated with them. Questions 1 and 3 are different only in terms of the entities associated with the quantities. The entity or unit in question 1 is 'football games' whereas 'baseball games' is the identified entity or unit in question 3. Changing the subjects or persons' names and the associated entities is the most widely adopted technique for creating new word problems in benchmark datasets.

Ungrammaticality

As many of the benchmark datasets are created using crowdsourcing, there are possibilities of grammatical errors. This was first shown by [34] in their work. They used ERG parser [38] to spot ungrammatical questions. We show some of the examples of such questions present in one of the benchmark datasets.

- Incorrect Number Marker in Nouns - *Joan found 70 seashells on the beach . she gave Sam some of her seashells . She has 27 seashell . How many seashells did she give to Sam ?*
- Incorrect Usage of Adjectives or Adverbs - *At a pie-eating contest , Erik got through 0.6666666666666666 pie before time was called ; Frank finished just 0.3333333333333333 pie . How much more pies did Erik eat than Frank ?*

3.2.2 Reduction of Lexical Overlap

In order to tackle the problem of lexical overlap, we devise a strategy to remove highly overlapping word problems from a dataset which results in maximum diversity inspired by the earlier works [34, 39]. Let $T(p)$ denote the set of unigrams appearing in a word problem p after removing all numeric quantities and punctuations. We meausre the lexical overlap between two word problems in terms of *Jaccard Similarity*. Let $LexSim(p, q)$ denote the lexical similarity between the word problems p and q computed as $LexSim(p, q) = |T(p) \cap T(q)|/|T(p) \cup T(q)|$

Let D denote a dataset containing n word problems. As a first step, we set a threshold value th for lexical similarity. For any word problem $p_i \in D$, we calculate lexical similarity of p_i with other problems $p_j, \forall j, j > i$. We remove all the word problems from D satisfying the property $LexSim(p_i, p_j) >= th$.

We use this technique on the benchmark datasets in English and present the results for different thresholds in Table 3.1.

We can observe that there is highest overlap between problems in the CC (Common Core) dataset containing word problems which need multiple arithmetic operations to solve them. ASDiv dataset is the most diverse dataset of all the datasets containing a single unknown. MAWPS also follows a similar trend as this is an amalgamation of different datasets including

Dataset	Size	Reduced Size For Different Similarity Thresholds					
		0.5	0.6	0.7	0.8	0.9	1.0
AI2 [6]	395	185	228	275	333	380	389
ASDiv [40]	2305	1948	2131	2227	2274	2298	2298
IL [10]	562	269	333	394	444	481	483
Single-Eq [34]	508	353	386	416	449	483	496
MAWPS [34]	2373	894	1035	1179	1316	1450	1802
CC [10]	600	114	116	117	118	121	364
Unbiased [35]	1492	856	1035	1197	1327	1431	1473

Table 3.1: Reduction of Datasizes after removal of similar problems

AI2, IL, Single-Eq, and CC datasets. This method can be used as a preprocessing tool for any dataset before designing a solver to leverage the property of diversity among problems.

We have used all the above English benchmark datasets except Unbiased for our approaches. A portion of the Hindi word problems used in our study includes Hindi translations [33] of word problems in the Unbiased dataset.

In this chapter, we have shed light on the nature of different benchmark datasets. If the datasets have significantly higher overlap between the constituent problems, any evaluation done on them will be an overestimation of the proposed systems. In chapters 7 and 8, we will explore how diversity in a dataset contributes to the efficiency of a word problem solver.

Chapter 4

Word Problem Solving Using Frame Identification

Although current word problem solving approaches are mostly neural, these systems lack explainability. They directly output equations or solutions without showing the intermediate steps of the solving process. We initially explored schema based techniques which deduce the equation and answer through a systematic interaction of schemas. These solvers can act as teaching aids for school children as these are not only capable of answering quantity related questions, but also they can answer any query related to different associated entities. They can help students understand language comprehension as well as mathematical concepts. In addition, this can be used as a guide to understand sentences containing quantities and their associations with other terms. Inspired by this, we present here a novel approach for automatic arithmetic word problem solving where frame identification acts as the main fulcrum for WPS.

4.1 Introduction

Early approaches [1, 3, 2, 4, 5] have relied on the concept of schemas for solving word problems. Most of these could only solve word problems with a single addition or subtraction operation. 3 major schema types were proposed: change, part-whole, and compare. These techniques provided a cognitive framework for better explainability. However, these systems had severe limitations in their ability to solve a wide range of problems. Bakman [4] extended the schema representation for better coverage and multi step arithmetic and introduced the notion of extraneous or irrelevant information. However, the approach only dealt with addition and subtraction operations. So, there was a need to develop frames for other arithmetic operations too. In this study, we developed frames encompassing all the operations. The frames are inspired by the concept of verb semantics.

4.2 Definition

A frame is a basic computational unit consisting of relevant information for solving a word problem. Instead of directly defining frames at a single level, we abstract them at two levels. The first level decides the role of a frame in a word problem. At this level, the frames are categorized based on the operations they evoke which is detailed in the next section. At the first level, a frame is either a *State Frame* or an *Action Frame*. State frames and action frames are identified by the verbs and other words in context. State frames are created based on stative verbs such as own, possess, contain, or phrases such as "there are", "there exists" etc. State frames act as the entity holders or containers for an entity. No operation is associated with these frames. Action frames correspond to verbs other than the stative verbs. Action frames act on state frames either causing a change in quantities of the state frames or create state frames as a result.

Every frame is unique and is identified by its slots. The slots are filled using the dependency parsed output of a sentence. The slots include entity holder, entity, quantity of the entity, recipient, and additional information such as place and time. The slots and frames help to identify the type of question asked and the entities referred. The frames are then used to build a graph where any change in quantities can be propagated to the neighboring nodes. Most of the current solvers can only answer questions related to a quantity, while our system can answer different kinds of questions such as 'who', 'what' other than the quantity related questions 'how many' due to the presence of different kinds of slots indicative of their roles.

Frame Types

To decide the number of frame types, a thorough study was done using English Framenet [41]. After analyzing different verbs and their corresponding frames, we came up with a list of 40 frames. However, many frames from this list had overlapping properties in terms of participants and conceptual roles. After eliminating the classes that were causing a high degree of ambiguity, we arrived at a concise list of 22 frame types, as shown in Appendix A. Some of them are mentioned here. The action frames are given with their associated operation inside parentheses. Action frames require one or two entity holders to perform actions. Transfer frames require two entity holders evoking addition operation in one entity holder while subtraction is performed in the other one. Other action frames only act upon single entity holders.

- State Frame
  - Possess
  - Contain

  - State Fact
  - Existence
- Action Frame
  - Gather (+)
  - Transfer Money (+, -)
  - Transfer Goods (+, -)
  - Use Resource (-)
  - Duplication (*)
  - Separate Entitites (/)
  - Create (+)
  - Getting (+)

In this chapter, we will discuss 3 major aspects of word problem solving using frame identification.

1. Frame Annotated Corpus (with a frame annotation tool)
2. Frame Identification Module
3. A new easily understandable framework for word problem solving

4.3 Corpus Annotation

As there were no annotations available for the frames, we started this study with the development of a corpus. For this task, the basic unit of annotation was a sentence.

We made the following assumptions.

- Every sentence has a single verb.
- Every sentence has a single frame as per the verb.

The questions for annotation are selected from the worksheets available under `https://www.math-aids.com/Word_Problems/`.

#Questions	**#Sentences**	**#Frames**
504	1421	1421

Table 4.1: Annotated Corpus Size

As frames are triggered by verbs, we created a list of frames and a list of words or verbs corresponding to a frame. 2 annotators were involved in the frame annotation task. The inter-annotator's agreement for frame annotation was 0.834 in terms of Fleiss' Kappa [1] which is considered perfect agreement.

Offline Frame Annotation Tool

We created a command line based tool using the Python programming language for facilitating annotation. The input for this tool is a question. First, the input question is split into its constituent sentences. For splitting the sentences, we used the Spacy [42] NLP toolkit. A verb to frame mapping is created to facilitate the annotation. The tool automatically identifies the frames if it finds any matching verb related to a frame. If no matching verb is found in any sentence, an annotator has to annotate the required frame ID or name for the frame. Finally, the equation required to solve the question is annotated.

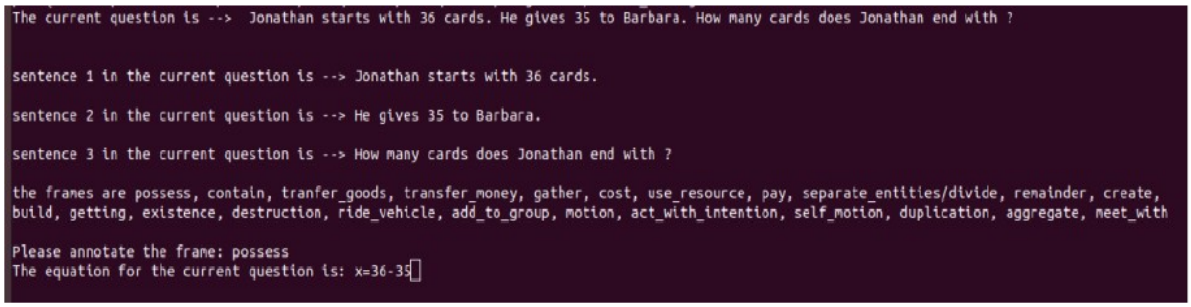

Figure 4.1: Initial Frames

For every question, the list of frames, the equation, and question related information are stored in an xml format which is as follows. All the text is converted into lowercase while saving the frames.

```
<question>
<questionid>1</questionid>
<questionstring>Jonathan starts with 36 cards. He gives 35 to Barbara.
How many cards does Jonathan end with ?</questionstring>
<framestring>jonathan starts with 36 cards.</framestring>
<frame>possess</frame>
<framestring>he gives 35 to barbara.</framestring>
```

[1] https://en.wikipedia.org/wiki/Fleiss's_kappa

```
<frame>transfer_goods</frame>
<framestring>how many cards does jonathan end with ?</framestring>
<frame>possess</frame>
</question>
```

4.4 Our Approach for Solving Word Problems

Our approach focuses on the extraction of knowledge from sentences, parsing the sentences, constructing frames, and finally solving the problem. Our approach is similar to Sundaram and Khemani [5]. We modeled every word problem as a graph of state and action frames. Action frames act on their respective state frames and state frames undergo changes in the constituent quantity. The intended question in a word problem is about specific slots in frames, most of the time the quantity. The slots are explained in the subsection 4.4.3. Our approach has the following steps:

- Preprocessing: Conversion of numbers in words to actual numbers
- Identification of Frames
- Parse Sentences
- Fill slots corresponding to frames
- Build a graph of frames
- Traverse the graph to find the answer

4.4.1 Approaches for Frame Identification

Frame Identification constituted the first step in our approach. We implemented different machine learning and deep learning based approaches for developing frame identifiers.

Machine Learning Approaches

The design of these classifiers was implemented using sklearn [43] machine learning library. The classifiers used were: Support Vector Machines [44] and Random Forests [45]. Each input text was represented as a TF-IDF [46] vector. TF-IDF (TF: Term Frequency, IDF: Inverse Document Frequency) assigns weights to words (or n-grams) based on its frequency in a document and its frequencies across documents to find out how important a word is to a document in a corpus. In this case, a document is a sentence. TF-IDF was calculated for word unigrams (uni) and bigrams (uni-bi) appearing in the text. We also experimented with character n-grams in

different ranges (2-to-6 and 3-to-6). We did not use any additional lexical or linguistic features such as parts-of-speech tags, morph features, wordnet [17] features, or dependency labels for the frame identification. The TF-IDF scores were computed at the sentence level. After trying out TF-IDF vectors at word and character level separately, we concatenated the vectors and retrained the models.

Deep Learning Approaches

Pre-trained language models trained on large corpora are proven to be useful in many NLP tasks. BERT [47] and its other variants [48, 49] are multi-layered and bidirectional encoder representations from transformer [27] models which are trained on huge corpora and can be fine tuned on any target downstream task to achieve performance improvements. For learning better contextual representation, these models are pre-trained with two tasks of masked token prediction and next sentence prediction. These models are very easy to fine tune in any supervised setting with just an addition of a single output layer using softmax activation. We used the Huggingface [50] transformer framework to fine tune the pre-trained available models for frame identification. Two variants of BERT were used for the experiments:

- Distil-RoBERTa-base: 6 layers of encoder, 12 heads, and 82 million parameters
- RoBERTa-base: 12 layers of encoder, 12 heads, and 125 million parameters

4.4.2 Results for Frame Identification

For both the approaches, we evaluated the models in a 5-fold stratified cross validation setting. This was performed due to the unbalanced nature of frame types in the annotated dataset. Both the transformer models were trained for 20 epochs with a learning rate of 0.00002 with a batch size of 16 and a weight decay rate of 0.01. The models were fine tuned on a GeForce RTX 1080 Ti GPU.

Tables 4.2 and 4.3 illustrate the results of the identifier using both the mentioned approaches. The best performing metrics are shown in bold.

4.4.3 Approach for Problem Solving

After the frame identification, we parsed each sentence to fill the frame slots. For dependency parsing, Stanford dependency parser [51] which was a neural network based parser was used. We used following slots or attributes for each frame which were identified from specific dependency labels whose dependency mapping are given below.

After each frame is created, a graph is built with all the frames. The quantity updates could be intimated to neighboring frames so that all the frames were updated at once. Each question queries specific frame slots which could be easily answered by traversing the graph once.

Model	Features	F1-Score
Linear-SVM	uni	0.87
	uni-bi	0.86
	char[2,6]	0.85
	char[3,6]	0.85
Random-Forest	uni	0.84
	uni-bi	0.81
	char[2,6]	0.84
	char[3,6]	0.85
Linear-SVM	uni+char[3-6]	**0.88**
Random-Forest	uni+char[3-6]	0.86

Table 4.2: Frame Classification with TF-IDF Features

Model	F1-Score
Distil-RoBERTa-base	0.94
RoBERTa-base	**0.95**

Table 4.3: Frame Classification with Transformer Based Models

4.4.3.1 Working Example of the System

For the arithmetic word problem: "John had 5 books. John gave Robert 2 books. How many books John have now?" The equation for this question is $x = 5 - 2$ and the solution is $x = 3$. We will derive the equation and the solution through the frames.

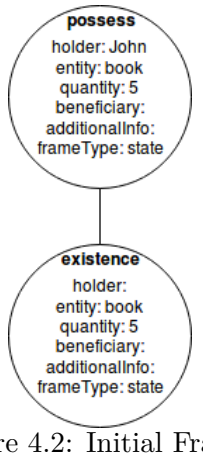

Figure 4.2: Initial Frames

Figure 4.2 shows the initial frames created after parsing the first sentence. In many word problems, questions are asked on information which are not present explicitly in the question. The answer to this kind of question can only be answered through proper inference. If the question in the above example is changed to "How many books are there?", a solver needs to infer that "Somebody has some books" means "There are some books." So, for every 'possess'

Dependency Label	Frame Slot
Subject	Entity Holder
dobj	Entity
amod	Attribute of Entity
iobj	Beneficiary
nummod	Quantity
nmod:case	Addidional Info

Table 4.4: Dependency labels to Frame Slot Mapping

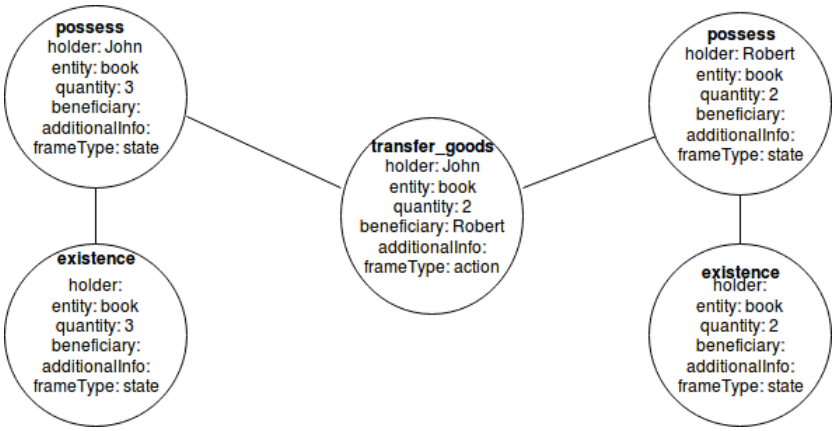

Figure 4.3: Complete Graph of Frames

frame, an existential frame is created and vice versa. So in Figure 4.2, there are two frames connected to each other instead of one. Both these frames are state frames. The second sentence gives information about a transfer operation carried out between two entity holders. The order and type of operation can be found by matching the entity holders and entities. So in this case, the transfer_goods frame triggers a subtraction operation in one possess frame with an update in quantity slot $5 - 2 = 3$. It also creates another possess frame triggering an addition operation with quantity coming from $0 + 2 = 2$. Once the updation happens in any 'possess' frame, it automatically gets updated in its neighboring 'existence' frame. Similarly, this kind of updation is repeated for all frames that are attached to 'possess' and 'existence' frames. The question sentence is also parsed to find out the frame type and type of question asked. "who" kind of question seeks an answer from entity holder slots of the frames, similarly "what" maps to entity slots. "how many" questions interrogates about the quantities involved in the frames. In the current system, the relation between all these question types and frame slots are predefined. Each action frame is associated with an operation which is pre-defined.

4.4.3.2 Results for Problem Solving

Table **4** shows the comparison of our system with ARIS. Our system with SVM based verb categorization was able to solve 115 questions out of 302 questions of the AI2 dataset

containing single addition and subtraction operations. This result improved when we integrated the Roberta-base verb categorization with our solver. The data and models can be found on

System	Accuracy
ARIS	77.7%
Our System + SVM Verb Categorization	37.8%
Our System + Robertabase-uncased Verb Categorization	43.2%

Table 4.5: Comparison of System Accuracy with ARIS

`https://github.com/Pruthwik/Frame-Identification-Models`.

4.5 Error Analysis and Observation

Error	Percentage	Example
Non-Linear Sequence	9	Joan found 70 seashells on the beach. Joan gave Sam some of her seashells. She has 27 seashells. How many seashells did she give to Sam?
Parsing Errors	27	Tim had 7 quarters and 9 nickels in his bank. His dad gave him 3 nickels and 5 pennies. How many nickels does Tim have now?
Coreference Errors	11	Joan grew 24 pumpkins, Keith grew 42 pumpkins, and Alyssa grew 13 pumpkins. They worked for 34 days on the farm. How many pumpkins did they grow in all?
Incorrect Frame Identification	15	Mary is baking a cake. The recipe calls for 7 cups of flour and 3 cups of sugar. She already put in 2 cups of flour. How many cups of flour does she need to add?
World Knowledge	18	Students at Arcadia schools are participating in a coat drive. 9437 coats have been collected so far. 6922 coats were collected from the high schools, and the rest from the elementary schools. How many coats were collected at the elementary schools?
Others	20	While playing a video game, Paul scored 3103 points. He and his cousin together have a total of 5816 points. How many points does Paul's cousin have?

Table 4.6: Error Categories and their frequencies in percentage

We found 6 major sources of errors by analyzing the errors made by the solver. We present the categories of errors in table 4.6.

The frame based solver expects the state of the world to be linear. If the order is scrambled, the solver incorrectly predicts the answer. In the example under parsing errors, the dependency of "now" is "dobj" which confuses the solver to consider it as the entity. Here, the solver fails to find any matching frame. In another example, the word "rest" refers to a subtraction operation which comes from world knowledge. Incorrect frame identification is another major source of error. In many cases, the coreferences are not resolved accurately. We used the CoreNLP suite [16] for coreference resolution. In the above example, the pronoun "they" was not resolved correctly which caused the solver to output an incorrect solution.

Between the machine learning (ML) models, Support Vector Machines performed better than Random Forests. Unigram TF-IDF vectors were the most salient features. After a grid search to determine the best possible features at word and character levels, we chose unigram and 3 to 6 character grams. When both these TF-IDF vectors are concatenated with each other, it spiked the performance in both the ML classifiers. But BERT based transformer models outperform the ML models by a significant margin of 7% in terms of F1-scores. This validates our hypothesis about the fine tuning of pre-trained language models. RoBERTa, an optimized BERT model was evaluated to be the best model. Distilled versions of the transformer model using the technique of knowledge distillation are also comparable to the best performing model. The major ambiguities for transformer based models lied in frames that evoke similar type of operations. We show the ambiguity pairs with the help of examples below.

- Duplication, Separate Entities
  - The pencils are divided evenly among 9 classrooms.
  - Gold: Separate Entities (due to the phrase 'divided evenly')
  - Prediction: Duplication (due to the word 'evenly' that is similar to the word 'each')
- Aggregate, Transfer Money
  - How many packs of groceries did she buy in all?
  - Gold: Aggregate (due to the word 'all')
  - Prediction: Transfer Money (due to the word 'buy')
- Create, Getting
  - She grew older and got 6 more on her birthday cake.
  - Gold: Create (due to the word 'grew')
  - Prediction: Getting (due to the word 'got')

Other kind of errors appeared when a sentence has multiple verbs. The model predicted the frame type for the latter verb whereas the gold annotation refers to the type of the first verb

as shown in the $3^r d$ example. This is a case of wrong annotation. Another interesting case was observed while solving a word problem using frame types. If the frame in a sentence is predicted incorrectly, but belongs to the same frame type or evokes a similar operation, then the solver is able to find the correct answer. This scenario is explained below through a worked out example.

- Question: *A village has a population of 30000. 5000 people immigrate to the village last year. What is the population of the village?*
- Sentence 1: *A village has a population of 30000.*
  - Gold Frame Type: *Contain* (we assign *possess* frame to persons, not locations)
  - Predicted Frame Type: *Possess*
- Sentence 2: *5000 people immigrate to the village last year.*
  - Gold Frame Type: *Add to Group*
  - Predicted Frame Type: *Add to Group*
- Equation: *30000 + 5000*

Almost all the arithmetic problem solvers output only the answers or equation, but our system outputs step wise explanation along with the answer and equation.

