Final Report: Multilingual Deep Neural Math Word Problem Solver

Ramya Sree Boppana
rboppana6

Shrija Mishra
smishra309

Sheryl Ratnam
sratnam6

1 Abstract

Researchers have applied various deep-learning methods to the design of automatic math word problem (MWP) solvers. However, most of this research has been focused on English-language MWPs. This paper presents a language agnostic MWP solver that aims to bridge the gap between conventional deep learning-based methods and multilingual word embeddings in this research space. We build upon three RNN-based models – Predictor, Generator, and GenPred- by incorporating a universal language agnostic word embedding module. In addition to leveraging template-based solution frameworks, our models uniquely optimize for transfer learning via multilingual word representations. We train and test on two, multilingual data corpora – Math23K in Chinese and a subset of Dolphin-S in English. The experimental results reveal an average answer accuracy at about 50%, which demonstrates that these models perform better relative to the proposed baselines. We find that the Predictor model exhibits the strongest performance with an answer accuracy of 72%. Our major contributions are two-fold. First, we build a large-scale multilingual, MWP dataset – LAMP32K – and integrate language agnostic word embedding layers across our models. Additionally, we provide an extensive evaluation and comparison of 3 RNN-based models on the multilingual MWP solving task. The code is hosted at the following link https://github.com/shrija14/Multilingual-Deep-Neural-Math-Word-Problem-Solver

2 Introduction & Related Work

The task of solving mathematical word problems (MWPs) aims to formalize a word problem in natural language and automatically derive a mathematical query to be processed. Most mathematics software interfaces remain incapable of reliably completing this task, as it requires a process of mapping natural language into machine-understandable logic forms in addition to inferring a numeric answer. Apart from arithmetic computation, this methodology can be abstracted to a range of quantitative reasoning skills required for intelligent autonomous agents, such as financial news analyses or general question answering services.

Various approaches to this research question have continuously evolved since its inception in the 1960s. A majority of precedent research has focused on applying symbolic or statistical learning methods to this problem. These methods were developed in the face of dataset limitations, and as a result, are ill-equipped to handle large, diversified datasets. Huang et. al. first identified this weakness by evaluating these concurrent state-of-the-art approaches on a large-scale, English language corpus, Dolphin18K, developed by the authors (Huang et al., 2016).

In response, a new family of deep-learning based methods has emerged to tackle this challenge. Wang et al. pioneered this research direction with the introduction of Deep Neural Solver, a seq2seq model that integrates a similarity-based retrieval and recurrent neural networks (RNN) (Wang et al., 2017). They also contributed a new Chinese language dataset, Math23K, to address the lack of publicly available, large-scale datasets for automatic MWP solving.

Following DNS, there have emerged multiple DL-based solvers that further corroborate this approach to MWP solving. Seq2SeqET, proposed by Wang et. al. extends DNS by suggesting the use of expression trees for templatization (Wang et al., 2018). However, as Wang et al. note, the seq2seq model is a black-box that lacks interpretability, plays a key role in assessing the usability of MWP solvers, particularly in applications to online education and tutoring. (Wang et al., 2019). To resolve this, Wang et al. introduce *Template-RNN* (T-RNN), which improves upon Seq2SeqET

Dataset Name	#prob	#temp	#words
Dolphin-S	140	110	19K
Math23K	23161	4326	822K
LAMP32K	32532	4381	830K

Table 1: Data Summary Statistics

by leveraging the template-based solution architecture together with RNNs (Wang et al., 2019). When evaluated on the Math23K dataset, the T-RNN model shows results that improve upon the Seq2Seq model by approximately 10% (Wang et al., 2019). T-RNN is the state-of-the-art method for Math23K.

We have observed that these aforementioned papers do not address language agnostic MWP solving. As Zhang et al. mention, solving English-language MWPs plays a dominating role in the literature (Zhang et al., 2019). Most existing proposed models are trained and evaluated on either Dolphin18K (English) or Math23K (Chinese), but never in conjunction. In seeking to bridge this language-dependent gap, we present a robust, language agnostic MWP solver. Our goal remains the same as previously described in our Project Proposal and Midway Report. We look to the two-stage T-RNN model in developing our approach. We extract the components of T-RNN to develop three RNN-based models - Generator, Predictor, and GenPred (combined). We compare model performance on the task of MWP solving and show that the Predictor model reaches the highest answer accuracy relative to the other models and baselines. Our key contribution involves embedding a language agnostic embedding module within these models to support multilingual word embeddings in the structure prediction and quantity representation stage. Our model for the language-agnostic word-embedding layer is motivated by the works of (Artetxe and Schwenk, 2019) and (Yang et al., 2019). In addition to building language-agnostic MWP solvers, we have also developed a bilingual dataset, LAMP32K, by combining Math23K and Dolphin-S.