Even though the motivation of the design of the frames came from FrameNet [41], the output of our system is not similar to FrameNet. If a verb had different meanings, we did not create different frames for different semantics. We focused more on the computational part of the involved frames.

4.6 Conclusion

In this chapter, we present an easily understandable framework for solving arithmetic word problems. We hope that this can assist the teacher in explaining arithmetic operations with the help of frames and slots appearing in them. In our approach, we have predefined action frames performing arithmetic operations. This task can be learned which action frame does what operation. As we rely on external tools to solve word problems, this introduces the problem of cascading errors. So, in the coming chapters, we will attempt to minimize these errors by designing end-to-end systems for WPS.

Chapter 5

Equation Generation by Composition of Classifiers

Frame based solvers as shown in the previous chapter are highly explainable and can very well act as teaching assistants. However, they often are not robust and require a large number of frame types to solve a variety of word problems. As neural based solvers can address these limitations of the frame based solvers, we move towards neural approaches. In this work, we developed two kinds of approaches. In the first approach, we decompose word problem solving into two different tasks: a. Operation Prediction and b. Relevant Operand Identification. Then, the final equation is composed of these constituents. The second approach attempts to develop a neural end-to-end arithmetic word problem solver that generates the equation components at once. The first approach is detailed in this chapter, whereas the next chapter describes the second approach.

We designed different systems for the first approach where the final equation is composed of the predictions from different classification tasks.

5.1 DILTON

Instead of taking the word problem as a single string and learning the representation of the whole string, we split it into problem text and question sentence. In this attempt, we develop a solver, DILTON [1]. It first predicts the basic arithmetic operation ('-','+','*','/') through a deep neural network based model, extracts the relevant operands, and then uses it to generate the equation and answer. Its architecture is inspired by memory networks [52] by learning separate representations for supporting sentences and the question sentence and then concatenating them. This separation is shown in Figure 5.1.

[1] Dilton was considered the smartest teenager in his school. We wanted to design an efficient system, so this name was chosen `https://en.wikipedia.org/wiki/Dilton_Doiley`

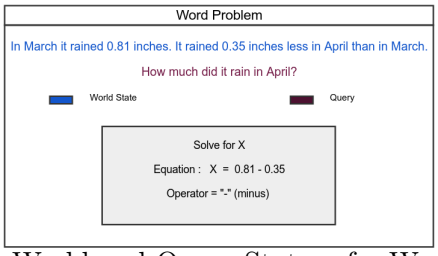

Figure 5.1: World and Query States of a Word Problem

5.1.1 Architecture

Our system is a pipeline consisting of 3 different modules that are detailed below. The workflow is shown in Figure 5.2.

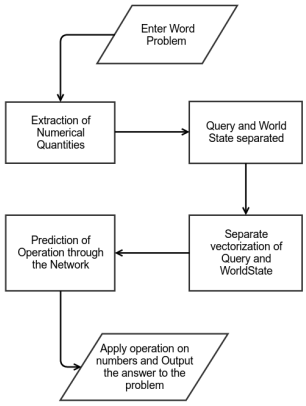

Figure 5.2: Workflow of DILTON

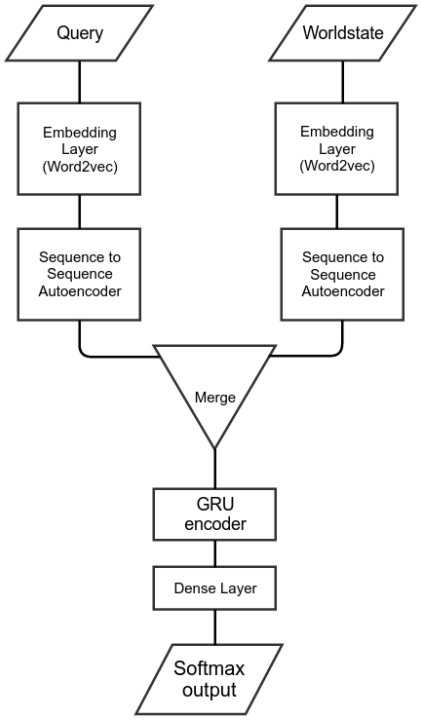

Figure 5.3: Architecture of DILTON

5.1.1.1 Sequence Encoder

The input to our system is the problem text P which has 2 relevant quantities $num1$, $num2$. As a first step, P is split into 2 parts.

1. Query - Question sentence

2. World State - the word problem without the final query which has the information required to answer the query.

We used word2vec [53] to convert each word in the world state and query for their vector representation. We then used sequence encoders with Gated Recurrent Unit (GRU) [54] to encode both the world state and the query separately. We merge these two separate representations by doing an element-wise sum.

5.1.1.2 Operand Prediction

To find the operands in a word problem, we need to first filter out irrelevant quantities.

- Question: *John has 3 pens and 2 pencils. Jane have John 5 more pens. How many pens John have now?*
- Relevant Quantities: *3 pens, 5 pens*
- Irrelevant Quantities: *2 pencils*

In this question, the quantity 2 is irrelevant which can be easily found out by a similarity match between the context of the quantity and the question asked. The quantity is associated with an entity 'pencil' whereas the queried entity is 'pen'. We experimented with different context window lengths across quantities and reported the results in Table 5.1.

5.1.1.3 Predicting the answer

The last layer in our architecture is a softmax layer, which is fully connected to the encoder's output, as shown in Figure 5.3. The output layer consists of 4 nodes corresponding to the basic arithmetic operators. The node with the highest probability is selected as the operator. Then an equation is composed of the relevant operands and the operator. Finally, we execute the equations to get the final answer of the word problem.

5.1.1.4 Dataset

We have used the publicly available MAWPS [34] dataset. This dataset consists of word problems containing a single addition or subtraction. The dataset contains 1751 single step arithmetic word problems.

5.1.1.5 Experimental Setup

We train this whole network end to end using categorical cross entropy loss and Adam [55] optimizer. The dropout rate [56] is set to 30% for regularization and to prevent overfitting. We

used 300 sized word2vec [53] pre-trained embeddings and network learned embeddings of 64 dimensions using a GRU [54] to encode both query and world state. The network was trained for 40 epochs.

5.1.1.6 System Evaluation

We evaluated our system on two datasets: one being the original dataset 5.1.1.4 and the other being a subset of AI2 [6]. We evaluated DILTON in a 5-fold cross validation setting on the original dataset. For the cross validation, we had 3 kinds of configurations. In the first configuration **rel**, only the relevant quantities in a word problem are substituted with special number tokens e.g. $\{num1, num2, ..\}$. These relevant quantities are extracted by using the gold equations. In the second configuration **all**, all the numeric values in the word problem are replaced with the special number tokens. In the third configuration **all+hidden**, an additional layer of 100 nodes is inserted before the output layer. The results are shown in Table 5.1. The AI2 dataset consists of 186 questions that require multiple operations to get solved. Currently, DILTON can not solve these questions. We compared our system against the 209 problems with single operations in the AI2 dataset. DILTON shows significant improvement over their accuracies. Table 5.2 shows the equation accuracy for different sizes of context window for any quantity.

Configuration	**Oper_Acc**	**Context Window**	**Quants_Acc**	**Equation_Acc**
rel	**88.23**	1	**97.72**	**84.81**
		2	97.69	84.75
		3	97.69	84.75
all	88.01	1	95.92	81.5
		2	94.98	79.72
		3	94.92	79.61
all+hidden	86.81	1	95.92	80.64
		2	94.98	78.92
		3	94.92	78.64

Table 5.1: 5-Fold Cross-validation results: Oper_Acc is Operation Accuracy, Quants_Acc is Quantity Accuracy, and Equation_Acc is Equation Accuracy

Context Window	Quant_Acc	Equation_Acc
1	**92.25**	**81.92**
2	79.35	70.47
3	77.74	69.04

Table 5.2: Quantity Identification and Equation Formation Accuracy on AI2

5.1.1.7 Error Analysis

We can observe from the results that the operation prediction accuracy in configurations rel and all in 5-fold cross-validation is very similar. 75% of the incorrect operation predictions occur in operators with same precedence i.e. the pairs of addition, subtraction and multiplication, division. We can also see that smaller context windows around numeric quantities perform better than larger context windows at finding relevant quantities. But relevant quantity prediction is significantly impacted if the number of irrelevant quantities increases which is evident from the superior performance of **rel** configuration than the **all** configuration. Including a hidden layer before the output layer does not improve the operation prediction accuracy, therefore the overall equation accuracy drops.

5.2 Fine-tuning Pre-trained Transformer Models

This technique provides a perfect setting for a task like word problem solving which is modeled as a decomposition of multiple classification problems. As large language models (LLM) are trained on huge sized corpora, fine-tuning them on some downstream tasks presents a test bed for evaluating their reasoning capacity in a complex NLU task of solving arithmetic word problems. The first task at hand is the relevance prediction of a quantity. It is a binary classification task where 1 means a quantity is relevant and 0 indicates the irrelevance of a quantity. The second task is a multi-class classification problem, as explained earlier. The classifier is tasked to predict the operation required for solving the word problem that is selected from the set of operators $\{+, -, *, /\}$. Two special tokens are added at the time of tokenization: CLS and SEP. CLS is added at the start of a sentence, which denotes sentence classification, while SEP is appended towards the end of a sentence, which acts as a separator. CLS token actually represents the meaning of a whole sentence and its hidden state representation is fed to classification layers. This kind of preprocessing is done with BERT models. We explored 2 variants of Bidirectional Encoder Representations from Transformers [47] or popularly known as **BERT** for these tasks. We briefly explain their architecture and usage.

RoBERTa

RoBERTa [48] is a robustly optimized version of the original BERT model. The original BERT was pre-trained using two unsupervised tasks namely masked language model (MLM) and next sentence prediction (NSP). RoBERTa uses dynamic masking unlike fixed masking in BERT and is trained on only complete sentences removing the NSP loss in training phase. This model is trained on 10 times larger corpora than BERT and uses larger batch sizes than the initial BERT model. Like BERT, RoBERTa and its distilled version replace the CLS and SEP tokens by $< s >$ and $< /s >$ to mark the start and end of the sentence or sequence. It also

utilized byte level byte pair encoding (BPE) [57] instead of widely followed character level BPE to learn better universal representations for the input sentences. For our experiments, we use RoBERTa-base with 12 encoder layers and 12 attention heads. The size of the hidden nodes is 768. The number of parameters in this model is 125 millions.

Distil RoBERTa

Distil RoBERTa is a distilled version of the RoBERTa model. We have used the distilled version of the RoBERTa base model. It also uses byte level BPE for tokenizing the input text segments. This model has around 35% fewer parameters (82 million) than the RoBERTa base. This has 6 encoder layers and 12 attention heads. Its major advantage is its speed. It performs comparable to the original model in most of the NLU tasks.

5.2.1 Quantity Relevance Prediction

Previous neural methods utilized this prediction task, but no dataset was publicly available. Hence, we built a dataset for the task relevant operand identification. We annotated 3718 such samples from 1751 sentences.
As the pre-trained models are trained on subwords, multi digit numbers are grouped into multiple subwords. In order to avoid this kind of undesired splitting, we substitute each number with single letters such as p, q, r, and so on. Earlier arithmetic solvers also followed this convention to mitigate the limitations of sparse representations for different numbers. For this task, we created two types of samples:

1. Tokens in a context window around a quantity

2. Tokens in a context window around a quantity and the question sentence

The size of the context window is 7 for this task i.e. $\{w_{i-3}, w_{i-2}, w_{i-1}, w_i, w_{i+1}, w_{i+2}, w_{i+3}\}$. If a token is not present at the mentioned index which usually happens towards the start and end of a sentence, we leave that index out. This setting was first suggested in Wang et al. [18]. Most of the neural approaches use the same setting for identifying the relevance of quantities or operands. For creating data samples, we followed some pre-processing steps which are detailed in the section below.

Tokenization

As an initial step, every word problem is tokenized using white spaces and punctuations.

Question Sentence Identification

We utilized Spacy parser [42] to parse a word problem with the identification of sentence boundaries. Often, the last sentence is presented as a question to solve. We apply an additional heuristic where the last sentence consists of an 'if' clause to find out the exact question span.

- Word Problem: *Marvin has p eggs. Jacqueline has q eggs. If Jacqueline gives all of her eggs to Marvin, how many eggs will Marvin have ?*
- Last Sentence: *If Jacqueline gives all of her eggs to Marvin, how many eggs will Marvin have ?*
- Question Text: *how many eggs will Marvin have ?*

While creating data samples with the context window and question, we concatenate them only when there is no lexical overlapping between them. Otherwise, we take only the non overlapping text.

The following example presents the approaches for data creation.

- Question: *Joshua has 12 Skittles and 7 eggs. If he shares the Skittles among 4 friends, how many Skittles does each friend get?*
- Tokenization and Replacement of Numbers: *Joshua has p Skittles and q eggs . If he shares the Skittles among r friends , how many Skittles does each friend get ?*
- Question Sentence: *how many Skittles does each friend get ?*
- Context Window Around Quantities with Relevance Score [0 if Irrelevant, 1 if Relevant]
  - Joshua has *p* Skittles and q - *1*
  - p Skittles and *q* eggs . If - *0*
  - the Skittles among *r* friends , how - *1*
- Context Window Around Quantities with Question Sentence and Relevance Score
  - Joshua has *p* Skittles and q how many Skittles does each friend get ? - *1*
  - p Skittles and *q* eggs . If how many Skittles does each friend get ? - *0*
  - the Skittles among *r* friends , how many Skittles does each friend get ? - *1*

5.2.1.1 Experiments and Results

Model	Setting	Quant_Acc	Macro F1
Similarity Match	-	93.7	74.4
Distil-RoBERTa-base	Context Window	97.8	90.3
	Context Window+Question Sentence	98.1	91.5
Roberta-base	Context Window	98.0	90.8
	Context Window+Question Sentence	**98.2**	**92.1**

Table 5.3: Relevance of Quantity Prediction Evaluation

For this task, we created a training-free baseline. We computed the cosine similarity scores between the contexts of quantities and the question sentence. We select the quantities with highest scores. In this context, the top two are chosen. For representing the texts for the baseline, we utilized the sentence BERT embeddings [58]. The other models were implemented using different variants of RoBERTa [48] which is a robustly optimized version of original implemetation of BERT. All these models were finetuned using the Huggingface [50] Transformer framework.

5.2.1.2 Implementation Details

Both the models were implemented using the Huggingface [50] transformer library. All the experiments were conducted on a single NVIDIA GeForce GTX 1080 Ti computational GPU with 11GB of GDDR5X RAM. All the models for both relevant operand prediction and operation prediction were trained for 20 epochs. Each epoch takes around 2 minutes to complete.

5.2.1.3 Discussion

The data for the quantity relevance prediction or significant number identification (SNI) task is mostly unavailable. So, it is difficult to compare the efficacy of our approach with earlier approaches. Most of the previous systems reported only the accuracy. Benchmark datasets are highly skewed when irrelevant numbers in word problems are concerned, at most, 10% of the problems contain numbers that are insignificant. Hence, we added macro F1 and macro accuracy scores as evaluation metrics for the models as the datasets have high class imbalance. We also observed a similar pattern in the results shown in Table 5.3 where there is a wide gap between the F1 and accuracy scores. When the question sentence for a word problem is added

to the context window of a quantity, the model predictions become more accurate. This verifies our hypothesis that bigger contexts for quantity prediction using BERT like models are indeed highly beneficial. RoBERTa model performed the best, but the distilled versions of the same model gave comparable results without degrading the former.

5.2.2 Operation Identification

For the operation identification, the input to the model is the complete question and the output is the required operation. We will take the same example as mentioned above.

- Question: *Joshua has 12 Skittles and 7 eggs. If he shares the Skittles among 4 friends, how many Skittles does each friend get?*
- Tokenization and Replacement of Numbers: *Joshua has p Skittles and q eggs . If he shares the Skittles among r friends , how many Skittles does each friend get ?*
- Operation: / or Division

Model	**Micro F1**
Distil-RoBERTa-base	94.06
roberta-base	**95.89**

Table 5.4: Operation Prediction Evaluation

Model	**Config**	**Operand Pred**	**Operation Pred**	**Equation_Acc**
Distil-RoBERTa-base	No Question	90.32	94.06	89.3
RoBERTa-base		90.76	**95.89**	90.9
Distil-RoBERTa-base	With Question	91.5	94.06	89.8
RoBERTa-base		92.1	**95.89**	**91.3**

Table 5.5: Equation Accuracies by Composing Operand and Operation Prediction

We use RoBERTa-base and the distilled version of RoBERTa-base (Distil-RoBERTa-base) for the task of operation identification. The results are reported in Table 5.4.

5.2.3 Discussion

The equation accuracy was computed by composing the predictions from the two models as shown in Table 5.5. The configurations denote two different settings for operand predictions. RoBERTa-base model performed the best in both the configurations. Operand predictions get a boost when the question sentence is added to the context of the operands, thus improving the overall equation accuracy. A similar pattern is also observed in operation identification where the results of the distil RoBERTa were comparable to the original model. In the case of operation identification, the major ambiguity lies in multiplication and division. Two examples are shown below:

- Word Problem 1: *I have p cents to buy candy . If each gumdrop costs q cents , how many gumdrops can I buy ?*
- Target Operation: /
- Predicted Operation: $*$
- Word Problem 2: *What is divided by p to get q ?*
- Target Operation: /
- Predicted Operation: $*$

As pointed out by Patel et al. [28], we also observe that deep learning networks focus primarily on shallow features of word problems where certain words are only attended to while deciding either operands or operations.

5.3 Conclusion

In this chapter, we designed different classification tasks to predict various components, such as operands and operations needed to solve a word problem. An equation is formed from these constituents and solved to get the intended solution. We implemented different techniques for these classification tasks and showed their efficacies. This kind of modeling will only work when the number of classification tasks is limited. In this case, the number of operands is 2 with a single operation. With the increase in the number of operands and operators, the number of classification tasks increases exponentially and becomes intractable to solve. The next chapter discusses end-to-end equation generation at once instead of depending on the outputs from the tasks.

Chapter 6

End-to-End Equation Generation

The previous chapter generated the required equation for a word problem by composing the results of operand and operator prediction modules. The overall performance of such a solver is impacted due to the error propagation from different modules. To overcome this limitation, we develop approaches to generate equations for word problems at once.

We experimented with multiple approaches to develop end-to-end equation generators which are detailed in the following sections.

6.1 EquGener

We introduce a novel method where we first learn a dense representation of the problem description conditioned on the question. We leverage this representation to generate the operands and operators in the appropriate order for forming the equation. This approach is unlike several sequence-to-sequence learners where the complete input word problem is fed to the encoder at once. This puts an additional constraint on the decoder as it has no clue of what is being asked in the question.

Our solver is an end-to-end memory network with an equation decoder. Our system handles problems involving a single arithmetic operation and can be extended to multi arithmetic operations. We call our system *EquGener*.

6.1.1 Base model

An attention based encoder-decoder [32] has been used as a baseline for our equation generation system as shown in Figure 6.1. Both the encoder and decoder employ Long Short-Term Memory (LSTM) to represent the input and target sequence respectively. In this architecture, the input sequence is encoded as a sequence of word vectors and the decoder has access to all these vectors instead of a single vector. Each word vector is a concatenated vector representation of pre-trained Glove [59] embeddings and the embeddings learned by the network from

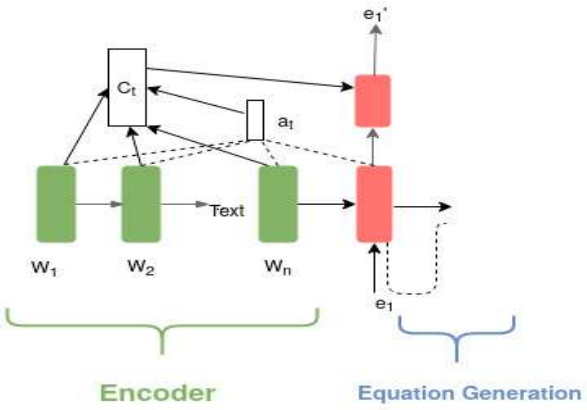

Figure 6.1: Architecture Diagram of Base Model

the training corpus. The equation generation for a word problem requires identifying words that indicate the presence of operands and operators. The j^{th} hidden state h_j of the encoder is computed as Equation 6.1 using an LSTM which is a function of previous hidden state h_{j-1} and current input s_j.

$$\mathrm{h_j} = \mathrm{f}(\mathrm{h_{j-1}}, \mathrm{s_j}) \tag{6.1}$$

The decoder is initialized with the hidden and cell states obtained at the last time-step of the encoder. Each hidden state of the input word sequence and the hidden state of the equation are compared to arrive at the alignment. The attention at time step t a_t is computed as per Equation [**?**]:

$$\begin{aligned} \mathrm{a_t} &= \mathrm{align}(\mathrm{h_t}, \tilde{\mathrm{h_s}}) \\ &= \frac{\exp(\mathrm{h_t}^{\mathrm{T}}.\tilde{\mathrm{h_s}})}{\Sigma_{\mathrm{s}'}\exp(\mathrm{h_t}^{\mathrm{T}}.\tilde{\mathrm{h_s}'})} \end{aligned} \tag{6.2}$$

The context vector c_t is computed as the weighted combination of the hidden states from the word sequence:

$$\mathrm{c_t} = \Sigma_t \mathrm{a_t} \times \mathrm{h_t} \tag{6.3}$$

The attentional hidden state on the decoder side is obtained by concatenating the context vector from the input word sequence c_t and equation hidden state h_t. W_c is a learned weight matrix.

$$\tilde{\mathrm{h_t}} = \mathbf{W_c}[\mathrm{c_t}; \mathrm{h_t}] \tag{6.4}$$

6.1.2 Memory Network Based Encoder

End-to-End memory networks [52] succeeds in representing sentences as well as captures the salience or the intent of the question in Question Answering systems. We used a variant of memory network *EquGener* to solve arithmetic word problems. The word representations in the supporting sentences act as memories and these are weighted as per the question. The relevant memories are assigned higher weights than the irrelevant ones. This weighted combination of memory vectors is then learned by the encoder to obtain a hidden representation of the word sequence appearing in the supporting sentences conditioned on the question words. The decoder then generates the equation conditioned on the encoded hidden representation. Two kinds of memory network settings are generally followed : (1) explicit identification of supporting sentences where the answer components lie and (2) without any information regarding supporting sentences. We used the later configuration for our system which required less supervision than the former. Considering our input to be a sequence of words $w_1, w_2, w_3, .., w_n$, these words are embedded into a lower dimension space d which can be achieved via an embedding matrix $\mathbf{A}$ of size $d \times V$ where V is the vocabulary size. These embeddings are learnt during training. Each word is represented as a vector concatenating its glove embedding [59] and its transformed inputs. *EquGener* learned the words that had to be attended to based only on the question text. This strategy helped us in handling words which were not available in pre-trained word vectors like num_i in the problem text where num_i represents the quantities or operands.

$$\mathrm{m_i} = [\mathrm{w_p}; \mathrm{w_e}] \tag{6.5}$$

where w_p and w_e denotes the pre-trained embeddings and learned word embeddings respectively. m_i denotes the memory at position i. The question sentence in a word problem is also embedded using another matrix B of a similar dimension to A. A is used to embed the words appearing in supporting sentences while embedding B is used for the words in the question. This represents the internal state $\mathbf{u}$ for the question. The match between internal state and memory elements helps in predicting the components involved in an equation. The match is found using a dot product between u and m_i and then applying a softmax function over it. For a single hop memory network, u is computed by passing all the question words through an embedding layer. If the embedding dimension is d and maximum question length is denoted as q_max, then u will be a matrix of dimension $q\_max * d$. For a multihop memory network, the u is updated iteratively as shown in Equation 6.10.

$$\begin{aligned} \mathrm{p_i} &= \mathrm{softmax}(\mathrm{u^T}.\mathrm{m_i}) \\ \mathrm{softmax}(\mathrm{x_i}) &= \frac{\mathrm{e^{x_i}}}{\sum_{\mathrm{j=1}}^{\mathrm{n}} \mathrm{e^{x_j}}} \end{aligned} \tag{6.6}$$

where n refers to the total number of words in the supporting sentences. There is a probability score for each word appearing in memories. Each memory m_i has a corresponding output vector

c_i which is obtained through another embedding matrix C. The output vector is a weighted sum of p_i and c_is.

We used a slight modification to this formulation used in memory network [52] which is given below in Equation 6.7. We describe two formulations for single layer and multi-layer memory networks.

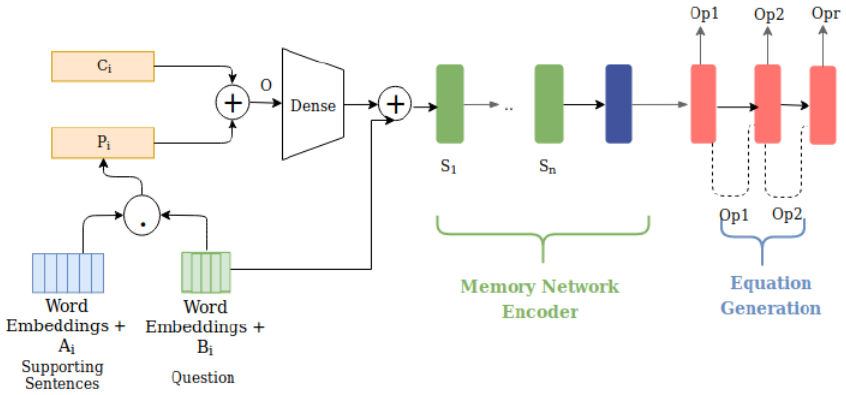

Figure 6.2: Architecture Diagram of EquGener for Single Layer

Single Layer

The output memory representation o is a dense representation of the words in the supporting sentences conditioned on the question.

$$o = \sum_{i=1}^{n} p_i + c_i \tag{6.7}$$

This output vector is passed through a fully connected dense network to capture a vector which is of equal dimension as the word representations. The sequence is fed to an LSTM encoder for representation of the sequence. This sequence is a sum of embeddings for the query and the vector after passing the output vector through the dense network.

$$d = \text{Dense}(o) \tag{6.8}$$

$$E = u + d \tag{6.9}$$

Multi-Layer

In the case of a multi-layer memory network, the hidden state gets updated in each hop with the discovery of new attention points in the memories according to the question. Here, the same

input and output embeddings are used across the layers. For a k-hop memory network where the memory layers are stacked on top of each other, the internal state is updated as follows,

$$\mathrm{u_{k+1}} = \mathrm{u_k} + \mathrm{d_k} \tag{6.10}$$

and at the final hop K,

$$u_{K+1} = u_K + d_K \tag{6.11}$$

The computation of d_k is same as the single layer case. The hidden state of the encoder is computed as per the equations 6.11-6.17 [1]

$$\mathrm{f_t} = \sigma(\mathrm{W_f} \cdot [\mathrm{h_{t-1}}, \mathrm{x_t}] + \mathrm{b_f}) \tag{6.12}$$

$$\mathrm{i_t} = \sigma(\mathrm{W_i} \cdot [\mathrm{h_{t-1}}, \mathrm{x_t}] + \mathrm{b_i}) \tag{6.13}$$

$$\tilde{\mathrm{C}}_\mathrm{t} = \tanh(\mathrm{W_C} \cdot [\mathrm{h_{t-1}}, \mathrm{x_t}] + \mathrm{b_C}) \tag{6.14}$$

$$\mathrm{C_t} = \mathrm{f_t} * \mathrm{C_{t-1}} + \mathrm{i_t} * \tilde{\mathrm{C}}_\mathrm{t} \tag{6.15}$$

$$\mathrm{o_t} = \sigma(\mathrm{W_o} \cdot [\mathrm{h_{t-1}}, \mathrm{x_t}] + \mathrm{b_o}) \tag{6.16}$$

$$\mathrm{h_t} = \mathrm{o_t} * \tanh(\mathrm{C_t}) \tag{6.17}$$

6.1.3 Decoder

The decoder takes the encoder representation learned in the memory network and predicts the sequence of operands and operators. The decoder is initialized with the last hidden state and cell state from the encoder. The decoder also uses an LSTM to predict the next output. The decoder predicts the output distribution using teacher forcing [60]. The hidden and cell states are computed according to the LSTM equations defined in section 6.1.2. In figure 6.2, the output tokens are referred to as $Op1$, $Op2$, *and* Opr which stand for the operands and the operator of the equation.

6.1.4 Experimental Setup

Operation	**Frequency**
$+$	472
$-$	445
$*$	226
$/$	171
Total	**1314**

Table 6.1: Frequency Analysis of Operations in Training Data

[1] http://colah.github.io/posts/2015-08-Understanding-LSTMs

6.1.4.1 Data

We used 1314 arithmetic problems with a single operation present in the MAWPS [34] dataset as our training set. The operations include all basic mathematical operations: addition (+), subtraction (-), multiplication (*), and division (/). The three benchmark datasets for evaluation are MA1, MA2, and IXL dataset [61] which are subsets of the AI2 dataset [6] released as a part of project Euclid [2]. We chose problems with only a single operation from these datasets - 103 from MA1, 118 from MA2, and 81 from IXL dataset.

6.1.4.2 Preprocessing

Every number appearing in a word problem is replaced by num_i in a random fashion where $i \ \varepsilon \ [1, 6]$. This is done to minimize the sparsity of different kinds of numbers appearing in the problem text. This also assisted in learning better representations of numbers in word problems. NLTK [62] was used to tokenize the sentences in a word problem. The last sentence in the list of tokenized sentences was considered as the question sentence and the rest as supporting sentences. The equations were labeled in postfix notation, e.g. *num1 num3 +*

6.1.4.3 Setting

There are **112** short trees and **119** tall trees currently in the park . Park workers will plant **105** short trees today . How many short trees will the park have when the workers are finished ?
There are num_3 short trees and num_4 tall trees currently in the park . Park workers will plant num_2 short trees today . How many short trees will the park have when the workers are finished ?
S_1. There are num_3 short trees and num_4 tall trees currently in the park . S_2. Park workers will plant num_2 short trees today . Q. How many short trees will the park have when the workers are finished ?

Table 6.2: Preprocessing Steps For Word Problems

The development set was fixed to 5% of the training data. The embedding weights for these were uniformly initialized. Dropout [56] rate of 0.2 was used while learning the embeddings A, B, and C. The embedding and output dimensions for the LSTM were set to be 64. The

[2] http://allenai.org/euclid.html

concatenated vector representation for each word is a vector of size 364. Maximum number of words appearing in the supporting sentences and question sentences were 56 and 34 respectively. No dropout rates were specified for the LSTMs. The recurrent weights were initialized as a random orthogonal matrix. The input weights were assigned from a Glorot [63] uniform distribution with the bias being initialized to zeros. Following this procedure, we were able to reduce the output vocabulary size. The encoder and decoder hidden and cell states were also fixed to be of dimension 64. Keras [64] deep learning library was used to build the network. Adam [55] optimizer was used for the optimization of the parameters. The system was trained for 50 epochs with the validation set. In the case of multiple hops, the same embeddings, A and C, were used in different layers.

$$A_1 = A_2 = .. = A_K \tag{6.18}$$

$$C_1 = C_2 = .. = C_K \tag{6.19}$$

$$u_{k+1} = u_k + o_k \tag{6.20}$$

We experimented with only 2 hops.

For the base model, the same glove embeddings and the same configurations for LSTM and embedding layer were used.

6.1.5 Comparison With Other Systems

System	MA1	IXL	MA2
ARIS[6]	83.6	75.0	74.4
KAZB[7]	89.6	51.1	51.2
Mitra et al.[61]	**96.27**	82.14	**79.33**
Att. encoder-decoder	92.23	71.61	63.56
EquGener 1 hop	91.26	**85.19**	55.08
EquGener 2 hops	94.18	**85.19**	55.08

Table 6.3: Comparison of the proposed system with other systems. Numerical values represent % of problems solved

We compared our system with other systems on MA1, MA2, and IXL datasets. Table 6.3 shows the accuracy of the systems in solving word problems in terms of the percentage of problems solved.

EquGener outperforms KAZB [7] significantly. KAZB uses a joint log-linear model distribution over a full system of predefined equations and alignments between the text and equation templates. As the alignment space is exponential while aligning with the slots, beam search is employed to find approximate solution. KAZB uses surface level features for the words and does not employ any semantic representation of them. So KAZB performs poorly on IXL,

where there is an information gap and irrelevant quantities. *EquGener* makes use of the dense semantic representation and can identify irrelevant quantities easily. Mitra et al. [61] was the state-of-the-art system. *EquGener* performs better than it on IXL dataset which has more information gaps. Mitra et al. [61] classify each addition or subtraction problem into 3 concepts and each concept is associated with a formula. Different features are defined for each formula. Multiple formulas can be applied to a word problem and a log linear model scores each formula based on its features. The biggest limitation of this system is its reliance on external tools like wordnet, dependency parser, and conceptnet. The system can only solve addition and subtraction problems. *EquGener* does not need any computation of extra features to solve word problems. It can solve problems with any arithmetic operation. But EquGener's performance dips for MA2 dataset containing problems with a high percentage of irrelevant information which even performs worse than attention based encoder decoder. This is evident from its wrong predictions for operands which is 58% on average while the operator is correctly identified in 91% of the problems in MA2.

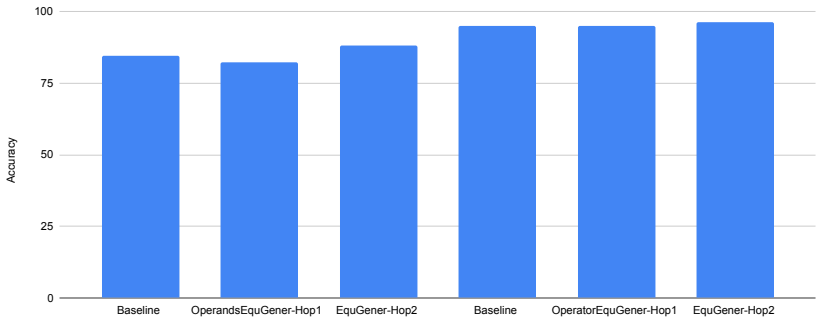

Figure 6.3: Operand and Operator Predictions for IXL

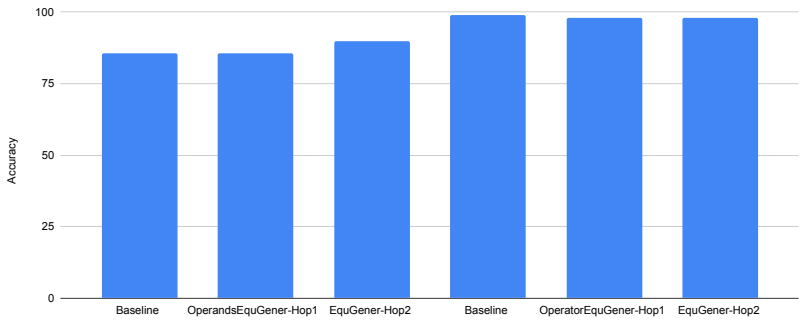

Figure 6.4: Operand and Operator Predictions for MA1

6.1.5.1 Operand Prediction

The operand prediction accuracies improve with the number of hops. *EquGener* requires several hops to identify the exact operands correctly. The major error in the operand prediction

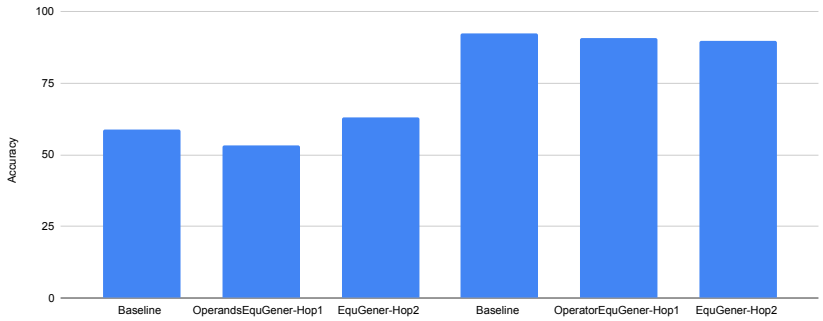

Figure 6.5: Operand and Operator Predictions for MA2

in our system resulted from the predictions that were out-of-order. The accuracy of the relevant quantity or operand prediction [65] was 89.1%. *EquGener* improved upon this accuracy by 3% in 2 of the datasets that are shown in Figures 6.3, 6.4, and 6.5.

6.1.5.2 Operator Prediction

EquGener outperforms the baseline system in operator prediction by a significant margin. This strengthens our hypothesis that memory networks are better at capturing the intention of verbs that have a direct correlation to arithmetic operations. The accuracy in verb categorization by ARIS [6] was 81.2%. The accuracy reported for LCA operation prediction was 88.7% with all features [65]. *EquGener* outperforms these two systems in operator prediction by a big margin with operator prediction accuracy 97% on an average on IXL and MA1, and 91% in MA2 that are shown in Figures 6.3, 6.4, and 6.5.

6.1.5.3 Error Analysis

Some of the erroneous outputs produced by *EquGener* are shown in the Table 6.4. In the first example, the system identified the operators accurately. However, the system could not identify the direction of transfer for the verb 'borrow', which resulted in an erroneous prediction of the order of equation components. We observe that this is a problem due to data sparsity and can be overcome by making more examples of this.

In the 2^{nd} example, the system predicted an operand *num5*, which is not present in the problem description. Though we expect the numerical values mentioned in the problem description not to be attached to any context, repeated occurrences may violate this assumption in certain cases. An architectural improvement to handle the numerical values can better the results. Similarly, in the 3^{rd} example, the system incorrectly identifies *num3* as a relevant operand. This error can be resolved by adopting an approach similar to Wang et al. [18] which modeled relevant operand identification as a classification task.

TestSet	Question	Predicted	Actual
MA1	Joan picked *num3* apples from the orchard . Melanie borrowed *num1* apples from her . How many apples does Joan have now ?	*num1 num3* -	*num3 num1* -
IXL	Tom went to *num1* hockey games this year , but missed *num4* . He went to *num3* games last year . How many hockey games did Tom go to in all ?	*num1 num5* +	*num1 num3* +
MA2	In num3 week , Mitch 's family drank num4 carton of regular milk and num2 carton of soy milk . How much milk did they drink in all ?	*num3 num2* +	*num4 num2* +

Table 6.4: Predicted and Actual Equations from Different Test Sets

6.1.5.4 Discussion

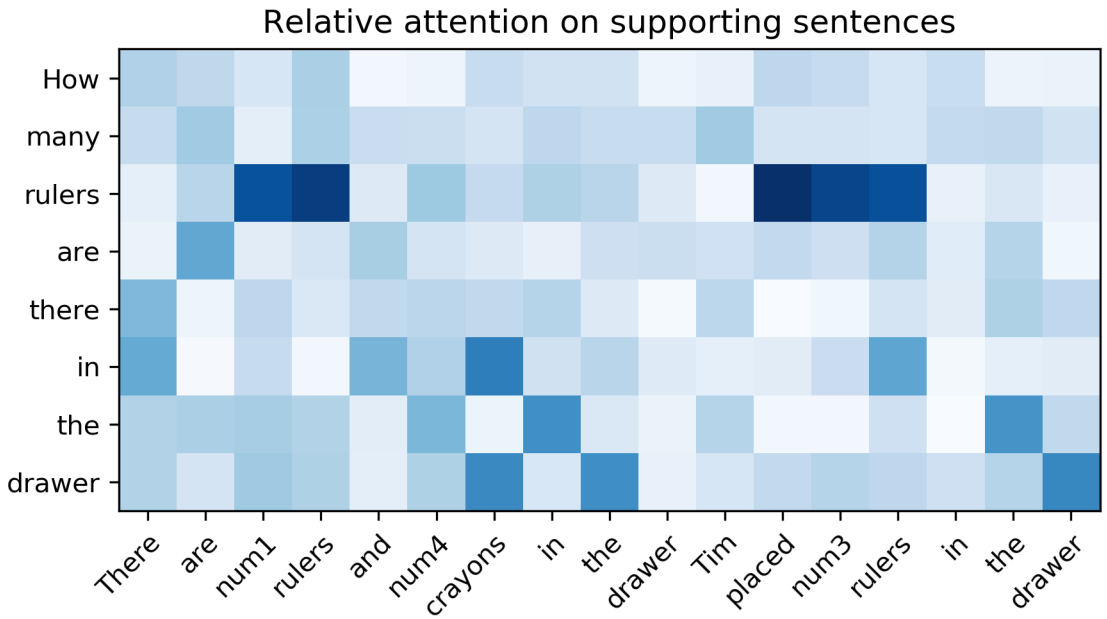

Figure 6.6: Heat Map of Attention for EquGener

The figure 6.6 below shows the relative attention of words in supporting sentences conditioned on question words. The words in the question are shown in the Y-axis, and the words in the X-axis constitute the supporting sentences. *EquGener* is able to figure out *num4* as an irrelevant quantity as it is associated with the entity crayons whereas the question is asked about the rulers. The verb "place" appears in the context of the rulers, so it also receives higher weights.

6.2 Equation Generation using other Neural Approaches

We implemented 2 kinds of architectures using the OpenNMT [66] toolkit.

- BiLSTM with Global Attention [31]
- Transformers [27]

6.2.1 Data Preprocessing

For the above mentioned architectures, pretrained subword embeddings [67] were used. Each word problem was tokenized using a subword tokenizer. The special number symbols e.g. 'num1', 'num2' etc. which were used in the earlier approaches got split into two tokens in the subword based tokenization. So, we use single alphabets 'p', 'q', 'r' for 'num1', 'num2', 'num3' respectively. One example of subword tokenization is given below.

- Original Word Problem: *A ship is filled with 10 tons of cargo . It stops in the Bahamas , where sailors load 15 tons of cargo onboard . How many tons of cargo does the ship hold now ?*
- Word Problem after replacing special number token: *A ship is filled with p tons of cargo . It stops in the Bahamas , where sailors load r tons of cargo onboard . How many tons of cargo does the ship hold now ?*
- Subword Tokenization: *_a _ship _is _filled _with _p _tons _of _cargo _. _it _stops _in _the _bahamas _, _where _sailors _load _r _tons _of _cargo _onboard _. _how _many _tons _of _cargo _does _the _ship _hold _now _?*
- Equation: $X = \_p \ \_r +$ (postfix notation)

6.2.2 Experimental Details

The configuration used for both the neural networks are given tables 6.5, 6.6.

6.2.3 Results

While training the models were evaluated after 100 validation steps. We tested the systems on 3 test sets as earlier systems. The results are shown in Table 6.8.

parameter	value
Word Embedding Size	300
Subword Embedding Size	300
Encoder Layers	2
Decoder Layers	2
Input Sequence Length	200
Output Sequence Length	200
Dropout Rate	0.3
Batch Size	64
Optimizer	Adam

Table 6.5: Configuration of BiLSTM model with Global Attention for English

parameter	value
Subword Ebedding Size	300
Encoder Layers	4
Decoder Layers	4
Heads	4
Input Sequence Length	200
Output Sequence Length	200
Nodes in Feed forward layer	2048
Dropout Rate	0.1
Attention Dropout Rate	0.1
Batch Size	64
Optimizer	Adam

Table 6.6: Configuration of Transformer Model for English

6.2.4 Comparison With Other Systems

System	MA1	MA2	IXL	Avg
ARIS[6]	83.6	74.4	75.0	77.7
KAZB[7]	89.6	51.2	51.1	64.0
Mitra et al.[61]	**96.27**	79.33	82.14	86.07
EquGener 2 hops	94.18	55.08	85.19	78.15
BiLSTM with Attention	94.18	**87.29**	**90.12**	**90.53**

Table 6.9: Comparison of proposed system with other systems for English. Numerical values represent Equation Accuracy

Name	#Word_Problems	Property
MA1	103	Most contain only relevant info
MA2	118	Most contain irrelevant info
IXL	81	Problems contain information gap

Table 6.7: Test set Statistics

6.2.5 Observation

BiLSTM with the attention based neural network performs better than the transformer network. The transformer based model reaches peak performance in a few number of training steps, but its performance degradation is drastic. However, the performance of the BiLSTM network with global attention is steady and is directly proportional to the increase in training steps. When compared to other systems, this model outperforms all the previous state-of-the-art scores. *EquGener* used word embeddings while the attention based BiLSTM model uses subword embeddings.

6.3 Finetuning Transformer Models

As explained in the previous chapter, we fine tuned pretrained transformer models for the equation generation task as well. For this, we used the encoder-decoder model based multi-task enabled text-to-text transfer transformer or *T5* models [68]. We zeroed in on this model because of its capability to transfer knowledge across different tasks pre-trained on a huge cleaned and naturally crawled English corpus. We used 3 variants of the T5 model whose details are shown in Table 6.10. All these models were implemented using the Huggingface transformer framework.

6.3.1 Results of Finetuned Transformer Models

T5 Small and T5 Base models were fine tuned for 20 epochs while the T5 Large model was fine tuned for 10 epochs. All the results are detailed in Table 6.11.

6.3.2 Discussion

We can observe that the T5-large model is superior to the T5-small and T5-base models. The T5-small model's performance degrades when there is missing and irrelevant information in the word problems, which is evident from its equation accuracies in MA2 and IXL datasets. Its operator identification accuracy is also low which cascades to result in lower eqution accuracies. T5-base is comparable to the large model in MA1 and IXL. But the T5-large model performs better in the presence of irrelevant information as well. T5-base and T5-large models outperform all the models described in this chapter.

Model	Data	Train_Steps	Equ_Acc
BiLSTM-Attention	MA1	500	17.476
		1000	74.757
		1500	**94.175**
		2000	**94.175**
	MA2	500	16.949
		1000	77.119
		1500	**87.288**
		2000	**87.288**
	IXL	500	22.222
		1000	83.951
		1500	88.889
		2000	**90.123**
Transformer	MA1	200	74.748
		300	67.961
		500	82.524
		1000	51.456
	MA2	200	53.39
		400	55.932
		500	47.458
		1000	44.915
	IXL	200	65.432
		300	61.728
		500	71.605
		1000	44.444

Table 6.8: Test Results for BiLSTM Attention and Transformer Networks for English

6.4 Performance Overestimation?

We calculated the average Jaccard similarity of the evaluation datasets with the training dataset to see whether overlapping between them impacts performance. We computed this similarity by first calculating the maximum similarity of each question in the test set with the training questions. As the next step, we performed the average of these maximum similarities to reach a score that denotes the overall similarity of the test set. The operands or numeric quantities were ignored while calculating the Jaccard similarity. Jaccard similarity can be defined as

$$Jac\_Sim(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

where Jac_Sim(A, B) is the Jaccard similarity between two sets A and B. In our case, one set contains the words in a test question and the other one is a collection of words in a question in the training dataset. This process is explained mathematically below in Equations 6.21 and

Model	Enc-Dec Layers	Attention Heads	Parameters (in millions)
T5-Small	6	8	60
T5-Base	12	12	220
T5-Large	24	16	770

Table 6.10: Details of the used T5 Models

Model	Dataset	Operand_Acc	Operator_Acc	Equation_Acc
T5-Small	MA1	96.12	91.26	91.26
	MA2	89.83	60.17	57.63
	IXL	98.76	67.9	66.67
T5-Base	MA1	**98.06**	**100**	**97.09**
	MA2	94.07	92.37	88.14
	IXL	**98.76**	96.3	95.06
T5-Large	MA1	**98.06**	**100**	**97.09**
	MA2	**97.46**	**99.15**	**96.61**
	IXL	**98.76**	**98.76**	**97.53**

Table 6.11: Results for Different T5 Variants

6.22.

$$max\_jac\_sim\_test_j = \max_{train_i \in Train} Jac\_Sim(test_j, train_i) \tag{6.21}$$

$$avg\_jac\_sim = (1/|Test|) * \sum_{j \in Test} max\_jac\_sim\_test_j \tag{6.22}$$

Test Set	Similarity Score
MA1	0.967
MA2	0.923
IXL	0.94

Table 6.12: Average Jaccard Similarity Scores of Test Sets with Training Set

where Train and Test denote the training dataset and test dataset respectively. We computed the average Jaccard similarities for all three test sets MA1, MA2, and IXL. This is shown in Table 6.12.

The evaluation accuracies are over estimated as there is very high similarity (1 means all the test examples are from the training set) between the evaluation and training datasets used in this chapter. This is also shown in earlier works on the MAWPS dataset, where the lexical overlap between the word problems is very high.

6.5 Conclusion

In this chapter, we showed the efficiency of deep neural networks in solving single variable word problems. The system has better accuracy compared to the frame based solvers or machine learning based solvers, but has a trade-off in terms of explainability. We also showed that the performance of the solver achieves its highest when the word problems in an evaluation dataset are lexically very similar to the word problems from the training set. In the next chapters, we show that diversity in word problems impacts the performance of the solvers.

Chapter 7

Data Augmentation Techniques

Any deep learning based technique requires reasonable amount of data to be trained on and perform well. Chapter 3 shows that the datasets available for word problems are resource poor in terms of number and type of problems. These datasets contain word problems with very high lexical and template overlapping. This makes the datasets' capability of generalization and solving unseen problems very limited. Therefore, one needs to increase the data size. Data augmentation techniques serve the purpose of ramping up the training samples. In this chapter, we will explore different data augmentation techniques to tackle this problem.

7.1 Data Augmentation Approaches

We briefly discuss a couple of data augmentation techniques here.

7.1.1 Easy Data Augmentation Techniques

Easy Data Augmentation (EDA) techniques [69] are simple data manipulation techniques for boosting the performance of text classification tasks. Table 7.1 shows the application of different EDA operations on a sentence "*John bought 10 apples.*". Although these techniques

operation	**sentence**
Random Insertion	*John sad bought 10 apples.*
Random Swap	*John bought apples 10.*
Random Deletion	*John apples 10.*
Synonym Replacement	*John purchased 10 berries.*

Table 7.1: Example of EDA Operations

can easily be incorporated to generate new word problems, many of the generated word problems

are either incomplete or completely ungrammatical. So, we have only made use of synonym replacement operation for generation in our proposed technique.

7.1.2 Data Augmentation using POS Tagging and Paraphrase Tables

Maimati et al. [70] introduced a new augmentation technique that ensures the quality of the generated sequence. Instead of just manipulating the words present in the input text, they utilize pos tags, paraphrase tables, and word embeddings for generating grammatical sequences. This technique consists of 3 steps for a sentence. In the first step, a sentence is tagged using the NLTK [71] pos tagger. Only content words tagged as noun, verb, adjective, or adverb are selected. In the second step, they use paraphrase tables [72]. A paraphrase table is a collection of lexical entries along with their corresponding pos tags and paraphrases. In this step, the embedding of words present in the paraphrase table is determined. The cosine similarity of the content words in the input sequence and the embeddings of words in the paraphrase table is evaluated. In the third step, the closest words from the paraphrase table entries are chosen and the original words are replaced with the most similar words to generate a new sentence.

7.2 Our Approach

We extend the augmentation technique proposed by [70] by adding linguistic constraints and other heuristics.

7.2.1 Selection based on Distributional Similarity

Instead of finding the closest match from all the words in the paraphrase table as explained in 7.1.2, we bin the content words present in the paraphrase table into the four content classes and discard the words that do not fall into these categories.

1. Adjective
2. Adverb
3. Noun
4. Verb

We used the Spacy toolkit [42] to Part-Of-Speech (POS) tag any input sentence. Only the tokens belonging to the above classes are selected for replacement. We used a pretrained word2vec [73] model for finding the embeddings of tokens. This word2vec model is trained on roughly 100 billion words from a Google News dataset and has a vocabulary of 3 million words. For any content word in the input sentence, the closest matches are searched in the pool of words corresponding to its POS class from the paraphrase table in terms of cosine similarity.

If a sentence has 'm' content words and top 'n' similar words for each content word are selected, then there will be a total of m^n possible choices for replacement. As any word problem consists of 2-3 sentences, this method will be able to generate an exponential number of word problems.

Let us look into some generated word problems.

- Original Word Problem: *For Halloween Debby and her sister combined the candy they received . Debby had 32 pieces of candy while her sister had 42 . If they ate 35 pieces the first night , how many pieces do they have left ?*

- Generated Word Problems:

    1. *For christmas Debby and her daughter combined the lollipop they received . Debby had 32 artifacts of lollipop while her daughter had 42 . If they ate 35 artifacts the first evening , how many artifacts do they have left ?*
    2. *For birthday Debby and her grandmother combined the lollipop they received . Debby had 32 artifacts of lollipop while her grandmother had 42 . If they ate 35 artifacts the first hours , how many artifacts do they have left ?*

The first example is generated when top 3 similar words are chosen for each content word. Similarly, the generation of the second example is done with the top 5 similar words for each of the content words.

We can see qualitative differences between the generations. An increase in the value of 'n' will create word problems with jarring word substitutions such as 'hours' in place of 'night'. In both examples, the substitution of 'artifacts' for 'pieces' is not apt. Although all other substitutions are acceptable in terms of grammaticality and synonyms, some substitutions degrade the overall word problem. So, we will add lexical constraints to improve the quality of the generated problems.

7.2.2 Additional Constraints

Substituting words based on just distributional similarity generates noisy sentences rather than grammatical ones. So, we include additional linguistic constraints in the technique described in 7.1.2 for generating word problems to ensure quality and grammatical correctness.

7.2.2.1 Synonym Similarity

Before replacing a word with a similar word determined by their closeness, we evaluate the synonym similarity [74] between the two words. This synonym similarity is measured as *'wup similarity'*.

'wup similarity' calculates how related two synsets are in terms of depth in the wordnet taxonomies, along with the depth of the Least Common Subsumer (LCS). It is calculated based on the distance between two synsets relative to each other in the hypernym tree.

If 'wup similarity' is less than a threshold, we consider the replacement infeasible. For our generation task, we fixed the threshold at 0.5.

7.2.2.2 Language Model

According to J.R. Firth, "a word is characterized by the company it keeps". So, a word cannot be blindly substituted by another word without considering its context and may end up in creating jarring sentences. We employ a language model to validate whether a word replacement is compatible with its context. We use a pre-trained KENLM [75, 76] language model for this which was trained on the Wall Street Journal corpus. The language model assigns a log probability score to an input text utilizing the saved probabilities upto 5-grams.

Let $w_{i-2}, w_{i-1}, w_i, w_{i+1}, w_{i+2}$ be a 5-gram and s_i be the word to be substituted in place of w_i. Let 'score' denote a function that assigns a score to a n-gram; here, we chose $n = 5$. Let 'th' represent a threshold.

If $score(w_{i-2}, w_{i-1}, w_i, w_{i+1}, w_{i+2}) - score(w_{i-2}, w_{i-1}, s_i, w_{i+1}, w_{i+2}) <= th$, then the $wi \rightarrow s_i$ substitution is deemed to be compatible. For our generation task, we fixed $th = 2$.

7.2.2.3 Gazetteer List

Different Gazetteer lists are also prepared for replacing different types of named entities, such as male names, female names, and currencies. There are other types of content words that are not present in the paraphrase tables and the similar words retrieved by the word2vec model are noisy. For the word "green", the three most similar words by the word2vec model are "wearin_o", "greener" and "workers_differently_Corenthal". In order to mitigate this problem, we also created a Gazetteer list of color names. This summarizes our proposed augmentation technique utilizing the paraphrase tables of content words.

7.2.3 Augmentation Using Translation

Ape210K [36] is the largest word problem dataset consisting of 210488 single variable word problems with 56532 equation templates. This kind of diversity both at the lexicon and templates can improve the performance of an automatic solver with respect to its ability for greater generalization and robustness to variations. But this dataset is available only in Chinese. Hence, to use this rich and diverse dataset for English and Indian languages, we first translated Ape210K dataset into English using a machine translation (MT) system. The choice of the machine translation system was finalized after completing a systematic study of the quality of translation of 100 word problems using 3 Chinese-English MT systems:

- Google Translate API
- In-House MT system [77]
- Opus MT System [78]

The translations were adjudged 'acceptable' if the output generated by an MT system is natural and fluent in English. The in-house and Opus MT systems generated disfluent and incomplete outputs for 30% of the word problems. Most of the Chinese to English translations made by the Google Translate API were acceptable. Hence, we finalized the usage of Google Translate API for this task. We translated 123430 word problem from the Ape210K dataset. The source was annotated with word problems and their corresponding equations. On the target side also, we copied the corresponding equations.
However, word problems generated by using MT systems have noise. We discuss the noise removal techniques used for our study.

7.2.3.1 Noise Removal Techniques

We employ different noise removal techniques.

1. **Matching of Named Entities**: For a Chinese word problem and its English translation, we first match the number of named entities in the Chinese text with its corresponding English one. If they match, then we consider it a valid translation. Otherwise we reject the translation. For this task, we used the Spacy [42] Named Entity Recognizers (NER) available for Chinese and English. This method of matching reduced the number of translations by two-thirds from 123430 to 47113.

2. *Non ASCII Characters*: A few word problems were of 'fill in the blanks' and 'symbol manipulation' type of questions using different non ASCII symbols. We filtered them out from our dataset. 178 of these are available in the translations. Some examples for such questions are shown below:

   - *Given that ▲-1.7=2.2, ▲+■=6.7, then what is the value of ■.*
   - *Specify a❍b=b*10+a*2, then what is the result of 2❍5❍9.*

3. *Erroneous Text*: Some translations contained erroneous text consisting parentheses. One such example is given below:

   - *A rope is 21 meters long, use (1/5), and leave the full length ((())/(())), if you use 6 meters, use the full length ((())/(())), the full length ((())/(())) is left, and how many meters are left.*

   We removed these kind of problems.

4. **Sampled Validation**: After removing the noise using the above methods, we sampled 500 samples from the translations. We manually validated them. If a translated word problem can be fully understood and solved using the given source equation, then we consider the word problem to be valid and solvable. We found only 3 problems unsolvable. Hence, we conclude that the dataset created using MT has valid word problems.

5. **Matching of Quantities**: After using all these noise removal techniques, we match the numeric quantities between the translation and its corresponding equation required to solve the Chinese word problem. We consider a problem valid if all the quantities in an equation is present in the problem text. This approach fails for implicit word problems which either involve unit conversion or domain knowledge to solve a word problem.
   For example: *The radius of a circle is 5 cm. What is its area?*. The equation to solve this word problem is $x = 3.14 * 5 * 5$. The value of 3.14 (pi or π) is not present in the problem text. This kind of domain knowledge is needed for solving geometry problems. We do not want to remove the valid translations for non matching numeric quantities.

   For our experiments, we split the translated datasets into two parts. The first part deals with *explicit* word problems where all the numbers in equations come directly from the word problems. The other type poses a different challenge for any solver. These are called *implicit* word problems where the required equation for solving the problem involves a numeric quantity that is not present in the problem text.

   Out of 44K translated word problems, 10965 are explicit in nature. Rest of the problems are implicit. As reported in Sharma et al. [33], solvers struggle to map words in a problem with an implicit number. They replaced implicit numbers with specific number symbols in an equation. Let us look at an example similar to the ones in Sharma et al. [33]:

   - Question: Harsh buys 48 apples from the market. How many dozens of apples does he have?
   - Equation With Numbers: 48/12
   - Question With Meta Symbols: Harsh buys p apples from the market. How many dozens of apples does he have?
   - Equation: 48/12
   - Equation With Meta Symbols: p/b

   Where b refers to the number 12 or dozen. Sharma et al. [33] show that this kind of implicit mapping is difficult to learn from word problems. For handling these type of cases, we introduced a method of embedding knowledge from a pre-defined knowledge base that can act as a cue for solving implicit word problems which will be explained in the next chapter.

#Operations	#Word_Problems
1	2955
2	4669
3	1476
4	548
Total	9648

Table 7.2: Operation-wise Distribution of Explicit Word Problems

A few word problems in the explicit dataset require more than 10 operations to solve them. We restrict ourselves to solving word problems that require maximum 4 operators, which is around 88% of the explicit dataset. The details of the explicit dataset are given in the table 7.2.

However, the remaining 30000 implicit word problems need to be verified by conducting experiments.

7.3 Impact of Data Augmentation Techniques on WPS

We demonstrate the efficacy of augmentation techniques on the performance of solvers. As an initial experiment, we trained solvers on the original AI2 [6] data and the ASDiv [40] dataset after applying the augmentation.

Experiments of AI2 dataset

We split each dataset in 80:20 ratio after shuffling for creating training and validation sets. As seen in the previous chapters, solvers achieved their highest performance while predicting equations written in postfix notation. This solver was created using a biLSTM network with global attention utilizing the openNMT deep learning framework as explained in the previous chapter. Pre-trained subword embeddings were used for representing word problems and equations. We used equations in postfix notation for all the experiments. Table 7.3 shows the details of the data used for this experiment.

Dataset	Train_Samples	Validation_Samples	Total
AI2	316	79	395
AI2+Augment	2072	517	2589

Table 7.3: Corpus Details for Data Augmentation Experiments

We tested the solvers on a completely unseen and different dataset IL (Illinois) [10] to study the impact of data augmentation. The test dataset consists of 321 word problems.

Model	Train_Steps	Equation_Accuracy
AI2 Model	600	32.399
	1100	34.268
	2000	31.152
AI2+Augment Model	1000	42.368
	1600	41.121
	2000	37.072

Table 7.4: Results of Solvers with Augmentation on IL dataset

Table 7.4 shows that the model trained on augmented data improves the performance by 8%. This proves the efficiency of the proposed data augmentation technique.

Experiments on ASDiv dataset

As shown in chapter 3, ASDiv [40] is the most diverse corpus available for single variable word problems in English. In this section, we want to investigate how big a factor diversity is for word problem solving.

ASDiv Dataset

No_of_Operation	#Word_Problems
1	985
2	338
Total	**1323**

Table 7.5: Distribution of Chosen ASDiv Word Problems

ASDiv dataset (Academia Sinica Diverse MWP Dataset) [40] was released as a diverse dataset containing English word problems of different grade levels and problem types from elementary mathematics curricula. We selected word problems with one or two operations. The dataset details are given in Table 7.5. We augment this dataset in 3 different ways and report the results in the next section.

Lexical Augmentation Using Paraphrase Tables

We augmented the ASDiv dataset with our proposed approach using distribution similarity of the content words from the Paraphrase tables. We also used all the mentioned constraints for the generation of word problems. We imposed another constraint on a number of valid lexical substitutions to increase the diversity in the generation. We also limit the number of generated word problems for a given word problem to 5 in order to avoid over generation of a single type of question. We did not use sentence level paraphrasing as the available paraphrasing models often changed the numbers in a word problem and as an effect of that the equation will also change.

No_of_Operation	**#Word_Problems**
1	1993
2	930
Total	**2923**

Table 7.6: Distribution of Augmented Word Problems with Original

The details of the dataset created after this augmentation are shown in table 7.6. We notice that the addition of constraints limits the number of generated word problems where the total number of generated problems is 1600. The average number of generations per word problem is 1.21.

Augmentation With Paraphrase Tables and Translation

No_of_Operation	**#Word_Problems**
1	4952
2	5613
3	1476
4	530
Total	**12571**

Table 7.7: Distribution of Operations for the Full Augmented Dataset

The final type of augmentation combined the generated word problems from both the augmentation techniques with the original dataset. In this experiment, we used the full dataset of explicit word problems as shown in table 7.2.

In the original dataset, the ratio of the number of questions with one operation to the number of questions with two operations is 2.91: 1. In the full augmented data in table 7.7, there are word problems with 3 and 4 operations whereas the number of word problems belonging to 1 and 2 operations is high. This dataset has 1098 unique equation templates. If the operands are replaced by a single meta tag, then the number of unique equation templates is 309.

Experimental Setup

We conducted 5-Fold stratified cross validation on the original and paraphrase table augmented datasets. All the three different equation notations: infix, postfix, and prefix are experimented with to study the impact of data augmentation on equation types. For the full augmented data, we report the average of 3 runs. For all the experiments, the operation distribution was taken into account for splitting the 5 folds in a stratified setting. The equations in Ape210k dataset are in infix notations. We converted them into postfix and prefix notations. We used an encoder-decoder model with pre-trained BERT embeddings to fine-tune on our datasets. RoBERTa-base embeddings were utilized for the first two experiments. For the full augmented dataset, we used XLM RoBERTa-base. There are 123 million parameters in the RoBERTa base model having 12 encoder and 12 decoder layers with 12 attention heads. XLM RoBERTa-base has 270 million parameters with the same number of encoder, decoder, attention head configurations as RoBERTa base. We froze all the encoder layers so that the gradients do not get updated for these layers during backpropagation. This reduced the number of parameters by half and also minimized the risk of overfitting. Most of the models were trained for 30 epochs. But, some of the models overfitted on training with higher epochs, hence an early stopping method was adopted. All the models were developed using the Huggingface deep learning library. We used a single NVIDIA GeForce GTX 1080 Ti GPU having 11GB of GDDR5X RAM using 6 CPU cores for the experiments.

Preprocessing

Earlier chapters shed light on the conversion of the numeric quantities into generic symbolic notations to avoid sparse representations. In a similar vein, we converted the numbers into 2 types of notations. This conversion is only applied to the relevant numbers in the problem text.

- Single Letters: p, q, r, ..
- String added with the occurrence count: number0, number1, number2,.., numberi

The example below will elucidate the process of conversion.

- Original Question: *The cafeteria had 51 apples . For lunch they handed out 41 to students and decided to use the rest to make pies . If each pie takes 5 apples , how many pies could they make ?*

- Converted Text With Single Letters: *The cafeteria had p apples . For lunch they handed out q to students and decided to use the rest to make pies . If each pie takes r apples , how many pies could they make ?*
- Equation Notations With Single Letters:
    1. Infix: ((p - q) / r)
    2. Postfix: p q - r /
    3. Prefix: / - p q r
- Converted Text With String and Occurrence Count: *The cafeteria had number0 apples . For lunch they handed out number1 to students and decided to use the rest to make pies . If each pie takes number2 apples , how many pies could they make ?*
- Equation Notations With String and Occurrence Count:
    1. Infix: ((number0 - number1) / number2)
    2. Postfix: number0 number1 - number2 /
    3. Prefix: / - number0 number1 number2

Results On Non Augmented Dataset

Config	Notation	Equation_Accuracy
Single Letters	Infix	43.91
	Postfix	45.04
	Prefix	44.06
String with Occurrence count	Infix	**45.5**
	Postfix	**48.29**
	Prefix	**49.96**

Table 7.8: 5 Fold Results With Different Number Representations on Original ASDiv Dataset

From table 7.8, it is evident that the conversion of numbers using 'numberi' (e.g. *number1*, *number2*, *number3* etc.) is superior to single lettered representations (e.g. *p*, *q*, *r*). This may have resulted due to high frequent subwords with single alphabets in words during unsupervised pre-training. Hence, we used this representation in our subsequent experiments. After analyzing the predicted outputs, we observe that there is a marked difference in terms of equation accuracy for different kinds of notations when the solver is trained on a small dataset. In this setting, the generation of equations in infix notations becomes challenging as there is very little data to learn the balancing of parentheses. 30% of the errors in infix equations are due to this imbalanced parantheses generation. With the increase in dataset size, as is seen in augmented

datasets, this issue gets resolved. Any significant difference between the equation notations ceases to exist in larger datasets.

Results On Augmented Datasets

The results on augmented datasets using different equation notations are detailed in table 7.9.

Config	Notation	Equation_Accuracy
Lexical Aug+Orig	Infix	49.1
	Postfix	49.9
	Prefix	47.6
Lexical Aug+Translation +Orig	Infix	46.4
	Postfix	48.1
	Prefix	46.6
Lexical Aug+Translation +Orig+1 and 2 operations	Infix	**53.7**
	Postfix	**56.5**
	Prefix	**54.6**

Table 7.9: Exact Equation Accuracies in the Full Augmentation Dataset

Config	Notation	Equation_Accuracy
Lexical Aug+Orig	Infix	50.3
	Postfix	50.7
	Prefix	50.6
Lexical Aug+Translation +Orig	Infix	50.0
	Postfix	50.3
	Prefix	47.7
Lexical Aug+Translation +Orig+1 and 2 operations	Infix	**56.7**
	Postfix	**58.4**
	Prefix	**56.0**

Table 7.10: Equation Accuracies with Equivalence in the Full Augmentation Dataset

Discussion

We observe from table 7.9 that augmentation improves the performance of the solver, but only marginally. With the full augmented dataset, we see a drop in the equation accuracy.

This is due to the fact that the dataset with full translations contains 3 and 4 operations, whereas ASDiv and lexically augmented datasets contain 1 and 2 operations. The drop in the performance of the solver is caused by its inability to generate equations with high number of operations. The full translation dataset also has less samples for these word problems. When we just report the equation accuracies for word problems with 1 and 2 operations, there is a marked improvement by 5%, and the best performance is reported with full augmentation. As a next step, we experimented with a sample from the full translations to study what quantity of data can be augmented to achieve optimal performance. In all these experiments, we only report equation accuracies which relies on direct matching between two strings. We observe that equation equivalence based evaluation accurately judges the model's performance which is shown in table 7.10. Otherwise, the performance is underreported using the exact equation accuracy. When the number of operations increases, the importance of having this kind of evaluation also becomes more necessary.

Augmentation With Sampled Ape210k Translations

We randomly sampled 1600 translations from all the full translations keeping the same distribution of operations as given in table 7.6. We kept the size the same for both the augmented dataset to study their individual effects. The distribution of the new dataset is shown in table 7.11.

No_of_Operation	**#Word_Problems**
1	3001
2	1522
Total	**4523**

Table 7.11: Distribution of Sampled Augmented Dataset

Experimental Setup

We conducted similar kind of experiments with the same parameters and model in a similar experimental setup.

Results

The results of the experiments with a sample of Ape210K and other augmentations are shown in Table 7.12. We do not report accuracies of one and two operations in this table as we chose a sample from the translations that included only one and two operations.

Config	Notation	Equation_Accuracy
Lexical Aug+Orig	Infix	49.12
	Postfix	49.88
	Prefix	47.6
Translation+Orig	Infix	45.5
	Postfix	48.29
	Prefix	49.96
Lexical Aug+Translation +Orig	Infix	**55.61**
	Postfix	53.97
	Prefix	54.86

Table 7.12: 5 Fold Results with Sampled Augmented Datasets with Exact Equation Accuracy

Discussion

From Tables 7.9 and 7.12, we can observe that there is no significant difference between them. Although the sampled dataset is 3 times smaller than the full one, the performance is very similar. An increase in data size does not always ensure an increase in scores. We can also see that when the original dataset is augmented with the translation, the equation accuracy does not improve significantly. This is even less than the results with the lexical augmentation.

We will discuss the role of diversity and relate it to performance in the next section.

7.4 Effects of Diversity on Solvers

Diversity plays a big role in a solver's generalization ability as well as the overall performance.

Corpus Lexicon Diversity

Miao et al. [40] introduced a concept of Corpus Lexicon Diversity (CLD) for any corpus which measures the lexical diversity of a constituent word problem with other word problems in a corpus. CLD is calculated by computing the similarity scores between two word problem using the BLEU [79] measure. We implemented a tool for evaluating CLD using the method detailed in [40] (As the paper did not provide any code for CLD computation). The code can be found at [1]. We computed the CLD scores for all the datasets used in our experiments. For this, the

[1] https://github.com/Pruthwik/nlu-asdiv-dataset/

text in each word problem is normalized where the names of persons and quantities are replaced by a meta symbols "PERSON" and "NUMBER". The example given below demonstrates the normalization process.

- Original Text: *Ellen has 6 more balls than Marin. Marin has 9 balls. How many balls does Ellen have?*
- Normalized Text: *PERSON has NUMBER more balls than PERSON. PERSON has NUMBER balls. How many balls does PERSON have?*

CLD For Datasets

Dataset	CLD
ASDiv	0.71
ASDiv + Lexical Aug	0.32
ASDiv + Translation	0.63
ASDiv + Lexical Aug + Sampled Translation	0.41
ASDiv + Lexical Aug + Full Translation	0.44

Table 7.13: CLD Scores of Different Datasets

As diversity increases in a training dataset, the equation accuracy of the solver trained on that dataset decreases. MAWPS dataset has a lower CLD score of 0.254. Hence, the equation accuracy of any solver trained on it is very high, where the state-of-the-art accuracy is 88.7 [80]. On the contrary, ASDiv is richly diverse with a CLD score of 0.71. This impacts the equation accuracy as shown in table 7.8. The maximum accuracy obtained on this dataset is 49.96. We can also observe that data augmentation positively impacts the performance of a solver. Lexical augmentation using paraphrase table and additional constraints generate less diverse word problems as shown in table 7.13 and records a huge drop in CLD scores. The augmentation of translation adds different types of problems to the original dataset and does not result in a heavy drop in CLD. Each of the augmentations improves the solver. Training on a dataset by combining the generated problems using both the augmentation methods results in creating the best model for word problem solving.

7.5 Conclusion

In this chapter, we discussed the need for augmentation techniques and also proposed two techniques to augment any word problem dataset. We also experimentally show that the augmentation method helps in the prediction of equations on an unseen dataset. We show empirically that there is hardly any difference in the generation of equations with different notations. We created a dataset by combining the outputs from two methods to create an augmented

dataset with a significantly higher number of word problems and varied equation templates. This improved the performance overall. Apart from lexical augmentation, translation can be used as an effective method for data augmentation in low resource settings. We will discuss this in detail in the next chapter.

Chapter 8

Word Problem Solving in Indian Languages

Although the field of word problem solving (WPS) has gained traction recently, all the attention is centered only around English. Other than English, there has been a surge in developing solvers for Chinese word problems [18, 36]. Efforts in this direction are far less for other languages.

8.1 Current State of WPS in Indian Languages (ILs)

There have been very few attempts [33, 81] in ILs for arithmetic word problem solving. As Indian languages are linguistically divergent from English, how existing technologies handle word problems in ILs is another motivation for this work. We hope that our efforts towards developing automatic solvers will spur research in this area.

8.2 Data Creation

Development of recent deep neural models [27], [47] have created new benchmarks in machine translation, eclipsing the previous state-of-the-art (SOTA) performance for many language pairs. However, these models require a significant amount of parallel data to train. Many Indian languages (IL) are resource poor in this regard. In the last decade, multiple initiatives [82], [83], [84] were undertaken to change this landscape of parallel corpora for ILs. Many SOTA NMT systems [85], [84], [86] have been developed across English-IL pairs and different IL pairs. We leverage one such system [87] to translate word problems from English into Hindi. This NMT system achieves **36.7** BLEU [79] score on the general domain. We then apply already defined WPS approaches to the translated Hindi problems and evaluate the performance. Initial experiments were done by translating the Common Core dataset in English into Hindi.

8.2.1 Issues of Translations

1. English: *George was working as a sacker at a grocery store where he made 5 dollars an hour. On Monday he worked 7 hours and on Tuesday he worked 2 hours. How much money did George make in those two days?*
   Hindi: जॉर्ज एक किराने की दुकान में काम करता था जहाँ उसने पाँच डॉलर प्रति घंटा कमाए । सोमवार को उन्होंने 7 घंटे काम किया और मंगलवार को उन्होंने 2 घंटे काम किया । उन दो दिनों में जॉर्ज ने कितना पैसा कमाया ?
   Gloss in Roman: *jorj ek kiraane kee dukaan mein kaam karata tha jahaan usane paanch dolar prati ghanta kamae . somavaar ko unhonne 7 ghante kaam kiya aur mangalavaar ko unhonne 2 ghante kaam kiya . un do dinon mein jorj ne kitana paisa kamaaya ?*

2. English: *Sarah baked 38 cupcakes for her school's bake sale. If her brother, Todd, ate 14 of them how many packages could she make if she put 8 cupcake in each package?*
   Hindi: सारा ने अपने स्कूलों की बेक सेल के लिए 38 केक बेक किए । यदि उसका भाई , टॉड , उनमें से १४ को खा ले , तो वह कितने पैकेट बना सकती थी यदि वह हर पैकेट में ८ कप केक डाले ?
   Gloss in Roman: *saara ne apane skoolon kee bek sel ke lie 38 kek bek kie . yadi usaka bhaee , tod , unamen se 14 ko kha le , to vah kitane paiket bana sakatee thee yadi vah har paiket mein 8 kap kek daale ?*

3. English: *For Halloween Emily received 54 pieces of candy. She ate 33 pieces then placed the rest into piles with 7 in each pile. How many piles could she make?*
   Hindi: हेलोवीन के लिए एमिली को कैंडी के 54 टुकड़े मिले । उसने 33 टुकड़े खाए और फिर प्रत्येक ढेर में 7 के साथ ढेर में रखा । वह कितने ढेर बना सकती थी ?
   Gloss in Roman: *heloveen ke lie emilee ko kaindee ke 54 tukade mile . usane 33 tukade khae aur phir pratyek dher mein 7 ke saath dher mein rakha . vah kitane dher bana sakatee thee ?*

The first two examples show inconsistency in the translation of numbers. In the first example, the first quantity got translated into its word equivalent while the other numeric values of the other quantities are intact during translation. The numeric value of the first quantity in the second example was transferred correctly during translation, but the values of other quantities underwent a change in script when translated into Hindi. To mitigate this inconsistency, we designed a post-processing tool (presented in chapter 9), which will convert any quantity in word format or native script into its corresponding numeric value in ASCII.

The third translation is jarring due to the ambiguity arising from the sense of the word 'pile'. In the last example, the underlined phrase was not translated into the target. In order to solve this issue, we need to train the MT systems with sentences that contain words with multiple senses.

For removing noisy translations, we have incorporated various filtering techniques. These steps include correct named entity transfer, correct number/constant transfer, sampled validation by humans.

8.3 WPS Approaches

Following a similar approach to English, we implemented two deep neural models for solving Hindi word problems.

- BiLSTM with Global Attention [31]
- Transformers [27]

All these models were implemented using the Open NMT toolkit [66].

8.3.1 Preprocessing

For preprocessing, we used an in-house built tokenizer [1] for splitting the sentences in a word problem and words present in a sentence. We used two kinds of pre-trained embeddings for Hindi wps -
1. word [88] 2. subword [67]
After the first round of tokenization for words, we used the 'bpemb' python library [2] for subword tokenization [67]. Subword tokens are created using byte pair encoding on Hindi wiki articles. For subword tokenization for Hindi, the vocabulary size is **200000**.

Preprocessing of Numbers

As seen in the earlier chapters, the numbers in word problems are substituted by special number tokens for better representation learning. In [89] and [90], we substituted any number appearing in the problem text with one special token from the set {num1,num2,num3}. But *numi* is split into *num* and 0 as subwords where 0 represents any single digit number. In order to stop the splitting of a word during subword tokenization, the numbers are substituted with single-letter characters like p, q, r, and the mapping of this replacement is preserved.

8.3.2 Data Details

We translated 600 problems from the common core dataset [10] and used the translated Hindi word problems for our experiments. The corpus details for the experiments are given in table 8.1.

[1] `https://github.com/Pruthwik/Tokenizer_for_Indian_Languages`
[2] `https://github.com/bheinzerling/bpemb`

Type of Split	#Word_Problems
Train	400
Validation	100
Test	100
Total	**600**

Table 8.1: Corpus Details of Hindi Word Problems

8.3.3 Equation Representations

Equations can be represented in 3 different notations. Different notations for any binary operation consisting of two operands a, b, and one operator 'op' are shown below.

1. Infix - *a op b*

2. Prefix - *op a b*

3. Postfix - *a b op*

Griffitth et al. (2019) [24], Griffitth et al. (2020) [25] showed that learning prefix and postfix notations for an equation is easier than learning the infix notation while generating equations for English word problems. We also designed different settings for three notation types to verify this claim. One example is shown below:

- Original Question: डेव को स्कूल से पहले 29 छोटी स्लीव शर्ट और 11 लंबी स्लीव शर्ट धोनी थी । स्कूल शुरू होने तक अगर वह उनमें से 35 को ही धो देता तो कितने नहीं धोते ?
  Gloss in Roman: *dev ko skool se pahale 29 chhotee sleev shart aur 11 lambee sleev shart dhonee thee . skool shuroo hone tak agar vah unamen se 35 ko hee dho deta to kitane nahin dhote ?*

- Question After Special Number Token Replacement: डेव को स्कूल से पहले q छोटी स्लीव शर्ट और p लंबी स्लीव शर्ट धोनी थी । स्कूल शुरू होने तक अगर वह उनमें से r को ही धो देता तो कितने नहीं धोते ?
  Gloss in Roman: *dev ko skool se pahale q chhotee sleev shart aur p lambee sleev shart dhonee thee . skool shuroo hone tak agar vah unamen se r ko hee dho deta to kitane nahin dhote ?*

- Infix Equation: $X = ((q + p) - r)$

- Prefix Equation: $X = - + q\ p\ r$

- Infix Equation: $X = q\ p + r -$

parameter	value
Word Embedding Size	300
Subword Embedding Size	300
Encoder Layers	2
Decoder Layers	2
Input Sequence Length	200
Output Sequence Length	200
Dropout Rate	0.3
Batch Size	64
Optimizer	Adam

Table 8.2: Configuration of BiLSTM model with Global Attention for Hindi

parameter	value
Subword Ebedding Size	300
Encoder Layers	4
Decoder Layers	4
Heads	4
Input Sequence Length	200
Output Sequence Length	200
Nodes in Feed forward layer	2048
Dropout Rate	0.1
Attention Dropout Rate	0.1
Batch Size	512
Optimizer	Adam

Table 8.3: Configuration of Transformer Model for Hindi

8.3.4 Experimental Details

We used the same configuration as was used for English experiements. We implemented two models which are detailed in Tables 8.2 and 8.3. The word and subword vectors are of 300 dimensions.

8.4 Evaluation

Initial experiments were done using word as well as subword embeddings with the BiLSTM models. As results shown in Table 8.5 with subword tokenization were superior, we conducted further experiments using different equation notations with subword embeddings only that are shown in Table 8.4. The validation steps were set to 100.

Model	Notation	Train Steps	Equation Accuracy
BiLSTM	Infix	1500	73
		2000	77
		2500	78
	Postfix	1500	69
		2000	69
		2500	69
	Prefix	1500	76
		2000	77
		2500	79
Transformer	Infix	100	46
		400	100
		500	32
		1500	0
		2000	24
		2500	23
	Postfix	200	99
		400	100
		1500	99
		2000	100
		2500	100
	Prefix	400	99
		1000	99
		1500	99
		2000	99
		2500	99

Table 8.4: Hindi Word Problem Solver Accuracies for Different Equation Notations

8.5 Observations

We observed that the transformer model comprehensively outperforms the BiLSTM model, as shown in Table 8.4. The BiLSTM based solver has similar accuracies for the infix and prefix equations with the postfix one performing the worst. The transformer model reaches the optimal accuracies quickly in smaller number of training steps and with increase in training steps the performance of the solver does not degrade in case of prefix and postfix notations. But in the case of infix notations, the transformer model overfits very early. With the increase in training steps, the model starts making errors with respect to parentheses balancing. This phenomenon was also reported in Griffitth et al. (2019) [24], Griffitth et al. (2020) [25]. As the prefix and postfix notation is devoid of parentheses, the errors related to them do not arise. The training data contains word problems with only relevant quantities (3 numbers in each problem). So, any question containing an irrelevant quantity can not be solved by the solver as it has only

Model	Emb Type	Train Steps	Validation Steps	Equation Accuracy
BiLSTM	word	1000	100	54
	subword	1000	100	63

Table 8.5: Hindi Word Problem Solver Accuracies with word and subword embeddings

learnt the representation in the presence 3 quantities. We also observed the dependence of solvers on shallow heuristics as shown by Patel et al. [28] where the solvers generated the equation without looking at the question text. These results were also overestimated because of high overlapping between the training and the evaluation data.

8.6 Data Augmentation

Similar to the one described in Chapter 7, we developed an augmentation technique for Hindi. As paraphrase tables were not publicly available for Hindi, we could not use the same method for Hindi. Nevertheless, we augmented the data by using word2vec pre-trained embeddings and gazetteers for person names enriched with a language model. We measured the CLD for each of the datasets as well. As the Hindi translations of the common core data had a very high level of overlapping, we discarded that data and used a Hindi dataset which has more diversity.

8.6.1 Distributional Semantics For Content Words

Similar to the augmentation approach followed for English, we augment an existing Hindi word problems dataset [33] by substituting the content words. For this, we replaced only nouns and verbs to generate new word problems. We utilized an in-house developed tokenizer [3] for tokenizing the sentences and words in the Hindi word problems. For POS tagging and other shallow parsing tasks, we developed a shallow parser [91]. For any noun or verb in a sentence, we first find three most similar words to it. These words are obtained by using a Hindi word2vec model [73]. We used pre-trained word2vec embedding [92] that was trained in house on Hindi news articles consisting of 5 million sentences. We also put an additional constraint that the similarity between the original word and its similar word should be above or equal to a threshold. The threshold was set to 0.6. Any word which does not fulfill this criterion was discarded.

8.6.2 Constraints of Linguistic Features

In order to maximize the lexical diversity, we imposed a constraint that different morphological forms of the same root should be filtered out. If a word and one of its similar words share the root, then the similar word is deemed to be invalid for substitution. This is a useful feature

[3] `https://github.com/Pruthwik/Tokenizer_for_Indian_Languages`

as Indian languages are morphologically richer and a single root can have different morph forms. Another constraint was added to match the gender with the candidate word. This restriction is applied to restrain the generation of ungrammatical sentences due to gender mismatch. All these linguistic properties are extracted using our in-house shallow parser [91].

8.6.3 Language Model

Instead of replacing a word with its distributionally similar word, a check for compatibility with its neighborhood in a sentence is verified. We utilize a pre-trained language model for it. We used a KENLM language model for Hindi [4]. The size of the context window was 5 for the language model. We use the same threshold as defined in Section 7.2.2.2 for checking the validity of the 5-gram after substituting a word with a similar word.

8.6.4 Gazetteer List

Two gazetteer lists are also prepared to replace different types of named entities, such as male names and female names, in Hindi. As person names share similar contexts irrespective of gender, the distributional similar contexts of a male person name can include female person names. To avoid this, these lists were developed.

8.6.5 Number of Substitutions

Instead of generating word problems with single word substitutions, we fixed a threshold on number of substitutions to count a generated problem to be valid. The threshold is set to 3 for generating Hindi word problems.

8.6.6 Maximum Generations per Word Problems

To control over the generation, the maximum number of word problems generated per word problem is kept at 5 after satisfying all these constraints.

8.6.7 Examples of Data Augmentation

The suggested augmentation method generates many valid word problems. For each question, we present two generated examples here.

[4] https://github.com/Open-Speech-EkStep/vakyansh-models

Valid Generation

- Original: शिखा ने 1 दुकान से 65 रु . का सामान खरीदा । उसने दुकानदार को 100 रु . का नोट दिया । बताओ , उसे कितने रुपये वापिस मिले ?

  Gloss in Roman: shikha ne 1 dukaan se 65 ru . ka saamaan khareeda . usane dukaanadaar ko 100 ru . ka not diya . batao , use kitane rupaye vaapis mile ?

  Generated 1: सलमा ने 1 दूकान से 65 रूपए . का बैग ख़रीदा । उसने व्यापारी को 100 रूपए . का नोट दिया । बताओ , उसे कितने रुपये वापिस मिले ?

  Gloss in Roman: salama ne 1 dookaan se 65 roope . ka baig khareeda . usane vyaapaaree ko 100 roope . ka not diya . batao , use kitane rupaye vaapis mile ?

  Generated 2: वेंडी ने 1 दूकान से 65 रूपए . का माल मंगाया । उसने व्यापारी को 100 रूपए . का नोट दिया । बताओ , उसे कितने रुपये वापिस मिले ?

  Gloss in Roman: vendee ne 1 dookaan se 65 roope . ka maal mangaaya . usane vyaapaaree ko 100 roope . ka not diya . batao , use kitane rupaye vaapis mile ?

- Original: टॉम के पास पहले से ही 2 गेम थे लेकिन उसने ₹13.6 का बैटमैन गेम और ₹5.06 का सुपरमैन गेम खरीदा । सुपरमैन गेम की तुलना में बैटमैन गेम कितना महंगा था ?

  Gloss in Roman: tom ke paas pahale se hee 2 gem the lekin usane ₹13.6 ka baitamain gem aur ₹5.06 ka suparamain gem khareeda . suparamain gem kee tulana mein baitamain gem kitana mahanga tha ?

  Generated 1: ओंकार के पास पहले से ही 2 गेम थे लेकिन उसने ₹13.6 का स्पैरो गेम और ₹5.06 का सुपरमैन गेम ख़रीदा । सुपरमैन गेम की तुलना में स्पैरो गेम कितना महंगा था ?

  Gloss in Roman: onkaar ke paas pahale se hee 2 gem the lekin usane ₹13.6 ka spairo gem aur ₹5.06 ka suparamain gem khareeda . suparamain gem kee tulana mein spairo gem kitana mahanga tha ?

  Generated 2: फ्रैंकलिन के पास पहले से ही 2 गेम्स थे लेकिन उसने ₹13.6 का स्पाइडरमैन गेम्स और ₹5.06 का सुपरमैन गेम्स मंगाया । सुपरमैन गेम्स की तुलना में स्पाइडरमैन गेम्स कितना महंगा था ?

  Gloss in Roman: phrainkalin ke paas pahale se hee 2 gems the lekin usane ₹13.6 ka spaidara–main gems aur ₹5.06 ka suparamain gems mangaaya . suparamain gems kee tulana mein spaidaramain gems kitana mahanga tha ?

Noisy Generations

Even in the presence of constraints, some substitutions using word embeddings are jarring and unnatural.

- Original: मार्था के पास 76 कार्ड थे । उसने एमिली को 3 दिए । मार्था के पास कितने कार्ड बचे हैं ?
  Gloss in Roman: maartha ke paas 76 kaard the . usane emilee ko 3 die . maartha ke paas kitane kaard bache hain ?
  Generated: केंड्रा के पास 76 कार्ड थे । उसने लीला को 3 किए । केंड्रा के पास कितने कार्ड बचे हैं ?
  Gloss in Roman: kendra ke paas 76 kaard the . usane leela ko 3 kie . kendra ke paas kitane kaard bache hain ?

- Original: 1 व्यक्ति के पास 63 भेड़ें है । उसने अपने 3 बच्चों में उनका बँटवारा किया । पहले बेटे को 22 भेड़ें मिलीं और दूसरे बेटे को 21 भेड़ें मिलीं , तो तीसरे बेटे को कितनी भेड़ें मिलीं ?
  Gloss in Roman: 1 vyakti ke paas 63 bheden hai . usane apane 3 bachchon mein unaka bantavaara kiya . pahale bete ko 22 bheden mileen aur doosare bete ko 21 bheden mileen , to teesare bete ko kitanee bheden mileen ?
  Generated: 1 व्यक्ति के पास 63 भेड़ें है । उसने अपने 4 देखभाल में उनका बटवारा किया । पहले बेटे को 22 भेड़ें मिलीं और दूसरे बेटे को 21 भेड़ें मिलीं , तो तीसरे बेटे को कितनी भेड़ें मिलीं ?
  Gloss in Roman: 1 vyakti ke paas 63 bheden hai . usane apane 4 dekhabhaal mein unaka batavaara kiya . pahale bete ko 22 bheden mileen aur doosare bete ko 21 bheden mileen , to teesare bete ko kitanee bheden mileen ?

The underlined words are jarring substitutions using word embeddings. In case of English, these jarring substitutions are avoided using the synonym similarities using the wordnet. As IndoWordnet [93] does not have this facility, we could not include this constraint in our generation.

8.7 Experimental Setup

As explained in chapter 7, 5 fold stratified cross-validation was conducted while designing the models. The stratified sampling was done on the number of operations required to solve a problem. Infix, postfix, and prefix equation notations are also experimented with. We used an encoder-decoder model with pre-trained multilingual BERT embeddings to fine-tune on Hindi datasets. XLM-RoBERTa-base [94] is used for this. There are 270 million parameters in the XLM-RoBERTa-base model, which has 12 encoder and 12 decoder layers with 12 attention

heads. We froze the encoder layers so that the gradient does not get updated for these lower layers during backpropagation. This reduced the number of parameters by half and also minimized the risk of overfitting. Most of the models were trained for 50 epochs. This model is trained in 100 languages that include the three languages: English, Hindi, and Telugu for this study. Some of the models achieved better performance in fewer epochs. We used one NVIDIA GeForce GTX 1080 Ti GPU having 11GB of RAM using 6 CPUs for the experiments.

Preprocessing

The relevant numbers in a word problems were replaced by meta tokens such as number0, number1, number2, .., numberi. One example depicts this process.

- Original Text: स्पर्स बास्केटबॉल टीम में 22 खिलाड़ी हैं। प्रत्येक खिलाड़ी के पास 11 बास्केटबॉलें हैं। उन सब के पास कुल कितनी बास्केटबॉलें हैं?

- Text Replaced by Numeric Meta Symbols: स्पर्स बास्केटबॉल टीम में number0 खिलाड़ी हैं। प्रत्येक खिलाड़ी के पास number1 बास्केटबॉलें हैं। उन सब के पास कुल कितनी बास्केटबॉलें हैं?

- English Gloss: Spurs basketball team has number0 players. Each player has number1 basketballs. How many basketballs are there with them?

- Equation Notations:

  - Infix: (number0 + number1)

  - Postfix: number0 number1 +

  - Prefix: + number0 number1

Dataset

The dataset distribution is shown in table 8.6. The original dataset consists of 2336 word problems where the number of single operation word problems is thrice the number of word problems that require two operations to solve. To balance out this unevenness regarding the number of operations, by using undersampling we created a more balanced dataset to reduce this ratio to 2: 1. In the non augmented dataset, the ratio of single operation word problems to double operation word problems is 2: 1 whereas it reduced to 1.57: 1 in the augmented dataset.

config	No_of_Operation	#Word_Problems
No Augmentation	1	860
	2	430
	Total	**1290**
With Augmentation	1	3318
	2	2108
	Total	**5426**

Table 8.6: Distribution of Full Augmented Dataset in Hindi

8.8 Results and Observation

Config	Notation	Equation_Accuracy	CLD
No Augmentation	Infix	34.81	0.648
	Postfix	38.53	
	Prefix	35.64	
With Augmentation	Infix	42.06	0.194
	Postfix	43.39	
	Prefix	44.57	

Table 8.7: 5 Fold Results For Hindi

A similar trend is also seen in Hindi. With the decrease in CLD, equation accuracy goes up. With the increase in dataset size, there exists no or very little difference in the generation of different kinds of equation notations.

Issues with Number Conversion

Most of the current solvers replace the numeric quantities appearing in a word problem with generic meta symbols to learn better representations for the numbers. This indeed improves the performance of the solver while generating equations during the decoding phase in an end-to-end neural network based system. This is a very efficient pre-processing step. However, many

word problems require the numeric values of the operands to derive the solution. Let us look at an example.

- Original Question: *There are two kinds of tickets for a circus. One is priced at 48 rupees and the other is priced at 80 rupees. There are 180 tickets for 48 rupees and 120 tickets for 80 rupees. A total of 260 tickets were sold. What is the maximum possible revenue that can be earned?*
- Equation With Numeric Values: $((80 * 120) + (48 * (260 - 120)))$
- Question With Meta Symbol Replacement: *There are two kinds of tickets for a circus. One is priced at number0 rupees and the other is priced at number1 rupees. There are number2 tickets for number0 rupees and number3 tickets for number1 rupees. A total of number4 tickets were sold. What is the maximum possible revenue that can be earned?*
- Equation With Meta Symbols: ((number1*number3)+(number0*(number4-number3)))

In this example, the information that $number1 > number0$ and $number4 > number3$ is very crucial to solve the problem. But there are no clues explicitly present in the question about this fact. This makes solving these kinds of problems challenging.

8.9 Augmentation of Data with Translation

We have briefly discussed this technique in Chapter 7. We developed an initial Hindi solver on the Hindi translations of the common core (CC) dataset. The performance of the solver trained on a subset of this dataset obtained 100% equation accuracy. The CC dataset has a very low corpus lexicon diversity score (CLD) of **0.004**. These perfect scores are attributed to very high overlapping validation and test datasets as well as the overall CLD score. The next Hindi solver was developed on a richly diverse corpus with CLD of 0.65. The solvers trained on the dataset obtained a maximum of 45% equation accuracy with lexical augmentation. To explore further effects of data augmentation on Indian language solvers, we augment the available datasets with translations. We chose two Indian languages: *Hindi* and *Telugu* for this experiment. Hindi was selected as a few resources are available. Telugu is an extremely low resource language for WPS as it does not have any word problem solving dataset for it. For creating datasets in these two languages, the English translations of Ape210K are translated into Hindi and Telugu using [87] transformer based MT systems [5]. We added translated data with the available Hindi data to train a solver. For Telugu, this is the first word problem dataset and automatic solver. We extended the English dataset as described in Chapter 7 and trained a model with a larger and more diverse dataset. We also developed benchmark datasets

[5]HimangY Translate: `https://ssmt.iiit.ac.in/translate`

in English, Hindi, and Telugu. This is the first attempt to create a multilingual benchmark parallel corpora including Indian languages in this domain.

8.10 Creation of Benchmark Datasets

The creation of quality benchmark datasets with diverse as well as adequate samples is crucial for testing the robustness of developed models. We studied word problem datasets from different resources such as school books, educational websites, blogs, and forums available for various languages. Taking inspiration from those, we manually developed benchmark datasets for English and a few Indian languages such as Hindi and Telugu. These benchmark datasets contain word problems that will adhere to the properties laid out by Patel et al. [28] and Mio et al. [40]. In this work, we release a 3-way parallel word problem dataset in English, Hindi, and Telugu created manually. 4.25% (2000 problems) of the English translations did not have equations; they directly contained the answer without involving the intermediate operations. We discarded those. As described in chapter 7, this dataset is composed of both explicit and implicit word problems. For this study, we restrict ourselves to explicit problems.

8.10.1 Method of Creation

The corpus is created following a two-step process. First, we translated the validation set of the Ape210K dataset into English using Google Translate API. The validation set consists of 5000 word problems. We picked 1127 word problems and manually transcreated them after applying noise removal techniques as explained in chapter 7. If the equations were erroneous, they were rectified. In case the word problem is unintelligible, then we newly form a word problem taking the equation as a basis. In the second stage, the verified English word problems are translated into Hindi and Telugu manually. For each language, two experts were selected. All the experts had completed their post graduation level of education. They had domain knowledge in mathematics as well. The Hindi and Telugu translators have more than five years of experience in translation and have previously completed translation assignments from diverse domains. For all these tasks, we used an in-house developed AI enabled post editing tool [95]. A snapshot of the tool is shown in figures 8.1 and 8.2.

8.10.1.1 English Word Problem Transcreation

The experts were given guidelines to create English word problems adhering to grammatical correctness and ensuring naturalness with solvable equations. The guidelines below were followed, and different examples were provided for a better understanding of the task.

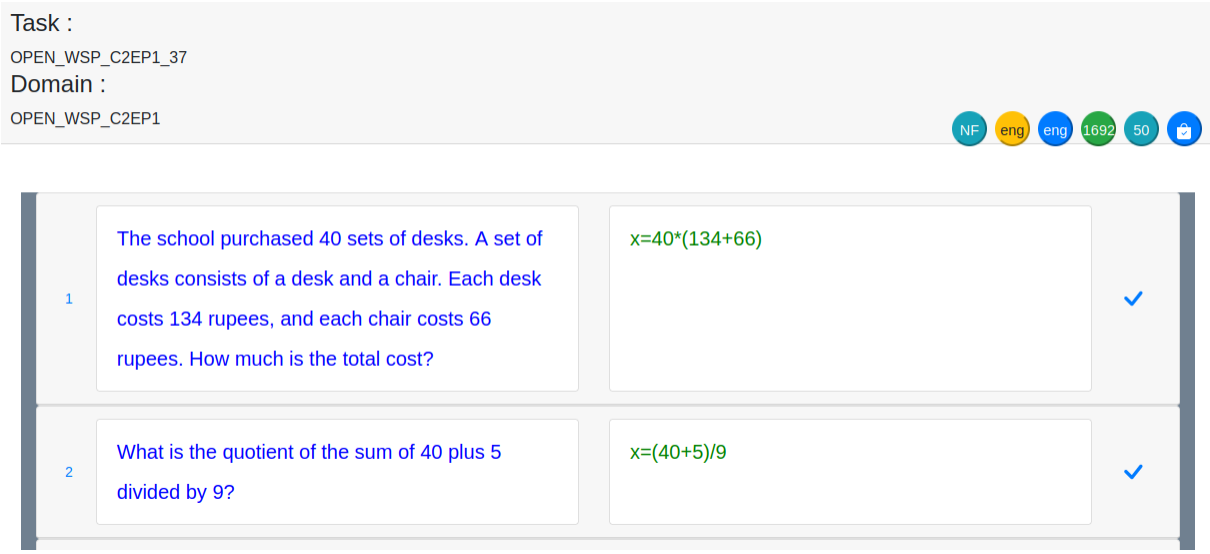

Figure 8.1: English Word Problem Editing

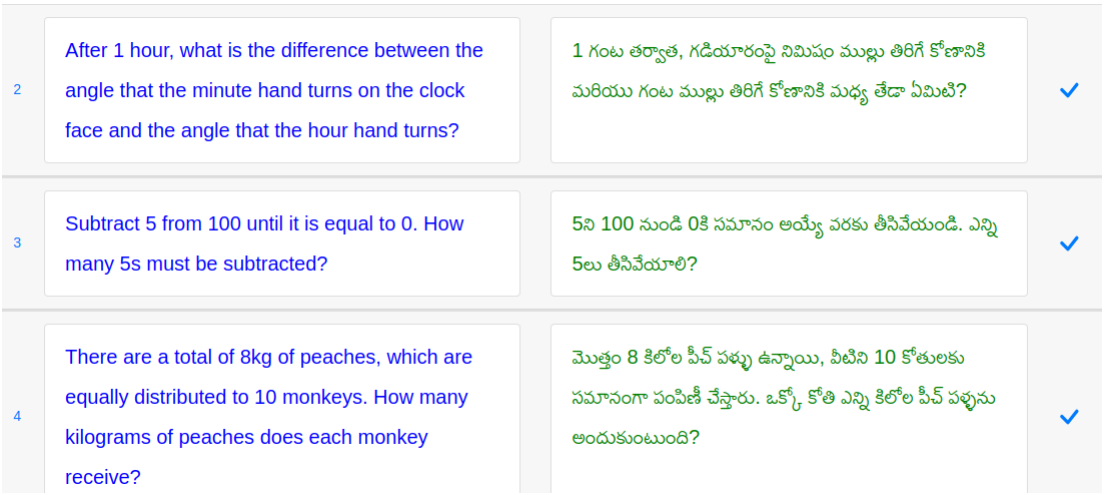

Figure 8.2: English to Telugu Manual Translation of Word Problems

Correction of Determiners

In many translations, there was incorrect usage of determiner *a* and *an*. The article *the* was heavily used with gross mistakes.

- Original: *The fruit store brought 125 boxes of apples, 20 kilograms each, and sold 1,600 kilograms. How many kilograms are left?*

- Corrected: *A fruit store brought 125 boxes of apples, 20 kilograms each, and sold 1,600 kilograms. How many kilograms are left?*

Simplify Long Sentences

If there are very long sentences, simplify the text by splitting them into smaller sentences.

- Original: *The grain store warehouse has 305 bags of flour, and the number of bags of rice is 12 times the number of bags of flour, how many bags of rice are there?.*
- Simplified: *The grain store warehouse has 305 bags of flour. The number of bags of rice is 12 times the number of bags of flour. How many bags of rice are there?*

Naturalness

If the text does not sound natural in English, add phrases or words to make the text more natural and better fit in the narrative. Similar guidelines were also followed in [33]. When fractions are used in word problems, the English translations go awry with many instances of missing words. In this example, the equation also acts as a support for the choice of words or a chunk of words.

- Original: *There are 32 poplar trees in the orchard, less than pear trees (1/9), how many pear trees are there?*
- Equation: *32/(1-(1/9))*
- Corrected: *There are 32 poplar trees in an orchard. The number of poplar trees is (1/9) times less than that of the number of pear trees. How many pear trees are there?*

Localisation

As the word problems in the benchmark corpora are developed in the Indian context, localization in terms of currencies, names of persons, locations, organizations, and other named entities are incorporated. To make the word problems more realistic, equations were also modified to reflect the changes made in numeric values instead of original numbers.

- Original: *In the donation activity for the disaster area , Class 61 donated 245 yuan , and Class 62 donated 45 yuan less than twice that of Class 1 . How much did the two classes donate in total ?*
- Localised: *In a donation activity for the disaster areas in Assam , a school collected some money from students and teachers . Class 6 donated 2450 rupees and class 5 donated 500 rupees less than twice that of class 6 . How much did the two classes donate in total ?*
- Original Equation: $245 + (245 * 2 - 45)$
- Equation After Changing Values According to Localisation: $2450 + ((2450 * 2) - 500)$

We also followed peer review process between the experts to ensure consistency and quality in the created word problems and equations.

8.10.1.2 Translation of English Word Problems

After the completion of the creation of English word problems, we switched to developing corresponding translations in Hindi and Telugu. As shown in the previous sections, this manual task was also carried out using the same post-editing tool [95]. The translators were directed to focus on both adequacy and fluency while translating into target languages. The following guidelines were given to them.

- Adequate: The translated word problems should convey the same meaning as the English equivalent.
- Fluent, and Natural: It should be natural and fluent in the target languages. The target audience for the word problem dataset is the students in respective vernacular. They should be capable of understanding and solving the problems.
- Transliteration: As we are dealing with arithmetic and geometry word problems, many of the English mathematical terms need to be transliterated into the target language. We took this decision based on the arithmetic books available in Hindi and Telugu. Words such as *cylinder*, *rhombus*, *cone* etc. are transliterated. If the terms find an equivalent in the target, then these words take precedence over the transliteration.

A similar peer review process between the experts was followed in translation.

8.11 Experimental Setup for Full Augmentation

For training the solver, we adopted the full augmentation strategy similar to that of English.

8.11.1 Dataset

We combine the lexically augmented data and translated data with the publicly available HAWP [33]. As there are no datasets available for Telugu, we use only the Telugu translations of the English explicit word problems 7.2.

Preprocessing

As we are using the direct outputs of MT systems, we added some noise removal techniques for English-Hindi and English-Telugu translations. After translation, we match the numeric quantities both in the source and target. If they do not tally, then we remove the translation.

Ape210K contains many problems that require simple calculation of a formulaic expression. While translating these from English to Hindi or Telugu, either numbers or operations go missing. This poses an additional challenge to the solver if we do not filter it out because of missing information. For this task, we are doing a two step translation using MT where the first step translates a Chinese word problem into English. The next step is to translate this English problem into Hindi or Telugu. This introduces cascading of errors. Let us look at a few examples.

- Telugu Example
  - English Translation: *The number A is 500, and the number B is 780 less than 5 times it. What is the number B?*
  - Telugu Translation: ఎ సంఖ్య 500, మరియు బి సంఖ్య 5 రెట్లు తక్కువ. సంఖ్య b అంటే ఏమిటి ?
  - Gloss in Roman: E sankhya 500, mariyu bi sankhya 5 retlu takkuva Sa□khya b ante emiti?
  - Number Missing: 780
- Hindi Example
  - English Translation: *There are 125 apple trees in an orchard, and the number of pear trees is 20 less than 4 times that of apple trees. How many trees are there in this orchard?*
  - Hindi Translation: एक बाग में 125 सेब के पेड़ हैं, और नाशपाती के पेड़ की संख्या सेब के पेड़ों की तुलना में 4 गुना कम है।इस बाग में कितने पेड़ हैं ?
  - Gloss in Roman: ek baag mein 125 seb ke ped hain, aur naashapaatee ke ped kee sankhya seb ke pedon kee tulana mein 4 guna kam hai.is baag mein kitane ped hain ?
  - Number Missing: 20

After this step, we had 9288 Hindi translated word problems and 9311 translated word problems in Telugu.

Language	**#Train**	**#Validation**	**#Test**
Hindi	13242	1472	1127
Telugu	8379	932	1127

Table 8.8: Dataset Details for Full Augmentation in Hindi and Telugu

The dataset details are present in table 8.8. Similar to the English dataset, 90% of the problems in both languages require 1 or 2 operations to solve. The CLD scores of the benchmark

datasets in English, Hindi, and Telugu are **0.706**, **0.764**, **0.777** respectively. For calculating the CLD, we used Indic NER [96]. Out of the 1127 problems from the benchmark datasets, only **432** are explicit in nature. We report the accuracies on the explicit datasets.

8.11.2 Experiments and Results

We use multilingual XLM RoBERTa base for our experiments. As the English solver built in chapter 7 was based on XLM RoBERTa, the initial experiments are designed to test the zero shot capability of this multilingual model. XLM RoBERTa-base supports 100 languages including Hindi and Telugu. All the results are reported in two settings: **Full** that reports the accuracy on the whole dataset with one, two, greater than two operations and 1+2op only reports accuracies of questions with 1 and 2 operations. The results are shown in Tables 8.9, 8.11, and 8.10.

Lang	**Type**	**Infix**		**Postfix**		**Prefix**	
		Full	**1+2op**	**Full**	**1+2op**	**Full**	**1+2op**
Hindi	Val	26.4	30.2	30.6	35.4	24.8	28.8
	Test	19.2	26.0	19.7	26.8	18.2	24.6
Telugu	Val	23.4	33.1	19.8	28.5	20.2	28.8
	Test	20.1	26.8	13.7	18.3	19.0	25.7

Table 8.9: Zero Shot Experiment Results for Hindi and Telugu. Lang stands for language and Type stands for type of the evaluation data. Full represents the accuracies on datasets containing 1, 2, and >2 operations while 1+2op represents the accuracies on datasets containing 1 and 2 operations.

The zero-shot learners rather fare poorly on both datasets. But the accuracy of predicting the equation correctly for a single operation is 40% on average. This shows that the model's performance gradually degrades with the increase in the number of operations. As a next step, we fine tuned the English solver on Hindi and Telugu word problems. The fine tuning was done for 20 epochs with the same parameters and settings as described above. We also fine-tuned on Hindi and Telugu data directly, but the results were inferior to fine-tuning an English solver.

Lang	Type	Infix		Postfix		Prefix	
		Full	**1+2op**	**Full**	**1+2op**	**Full**	**1+2op**
Hindi	Val	55.2	60.9	51.8	58.0	52.4	58.9
	Test	26.5	32.1	25.1	31.5	23.0	28.9
Telugu	Val	39.5	47.3	35.1	43.8	32.9	41.1
	Test	24.2	29.8	21.6	26.6	20.4	27.3

Table 8.10: Exact Equation Accuracies on English Fine Tuned Model for Hindi and Telugu. Lang stands for language and Type stands for type of the evaluation data. Full represents the accuracies on datasets containing 1, 2, and >2 operations while 1+2op represents the accuracies on datasets containing 1 and 2 operations.

Lang	Type	Infix		Postfix		Prefix	
		Full	**1+2op**	**Full**	**1+2op**	**Full**	**1+2op**
Hindi	Val	58.4	64.1	54.5	60.8	54.6	61.4
	Test	38.2	47.6	35.01	45.1	30.8	40.1
Telugu	Val	50.0	56.7	50.3	58.5	47.7	56.0
	Test	33.4	42.4	31.8	41.0	26.3	36.7

Table 8.11: Equivalent Equation Accuracies on English Fine Tuned Model for Hindi and Telugu. Lang stands for language and Type stands for type of the evaluation data. Full represents the accuracies on datasets containing 1, 2, and >2 operations while 1+2op represents the accuracies on datasets containing 1 and 2 operations.

8.11.3 Discussion

Fine tuning on a pre-trained model leverages transfer learning to achieve optimal performance. Our experiments show that multilingual models can prove to be beneficial in case of scarcity of resources. We also observed that the Hindi solver performs well compared to Telugu as it is augmented with a human created dataset along with lexical augmentation. The noise in the translations can hinder learning which is seen in Telugu. The model does not learn to generate bigger equations with a larger number of operations. This may be attributed to fewer number of examples for 3 and 4 operation word problems. These results are very similar to the

results reported by Patel et al. [28] on the manually created SVAMP dataset. The word problem solvers, although trained on a large set of word problems, fail to solve single variable word problems and perform poorly on manually crafted datasets with diversity at both lexical and template levels. We also see a similar significance of equation equivalence for Hindi and Telugu. Exact matching penalizes the equivalence of equations and under-report the performance.

8.12 Testing ChatGPT for Word Problem Solving

We tested the capabilities of mathematical word problem solving on ChatGPT, a large language model modeled with reinforcement learning from human feedback (RLHF). For this, we took 10 examples from our dataset in English, Hindi, and Telugu.

8.12.1 Experiment For English

For English, out of the 10 samples, ChatGPT was able to solve 8 of them correctly. The following two questions were answered incorrectly by ChatGPT.

Linguistic Ambiguity

- Question: *For the first main dish, they were asked to cook steak. If the third and second team cooked 240 plates of steak and the first team cooked 75 plates less than what the second and third team made, how many steaks did they cook altogether?*
- Predicted Equation: *x = 405 - 480*
- Predicted Answer: *-75*
- Correct Equation: *x = 240 + (240 - 75)*
- Correct Answer: *405*

The model accurately identifies the subtraction operation related to the word less. But, it incorrectly assumes that the second and third teams individually make 240 plates.

Issues with a large list of numbers

- Question: *Calculate: 1 + 2 - 3 + 1000 - 1001 + 2009 - 2007 + 9002 + 1 + 2 - 3 + 1000 - 1001 + 2009 - 2007 + 9002*
- Answer: 18002

$(1+2-3)+(1000-1001)+(2009-2007)+9002+(1+2-3)+(1000-1001)+(2009-2007)+9002$

While answering this question, chatGPT groups numbers to easily calculate the values as written below. It incorrectly calculates one of the values of the groups. So, the final solution is incorrectly evaluated.

8.12.2 Experiment For Hindi

Out of the 10 examples, ChatGPT could solve 5. For one problem, it did not generate any equation and failed to understand that it is a mathematical whereas the other four problems were answered incorrectly.

Incorrect Equation Usage

- Question: राहुल, ब्लू टीम का कप्तान मैच से पहले घबरा रहा है। उनकी पहली प्रतिद्वंद्वी लाल टीम ने उन्हें हरा दिया। यदि वे 13 रन से हारे और लाल टीम ने 61 रन बनाए, तो उनकी टीम ने कितने रन बनाए थे?
- Gloss in Roman: raahul, bloo teem ka kaptaan maich se pahale ghabara raha hai. unakee pahalee pratidvandvee laal teem ne unhen hara diya. yadi ve 13 ran se haare aur laal teem ne 61 ran banae, to unakee teem ne kitane ran banae the?
- Predicted Equation: `x = 61 + 13`
- Correct Equation: `x = 61 - 13`

In this example, the word *lost* is associated with subtraction, but chatGPT identifies it to evoke an addition operation. It uses a wrong formula to arrive at the answer:
Runs scored by the Blue team = Runs scored by the Red team + Difference in runs between the teams

Incorrect Operation Identification

- Question: आर्केड में खेलते हुए फ्रैंक ने 33 टिकटें 'व्हेक ए मोल' और 9 टिकटें 'स्के बॉल' खेलकर जीतीं। यदि वह टॉफ़ियाँ खरीदने की कोशिश कर रहा था, जिनकी कीमत 6 टिकटें थी, तो वह कितनी खरीद सकता था?
- Gloss in Roman: aarked mein khelate hue phraink ne 33 tikaten vhek e mol aur 9 tikaten ske bol khelakar jeeteen. yadi vah taaaifiyaan khareedane kee koshish kar raha tha, jinakee keemat 6 tikaten thee, to vah kitanee khareed sakata tha?
- Predicted Equation: `x = (33 + 9) * 6`

- Correct Equation: x = (33 + 9) / 6

This question is a multi-step arithmetic problem. The first operation and its operands are correctly identified. In the second operation, it misinterprets that the unit of the first operation is toffees, not tickets. Hence, the second predicted operation is multiplication in case of a division.

8.12.3 Experiment For Telugu

Out of the 10 examples, ChatGPT could answer 4 correctly. 6 were answered incorrectly where it did not generate any equation for 2 problems.

Incorrect Operation Identification

- Question: మిల్లెట్ ఒక జత స్నీకర్లకు 135 యువాన్లు మరియు 85 యువాన్లను ఒక ఫుట్బాల్ కోసం ఖర్చు చేస్తుంది మరియు సేల్స్ పర్సన్ 250 యువాన్లను చెల్లిస్తుంది. సేల్స్ వ్యక్తి ఆమెను ఎంత తిరిగి పొందాలి ?
  - Gloss in Roman: Millet oka jata snikarlaku 135 yuvanlu mariyu 85 yuvanlanu oka phutbaal kosam kharcu cestundi mariyu sels parsan 250 yuvanlanu cellistundi. Sels vyakti amenu enta tirigi pondali?
  - Predicted Equation: x = 135 + 85 + 250
  - Correct Equation: x = 250 – (135 + 85)

It incorrectly identifies that the question is about total money spent and adds up all the numbers.

Missing Operands

- Question: తోటలో 38 వరుసలు ఆపిల్ చెట్లు, ఒక్కొక్క వరుసలో 25 చెట్లతో, ప్రతి చెట్టుకు సగటున 60 కిలోల ఆపిల్ లభిస్తుంది. ఈ ఆర్చర్డ్ మొత్తం ఎన్ని కిలోల ఆపిల్లను పొందవచ్చు ?
  - Roman in English: Totalo 38 varusalu apil cetlu, okkokka varusalo 25 cetlato, prati cettuku sagatuna 60 kilola apil labhistundi. i arcard mottam enni kilola apillanu pondavaccu?
  - Predicted Equation: x = 38 * 60
  - Correct Equation: x = 25 * 38 * 60

ChatGPT ignores the information about the rows (varusalu) in the question and only predicts the total number of apples (apil) from the trees (cetlu).

8.13 Conclusion

In this chapter, we discussed the development of data and approaches for WPS in ILs. We developed benchmark datasets for English, Hindi, and Telugu. We designed a solver for Telugu by augmenting it with the translations of English word problems, which provides a baseline for resource-poor languages. We also show that the availability of a human created dataset improves the solver rather than a noisy augmentation technique such as machine translation. We report the frailties of word problem solvers in solving diverse arithmetic word problems. We also tested some examples in different languages with ChatGPT and showed some of the limitations of it. It performs poorly for the Indian languages.

Chapter 9

Use of WPS Components in NLP Applications

The earlier chapters show how an arithmetic word problem can be solved using frame based and neural approaches. Arithmetic word problems have a fixed structure and require NLU to solve them. As WPS involves this step, it can be used in other scenarios, too. We explore two such scenarios.

9.1 Number Conversion in Machine Translation

Recent advances in the field of machine translation have helped different low-resource languages to be accessible in other languages. Although the quality of translation outputs across different language pairs has improved significantly with the advent of subword units [97], the translation of numbers is still challenging. If numbers are represented as their word equivalents, identification and translation into the target language becomes a daunting task. To tackle this, researchers apply different preprocessing techniques for handling numbers either by replacing them with special token identifiers like *UNK* or *NUM*. In the case of subwords, they are represented as 0+ (regular expression of 1 or more 0s) where *0* denotes any single-digit number, *00* for any two-digit number, and so on. If a sentence is tokenized at the subword level in this way, it is impossible to get back the original numbers appearing in the text. We propose a conversion technique for identifying numbers in word form for English and other Indian languages. This can be plugged into any NMT system as a preprocessing task.

9.1.1 Issues in Numeric Translation

Based on the efficiency of NMT systems, the translation of numbers varies. We used the "Google Translate" for this study in two different language pairs: English-Hindi and English-Odia and show the performance of the translation variance in the following sections.

9.1.1.1 Number Conversion

The example shown in Figure 9.1 erroneously converts the English number into Hindi and Odia. The numeric value of the number '2 million three thousand two hundred' mentioned in English is *2003200*, but the translated quantity both in Hindi and Odia has a value of *203200* which is 10 times less than the English equivalent.

English	Hindi (Roman)	Odia (Roman)
I have two million three thousand two hundred rupees.	mere paas do laakh teen hajaar do sau rupaye hain.	mora dui lakhya tini hajara dui shaha tanka achi

Figure 9.1: Incorrect Number Conversion

9.1.1.2 Conversion of Words into Numeric Values

We convert the numbers that are expressed in words into their corresponding numeric values for the source sentence. This, in turn, eliminates the problems arising due to the conversion, and the efficiency of the NMT systems increases, as shown in Figure 9.2. The developed system is available at `https://github.com/Pruthwik/word2number-convertor`.

English	Hindi (Roman)	Odia (Roman)
I have 2.31 litres of milk.	mere paas 2.31 leetar doodh hai.	mora 2.31 litar kshira achi
I have 2003200 rupees.	mere paas 2003200 rupaye hain.	mora 2003200 tanka achi

Figure 9.2: Translation of Sentences

9.2 Equation Identification and Conversion into Math Notations in Transcripts

Massive Open Online Courses (MOOCs) have opened up a diverse landscape for all to access quality education on handheld devices and have succeeded in democratizing the space of education. A person in the remotest place in India can listen to the MOOC lectures offered by Dan Jurafsky at Stanford University and learn the basics of Natural Language Processing. The last decade has witnessed rapid growth in the number of universities joining different web platforms and offering myriad courses online. Till December 2016, 700+ universities were offering 6500+ courses [98]. It has increased exponentially from then on and more so in the times of COVID-19. As the majority of the MOOCs provide content in English, there have been efforts [99, 100] to overcome this language barrier and translate content into different

languages. In India, SWAYAM and NPTEL platforms are translating educational video lectures from English into multiple Indian languages. As a huge number of lecture videos are created on a regular basis, creating manual translations in multiple languages becomes an impossible task. So, efforts are underway to develop supporting tools for Automatic Speech Recognition (ASR), Machine Translation (MT), and Text-to-Speech (TTS) in a human-in-the-loop paradigm. Our proposed solution is a post processing step after ASR before supplying it to MT systems.

9.2.1 Motivation

MOOC lectures are usually subtitled. A subtitle file consists of plain text and timelines of the text. This text is created using transcription. Transcription is the process of converting the speech in audio into written text. In simple terms, a transcript (outcome of transcription) is the textual form of what is spoken in audio. Many a times, the transcript contains plain text. Numbers and equations are written as words e.g. *A line can be represented by an equation y equals to m into x plus c.* This kind of text is cumbersome to read. It is even more challenging to understand its meaning for a course that requires mathematical symbols and equations, as shown in the example.

- English: The equation of a line is y equals m into x plus c.
- Hindi: एक रेखा का समीकरण y बराबर है m x प्लस c में ।
- Hindi in roman script: ek rekhaa kaa samiikaran y baraabar he m x plus c me

If such a course gets translated, it will have the same properties of an unintelligible source transcript and will be difficult for a reader or listener in the target language. When mathematical notations are used in the source text, the translations tend to preserve the same notations.

- English: The equation of a line is y = m * x + c.
- Hindi: रेखा का समीकरण y = m * x + c है ।
- Hindi in roman script: rekhaa kaa samiikaran y = m * x + c he

We present an approach to identify equation spans and convert them into their mathematical notations.

9.2.2 Equation Identification

We modeled the *Equation Identification* task as a sequence labeling task. The input in our task is a sentence, and the output is a sequence of labels annotated using the IOB tagging scheme [101]. We used 3 labels for annotating words in any sentence.

1. *B-EQ* - denotes the starting of an equation

2. *E-EQ* - denotes the continuation of an equation

3. *O* - denotes the absence of an equation

An example of a sentence annotated with this scheme is shown in Figure 9.3.

The	O	
probability		O
of	O	
an	O	
event	O	
e	O	
is	O	
calculated		O
as	O	
p	B-EQ	
of	I-EQ	
e	I-EQ	
is	I-EQ	
equal	I-EQ	
to	I-EQ	
n	I-EQ	
of	I-EQ	
e	I-EQ	
divided		I-EQ
by	I-EQ	
n	I-EQ	
of	I-EQ	
s	I-EQ	
.	O	

Figure 9.3: IOB annotation for Equation Identification

9.2.2.1 Data Preparation

As any pre-existing dataset for this task was missing, we created an annotated dataset ourselves. For dataset creation, we choose a probability course offered in NPTEL [1]. Firstly, we collected the available English transcripts from the NPTEL website. Instead of annotating everything manually, we followed a semi-automatic approach. We designed an annotator program using regular expressions to find the common mathematical patterns. After the automatic annotation, an expert validates the annotated equations. For the task, we performed inter-annotator agreement with two annotators and obtained an agreement or kappa score of 0.847, which shows perfect agreement. The annotators disagreed on long equations - one annotator chose the whole as a single equation while the other one split it into two smaller equations.

9.2.2.1.1 Regular Expression based Identifier

Regular expressions used to extract the equations are represented in Figure 9.4. Notations are described below:

[1] `https://nptel.ac.in/`

```
LEFT = (left_str \s)*
RIGHT = (\s right_str)*
MID = (mid_str \s)*
VAR = LEFT vars_str RIGHT
Z = (\s | . | , | ↑ ; ) VAR (\s MID VAR)* (\s | . | , | ; | $ )
REGEXP = Z (assign_str Z)*
```

Figure 9.4: Regular Expressions to extract equations

- $left\_str$ → single operators appearing to the left of an operand in a sentence e.g. inverse of x, factorial of y, minus xyz, mod of p
- $left\_str$ → single operators appearing to the left of an operand in a sentence e.g. y factorial, x square, x cube, z naught
- mid_str → binary operators appearing in between two operands in a sentence e.g. x multiplied by y, w divided by z, w raised to the power 4.
- $assign\_str$ → assignment phrases e.g. is like, gives us, is given by the formula, will become
- $vars\_str$ → mathematical variables (usually single alphabets or Greek variables)

9.2.2.1.2 Challenges in Identification

Due to the inconsistencies during transcription, there lies a lot of ambiguities in identification of mathematical variables. Sometimes multiple alphabets are written without spaces in between them. This causes chances of misinterpreting the actual semantics of the equations. So, we inserted spaces in between alphabets when they occurred together. The biggest ambiguity arose between 'a' and 'i'. Both of these alphabets can either be used as a pronoun or a variable. We used part-of-speech information of the contextual words for disambiguation.

9.2.2.1.3 Corpus Details

We annotated **1255** sentences containing **2868** equations from the probability domain.

9.2.3 Approaches for Span Identification

We used machine learning based and neural network based approaches.

9.2.3.1 Conditional Random Field

The initial models were developed using conditional random fields (CRF) [102]. Word based contextual features are incorporated for different context window sizes. We used the CRF++ toolkit [103] for implementation of CRF.

9.2.3.2 Neural Networks Based Models

Different neural network models were developed for this task. The details of the used features and training parameters are given below.

1. Bidirectional LSTM [104]
    - Word Embeddings: Glove [59]
    - Dimension: 300
    - Optimizer: RMSProp [105]
    - Epochs: 30
    - Loss Function: Categorical Cross Entropy
    - Implemented using Keras Framework
2. bidirectional GRU [54]
    - Word Embeddings: Glove [59]
    - Dimension: 300
    - Optimizer: RMSProp [105]
    - Epochs: 30
    - Loss Function: Categorical Cross Entropy
    - Implemented using Keras Framework
3. BiLSTM-CRF [106]
    - Word Embeddings: Glove [59] embeddings
    - Dimension: 300
    - Optimizer: Stochastic Gradient Descent [107]
    - Epochs: 20
    - Loss Function: Negative Log Likelihood
    - Implemented using Pytorch Framework
4. Tranformer: RoBERTa-base Finetuning [104] for Token Classification Task
    - RoBERTa-base Byte Level Subword Embeddings
    - Dimension: 768
    - Optimizer: AdamW [108, 109]
    - Epochs: 20
    - Loss Function: Cross Entropy
    - Implemented using HuggingFace Pytorch Framework

9.3 Results

The results for CRF are presented in Table 9.1. The increase in F1 scores is marginal with an increase in context windows, but bigger context windows capture more equation terms. This was the baseline for the task.

Window Size	Precision	Recall	F1
3	0.38	0.44	0.41
5	0.40	0.41	0.41
7	0.43	0.42	0.42

Table 9.1: CRF Results for Equation Identification

Model	Precision	Recall	F1
BiLSTM	0.62	0.67	0.64
BiGRU	0.65	0.66	0.65
BiLSTM-CRF	0.64	0.69	0.66
RoBERTa-base-Finetuning	0.66	0.7	0.68

Table 9.2: Results of Neural Networks for Equation Identification

We performed *5-fold* cross-validation for all our models. Results in Table 9.2 show that the neural networks based equation identifiers outperform the CRF based models by a significant margin. But all the models suffer from low precision. The actual spans of the equations are not identified properly. In most cases, the starting span is correctly identified whereas the models mislabel the end of the span. Most times the predicted equation span exceeds the original gold span.

9.3.1 Conversion into Math Notations

The second task is to convert the identified equation terms into their corresponding symbolic forms. We created a bottom up parser using the CYK [110] algorithm to build an expression tree.

9.3.1.1 Grammar Modeling and CYK Parsing

The grammar used for the conversion into notation is shown in Figure 9.5. S, Bin, Binop, L, R, X are all non-terminals and var, sinop_l, sinop_r, binop are preterminal sysmbols. The

S -> S Bin | L S | S R | S X | var
X -> X X | L S | S R | var
L -> sinop_l
R -> sinop_r
Bin -> Binop S
Binop -> binop

Figure 9.5: Grammar for Conversion of the Equation Terms

terminal symbols are the actual mathematical symbols like '+,-,*,/,<' etc. Binop represents the strings for the binary operators. sinop_l and sinop_r denote the string appearing to the left and right of operands respectively. var represents the variable or operand strings. These notations are similar to those already shown in Section 9.2.2.1.1. Abstract Syntax Tree (AST) imposes

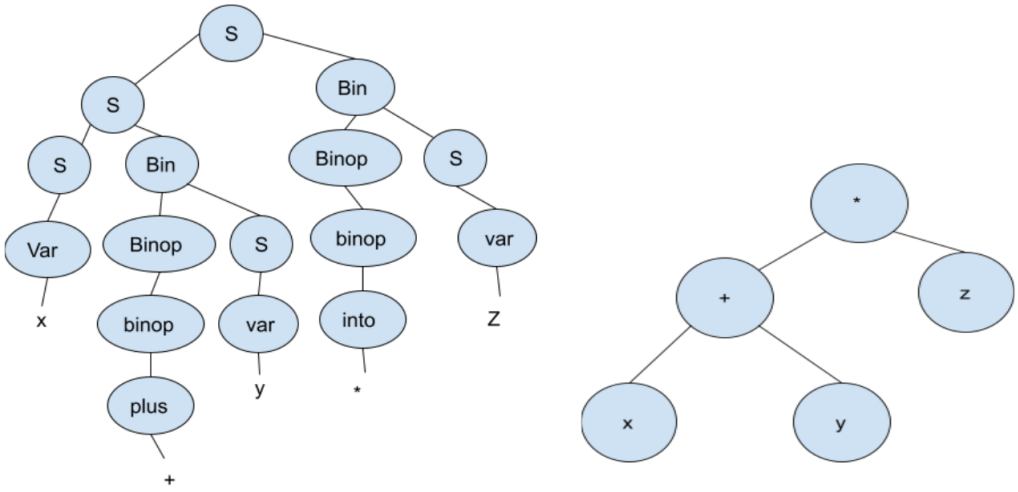

Figure 9.6: Abstract Syntax Tree and Expression Tree for *x plus y into z*

a precedence order on the nodes which in turn is helpful for the expression tree. We output all possible parses for any input string. We show one AST and expression tree in Figure 9.6. Another expression tree possible for 'x plus y into z' is $x + (y * z)$.

With the pre-defined ruleset, we correctly convert 43% of the identified equations. We do not consider the symbol conversions or equations with a single word for calculating the accuracy. We will explore the idea of modeling this as a sequence labeling task where the input is the identified equation terms and the output is the corresponding mathematical symbols.

Chapter 10

Conclusion and Future Work

In this chapter, we will summarize the main contributions of this thesis. We discuss some future directions spawning out of this research as well.

10.1 Conclusion

In this work, we discussed different word problem solving approaches and their strengths and weaknesses. We started with frame based techniques and refined some existing approaches as well. We then showed the efficacy of deep learning models and also their limitations. Generation with different equation notations are thoroughly experimented with. We also discussed the characteristics of available word problem datasets in depth. Development of effective and robust data augmentation techniques is the need of the hour. We proposed one generic augmentation technique to enrich any word problem dataset and show the impact of diversity in word problem datasets on the performance of automatic solvers. The data augmentation technique was tested on two languages: English and Hindi. We also show that translations can be used as a data augmentation technique when the resources are really scarce. This can boost the field of word problem solving for resource poor languages such as Indian Languages.

Word problem solving techniques can also assist various NLP tasks. Equation span identification and conversion is one of them for which initial models have been developed. This can easily be integrated into any Speech-to-speech MT system and can improve the performance of any MT system in translating mathematical equations and formulaic expressions. With the development of different models for word problem solving, there is a need to create benchmark datasets that take into account diversity in terms of both lexicon and templates. To address this issue, we manually created 3 benchmark datasets in English, Hindi, and Telugu. As one of the initial teams to work on word problem solving in Indian languages, we developed solvers in English, Hindi, and Telugu. We also empirically show how transfer learning helps when fine tuning on a new task in multilingual models.

10.2 Future Directions

Although we presented some effective modeling strategies and data augmentation techniques in the domain of word problem solving, there remains some promising future directions.

- Although we showed the importance of equation equivalence in evaluation, we did not explore its effects when the number of operations increases. A graph based evaluation can also be pondered.
- Graph based transformers are state-of-the-art models for English, but are not explored for Indian languages. This would be an interesting area which requires additional toolkits such as dependency parser, named entity recognizer.
- A thorough investigation into the difference in generation of equations using different notation types is an immediate future task.
- In this study, we could explore only two Indian languages. It will be an interesting area of research to explore the language relatedness between languages to solve word problems.
- Speech transcripts from mathematics and technology domains present a different challenge where the input is in spoken form. Equation identification and conversion is a starting step in this direction. Equation or mathematical notation conversion can open up many possibilities for word problem solving when the text is spoken.

Appendix

A List of Frames

1. Possess
2. Contain
3. Existence
4. State Fact
5. Transfer Goods Negative
6. Transfer Goods Positive
7. Transfer Money
8. Gather
9. Choose
10. Use Resource
11. Separate Entities
12. Getting
13. Create
14. Destruction
15. Add to Group
16. Motion
17. Duplication
18. Aggregate
19. Total
20. Addition
21. Negation
22. Require

Related Publications

1. Purvanshi Mehta, Pruthwik Mishra, Vinayak Athavale, Manish Shrivastava, and Dipti Misra Sharma, "Deep Neural Network based system for solving Arithmetic Word problems", Proceedings of the IJCNLP 2017, System Demonstrations, Tapei, Taiwan, pages 65–68

2. Pruthwik Mishra, Litton J Kurisinkel, and Dipti Misra Sharma, "Arithmetic Word Problem Solver using Frame Identification", CICLing-2018

3. Pruthwik Mishra, Litton J Kurisinkel, Dipti Misra Sharma, and Vasudeva Varma, "EquGener: A Reasoning Network for Word Problem Solving by Generating Arithmetic Equations", Proceedings of the 32nd Pacific Asia Conference on Language, Information and Computation, Hong Kong, pages 456-465

4. Vandan Mujadia, Pruthwik Mishra, and Dipti Misra Sharma, "Deep Contextual Punctuator for NLG Text", SEPP-NLG 2021 Shared Task

5. Harshita Sharma, Pruthwik Mishra, and Dipti Misra Sharma, "HAWP: a Dataset for Hindi Arithmetic Word Problem Solving", Proceedings of the Thirteenth Language Resources and Evaluation Conference, Marseille, France, pages 3479–3490

6. Pruthwik Mishra, Litton J Kurisinkel, and Dipti Misra Sharma, "Arithmetic Word Problem Solver using Frame Identification", Extended Version, POLIBITS Journal, Accepted, Not Published

7. Pruthwik Mishra, Vandan Mujadia, and Dipti Misra Sharma, "Multi Task Learning Based Shallow Parsing for Indian Languages", submitted to TALLIP Journal, Received 1st Review

8. Harshita Sharma, Pruthwik Mishra, and Dipti Misra Sharma, "Verb Categorisation for Hindi Word Problem Solving", Accepted in ICON-2023

Bibliography

[1] D. G. Bobrow, "Natural language input for a computer problem solving system," *Ph. D. Thesis, Department of Mathematics, MIT*, 1964.

[2] C. R. Fletcher, "Understanding and solving arithmetic word problems: A computer simulation," *Behavior Research Methods, Instruments, & Computers*, vol. 17, no. 5, pp. 565–571, 1985.

[3] D. Dellarosa, "A computer simulation of children's arithmetic word-problem solving," *Behavior Research Methods, Instruments, & Computers*, vol. 18, no. 2, pp. 147–154, 1986.

[4] Y. Bakman, "Robust understanding of word problems with extraneous information," *arXiv preprint math/0701393*, 2007.

[5] S. S. Sundaram and D. Khemani, "Natural language processing for solving simple word problems," in *Proceedings of the 12th International Conference on Natural Language Processing*, 2015, pp. 394–402.

[6] M. J. Hosseini, H. Hajishirzi, O. Etzioni, and N. Kushman, "Learning to solve arithmetic word problems with verb categorization," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 523–533.

[7] N. Kushman, Y. Artzi, L. Zettlemoyer, and R. Barzilay, "Learning to automatically solve algebra word problems," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2014, pp. 271–281.

[8] L. Zhou, S. Dai, and L. Chen, "Learn to solve algebra word problems using quadratic programming," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 817–822.

[9] S. Upadhyay and M.-W. Chang, "Annotating derivations: A new evaluation strategy and dataset for algebra word problems," *arXiv preprint arXiv:1609.07197*, 2016.

[10] S. Roy and D. Roth, "Solving general arithmetic word problems," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 1743–1752.

[11] ——, "Unit dependency graph and its application to arithmetic word problem solving," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 31, no. 1, 2017. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/10959

[12] S. Roy, S. Upadhyay, and D. Roth, "Equation parsing: Mapping sentences to grounded equations," *arXiv preprint arXiv:1609.08824*, 2016.

[13] B. Amnueypornsakul and S. Bhat, "Machine-guided solution to mathematical word problems," in *Proceedings of the 28th Pacific Asia Conference on Language, Information and Computing*, 2014.

[14] Y.-C. Lin, C.-C. Liang, K.-Y. Hsu, C.-T. Huang, S.-Y. Miao, W.-Y. Ma, L.-W. Ku, C.-J. Liau, and K.-Y. Su, "Designing a tag-based statistical math word problem solver with reasoning and explanation," in *International Journal of Computational Linguistics & Chinese Language Processing, Volume 20, Number 2, December 2015-Special Issue on Selected Papers from ROCLING XXVII*, 2015.

[15] D. C. Liu and J. Nocedal, "On the limited memory bfgs method for large scale optimization," *Mathematical programming*, vol. 45, no. 1-3, pp. 503–528, 1989.

[16] C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky, "The stanford corenlp natural language processing toolkit," in *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, 2014, pp. 55–60.

[17] G. A. Miller, "Wordnet: a lexical database for english," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[18] Y. Wang, X. Liu, and S. Shi, "Deep neural solver for math word problems," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017, pp. 845–854.

[19] D. Huang, 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," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2016, pp. 887–896.

[20] Z. Xie and S. Sun, "A goal-driven tree-structured neural model for math word problems." in *IJCAI*, 2019, pp. 5299–5305.

[21] W. Ling, D. Yogatama, C. Dyer, and P. Blunsom, "Program induction by rationale generation: Learning to solve and explain algebraic word problems," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2017, pp. 158–167.

[22] D. Huang, J.-G. Yao, C.-Y. Lin, Q. Zhou, and J. Yin, "Using intermediate representations to solve math word problems," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2018, pp. 419–428.

[23] Q. Liu, W. Guan, S. Li, and D. Kawahara, "Tree-structured decoding for solving math word problems," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019, pp. 2370–2379.

[24] K. Griffith and J. Kalita, "Solving arithmetic word problems automatically using transformer and unambiguous representations," in *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, 2019, pp. 526–532.

[25] ——, "Solving arithmetic word problems with transformers and preprocessing of problem text," *arXiv preprint arXiv:2106.00893*, 2021.

[26] L. Wang, Y. Wang, D. Cai, D. Zhang, and X. Liu, "Translating a math word problem to an expression tree," *arXiv preprint arXiv:1811.05632*, 2018.

[27] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[28] A. Patel, S. Bhattamishra, and N. Goyal, "Are nlp models really able to solve simple math word problems?" *arXiv preprint arXiv:2103.07191*, 2021.

[29] S. S. Sundaram, S. Gurajada, M. Fisichella, S. S. Abraham, *et al.*, "Why are nlp models fumbling at elementary math? a survey of deep learning based word problem solvers," *arXiv preprint arXiv:2205.15683*, 2022.

[30] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[31] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[32] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.

[33] H. Sharma, P. Mishra, and D. M. Sharma, "Hawp: a dataset for hindi arithmetic word problem solving," in *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, 2022, pp. 3479–3490.

[34] R. Koncel-Kedziorski, S. Roy, A. Amini, N. Kushman, and H. Hajishirzi, "Mawps: A math word problem repository," in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2016, pp. 1152–1157.

[35] S. Roy and D. Roth, "Mapping to declarative knowledge for word problem solving," *Transactions of the Association for Computational Linguistics*, vol. 6, pp. 159–172, 2018.

[36] W. Zhao, M. Shang, Y. Liu, L. Wang, and J. Liu, "Ape210k: A large-scale and template-rich dataset of math word problems," *arXiv preprint arXiv:2009.11506*, 2020.

[37] S.-y. Miao, C.-C. Liang, and K.-Y. Su, "A diverse corpus for evaluating and developing english math word problem solvers," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020, pp. 975–984.

[38] D. Flickinger, "On building a more effcient grammar by exploiting types," *Natural Language Engineering*, vol. 6, no. 1, pp. 15–28, 2000.

[39] B. Birnbaum and K. J. Goldman, "An improved analysis for a greedy remote-clique algorithm using factor-revealing lps," *Algorithmica*, vol. 55, no. 1, pp. 42–59, 2009.

[40] S.-Y. Miao, C.-C. Liang, and K.-Y. Su, "A diverse corpus for evaluating and developing english math word problem solvers," *arXiv preprint arXiv:2106.15772*, 2021.

[41] C. F. Baker, C. J. Fillmore, and J. B. Lowe, "The berkeley framenet project," in *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics-Volume 1*. Association for Computational Linguistics, 1998, pp. 86–90.

[42] M. Honnibal and I. Montani, "spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing," 2017, to appear.

[43] F. Pedregosa, *et al.*, "Scikit-learn: Machine learning in python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Nov. 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=1953048.2078195

[44] C. Cortes and V. Vapnik, "Support vector machine," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[45] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[46] K. Sparck Jones, "A statistical interpretation of term specificity and its application in retrieval," *Journal of documentation*, vol. 28, no. 1, pp. 11–21, 1972.

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

[48] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[49] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[50] T. Wolf, *et al.*, "Huggingface's transformers: State-of-the-art natural language processing," *arXiv preprint arXiv:1910.03771*, 2019.

[51] D. Chen and C. Manning, "A fast and accurate dependency parser using neural networks," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 740–750.

[52] S. Sukhbaatar, J. Weston, R. Fergus, *et al.*, "End-to-end memory networks," in *Advances in neural information processing systems*, 2015, pp. 2440–2448.

[53] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[54] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[55] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[56] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting." *Journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[57] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[58] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. [Online]. Available: https://arxiv.org/abs/1908.10084

[59] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[60] A. M. Lamb, A. G. ALIAS PARTH GOYAL, Y. Zhang, S. Zhang, A. C. Courville, and Y. Bengio, "Professor forcing: A new algorithm for training recurrent networks," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 4601–4609. [Online]. Available: http://papers.nips.cc/paper/6099-professor-forcing-a-new-algorithm-for-training-recurrent-networks.pdf

[61] A. Mitra and C. Baral, "Learning to use formulas to solve simple arithmetic problems." in *ACL (1)*, 2016.

[62] S. Bird and E. Loper, "Nltk: the natural language toolkit," in *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*. Association for Computational Linguistics, 2004, p. 31.

[63] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249–256.

[64] F. Chollet *et al.*, "Keras," https://github.com/fchollet/keras, 2015.

[65] S. Roy and D. Roth, "Solving general arithmetic word problems," *arXiv preprint arXiv:1608.01413*, 2016.

[66] G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. M. Rush, "Opennmt: Open-source toolkit for neural machine translation," *arXiv preprint arXiv:1701.02810*, 2017.

[67] B. Heinzerling and M. Strube, "BPEmb: Tokenization-free Pre-trained Subword Embeddings in 275 Languages," in *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, N. C. C. chair), *et al.*, Eds. Miyazaki, Japan: European Language Resources Association (ELRA), May 7-12, 2018 2018.

[68] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *The Journal of Machine Learning Research*, vol. 21, no. 1, pp. 5485–5551, 2020.

[69] J. Wei and K. Zou, "Eda: Easy data augmentation techniques for boosting performance on text classification tasks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019, pp. 6382–6388.

[70] M. Maimaiti, Y. Liu, H. Luan, Z. Pan, and M. Sun, "Improving data augmentation for low-resource nmt guided by pos-tagging and paraphrase embedding," *Transactions on Asian and Low-Resource Language Information Processing*, vol. 20, no. 6, pp. 1–21, 2021.

[71] E. Loper and S. Bird, "Nltk: The natural language toolkit," *arXiv preprint cs/0205028*, 2002.

[72] C. Bannard and C. Callison-Burch, "Paraphrasing with bilingual parallel corpora," in *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, 2005, pp. 597–604.

[73] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013.

[74] Z. Wu and M. Palmer, "Verb semantics and lexical selection," *arXiv preprint cmp-lg/9406033*, 1994.

[75] K. Heafield, "Kenlm: Faster and smaller language model queries," in *Proceedings of the sixth workshop on statistical machine translation*, 2011, pp. 187–197.

[76] K. Heafield, I. Pouzyrevsky, J. H. Clark, and P. Koehn, "Scalable modified kneser-ney language model estimation," in *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2013, pp. 690–696.

[77] S. Yadav, A. Ahsan, M. Shrivastava, and D. M. Sharma, "Chinese to english machine translation system using transformers," 2020, to appear.

[78] J. Tiedemann and S. Thottingal, "Opus-mt–building open translation services for the world," in *Proceedings of the 22nd Annual Conference of the European Association for Machine Translation*. European Association for Machine Translation, 2020.

[79] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.

[80] V. Kumar, R. Maheshwary, and V. Pudi, "Practice makes a solver perfect: Data augmentation for math word problem solvers," *arXiv preprint arXiv:2205.00177*, 2022.

[81] O. Chatterjee, I. Pandey, A. Waikar, V. Kumar, and G. Ramakrishnan, "Warm: A weakly (+ semi) supervised math word problem solver," in *Proceedings of the 29th International Conference on Computational Linguistics*, 2022, pp. 4753–4764.

[82] G. N. Jha, "The tdil program and the indian langauge corpora intitiative (ilci)." in *LREC*, 2010.

[83] G. Ramesh, *et al.*, "Samanantar: The largest publicly available parallel corpora collection for 11 indic languages," *ArXiv*, vol. abs/2104.05596, 2021.

[84] J. Philip, S. Siripragada, V. P. Namboodiri, and C. Jawahar, "Revisiting low resource status of indian languages in machine translation," in *8th ACM IKDD CODS and 26th COMAD*, 2021, pp. 178–187.

[85] J. Philip, V. P. Namboodiri, and C. V. Jawahar, "A baseline neural machine translation system for indian languages," 2019.

[86] V. Goyal, S. Kumar, and D. M. Sharma, "Efficient neural machine translation for low-resource languages via exploiting related languages," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*, 2020, pp. 162–168.

[87] V. Mujadia and D. M. Sharma, "Boosting mt performance for indian languages," 2022, to appear.

[88] T. Mikolov, E. Grave, P. Bojanowski, C. Puhrsch, and A. Joulin, "Advances in pre-training distributed word representations," in *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[89] P. Mishra, L. J. Kurisinkel, D. M. Sharma, and V. Varma, "Equgener: A reasoning network for word problem solving by generating arithmetic equations," in *Proceedings of the 32nd Pacific Asia Conference on Language, Information and Computation*, 2018.

[90] P. Mehta, P. Mishra, V. Athavale, M. Shrivastava, and D. Sharma, "Deep neural network based system for solving arithmetic word problems," in *Proceedings of the IJCNLP 2017, System Demonstrations*, 2017, pp. 65–68.

[91] P. Mishra, V. Mujadia, and D. M. Sharma, "Multi task learning based shallow parsing for indian languages," 2023, to appear.

[92] G. Katrapati, "Developing a word2vec model for hindi from news articles," 2017, to appear.

[93] P. Bhattacharyya, "Indowordnet," in *International Conference on Language Resources and Evaluation*, 2010.

[94] A. Conneau, K. Khandelwal, N. Goyal, V. Chaudhary, G. Wenzek, F. Guzmán, E. Grave, M. Ott, L. Zettlemoyer, and V. Stoyanov, "Unsupervised cross-lingual representation learning at scale," *arXiv preprint arXiv:1911.02116*, 2019.

[95] V. Mujadia and D. M. Sharma, "Post edit me: An ai enabled post editing tool for speech to speech machine translation," 2021, to appear.

[96] A. Mhaske, H. Kedia, S. Doddapaneni, M. M. Khapra, P. Kumar, R. Murthy V, and A. Kunchukuttan, "Naamapadam: A large-scale named entity annotated data for indic languages," *arXiv preprint arXiv:2212.10168*, 2022.

[97] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," 2016.

[98] J. Chauhan, "An overview of mooc in india," *International Journal of Computer Trends and Technology*, vol. 49, no. 2, pp. 111–120, 2017.

[99] V. Kordoni, A. van den Bosch, K. L. Kermanidis, V. Sosoni, K. Cholakov, I. Hendrickx, and M. Huck, "Enhancing access to online education: Quality machine translation of mooc content," 2016.

[100] V. Sosoni, K. L. Kermanidis, M. Stasimioti, T. Naskos, E. Takoulidou, M. Van Zaanen, S. Castilho, P. Georgakopoulou, V. Kordoni, and M. Egg, "Translation crowdsourcing: Creating a multilingual corpus of online educational content," in *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[101] L. Ramshaw and M. Marcus, "Text chunking using transformation-based learning," in *Third Workshop on Very Large Corpora*, 1995. [Online]. Available: https://aclanthology.org/W95-0107

[102] J. Lafferty, A. McCallum, and F. C. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," 2001.

[103] T. Kudo, "Crf++: Yet another crf toolkit (2005)," *Available under LGPL from the following URL: http://crfpp. sourceforge. net*, 2015.

[104] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional lstm and other neural network architectures," *Neural networks*, vol. 18, no. 5-6, pp. 602–610, 2005.

[105] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent," *Cited on*, vol. 14, no. 8, p. 2, 2012.

[106] Z. Huang, W. Xu, and K. Yu, "Bidirectional lstm-crf models for sequence tagging," *arXiv preprint arXiv:1508.01991*, 2015.

[107] L. Bottou, "Stochastic gradient descent tricks," in *Neural Networks: Tricks of the Trade: Second Edition*. Springer, 2012, pp. 421–436.

[108] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[109] ——, "Fixing weight decay regularization in adam," 2018.

[110] I. Sakai, "Syntax in universal translation," in *Proceedings of the International Conference on Machine Translation and Applied Language Analysis*, 1961.