3 Methods

3.1 Data

We use subsets of the largest publicly available datasets for MWP that span multiple languages.

Dolphin 18K (Huang et al., 2016) is an English-language dataset, consisting of over 18460 annotated web-answered questions from Yahoo! Answers. Dolphin-S is a subset of Dolphin18K that retains MWPs whose templates are associated with only one problem. We reduce this dataset further by filtering out multi-variable equations and single-variable equations with variables on both sides. This subsequently reduces Dolphin-S to 200 rows. In order to overcome this lack of data, we have augmented this dataset with modifications of the 200 rows. This gave us approximately 9K rows.

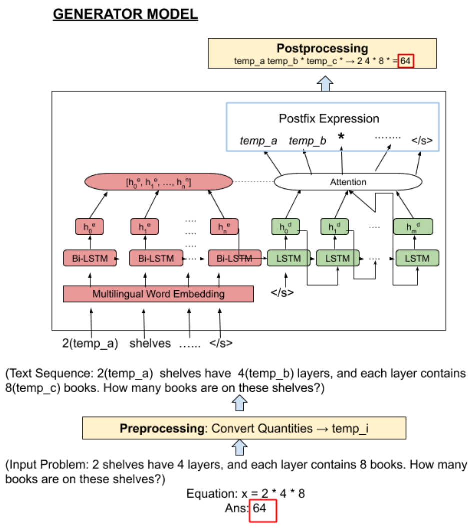
Math23K (Wang et al., 2017) is a Chinese-language dataset that contains Chinese elementary school math word problems scraped from online education websites. The dataset consists of 23161 single-variable MWPs and 2187 templates. Given that the data is in Chinese, we used Google Translate to debug and understand the dataset. Preprocessing resulted in a minimal rows being dropped resulting in data size of 23130.

Our combined dataset LAMP32K (Language Agnostic Math Problems) has 32532 rows with English and Chinese single variable math word problems. These equations have been tweaked to comply with the format where the unknown is on the left side and known quantities are on the right side. Table 1 includes the summary statistics for all the mentioned datasets, detailing the number of problems, equation templates, and words in each. The pre-processing procedures performed on these datasets are described in detail in section 3.2.1. We see that there are changes in Dolphin-S and Math23K as per our previous reporting. This is because we have incorporated questions with constants and percentages. Therefore, we don't filter out all equations where the number of literals in the equation is less than the ones in question.

3.2 Models/Analysis

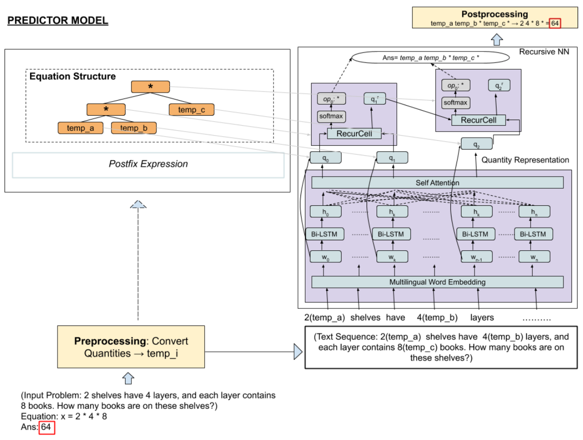
We build upon 3 RNN-based models derived from the T-RNN model proposed by Wang et al (Wang et al., 2019). To build a language agnostic MWP solver, we extend each of these models by incorporating a multilingual word embedding layer. This module layer is derived from USE, a multi-task dual-encoder framework built for transfer learning (Yang et al., 2019). We then proceed to evaluate each model's performance. The following a high-level overview of each model:

1. *Generator Model*: This model is geared towards generating an equation template structure by transforming the problem text to a math equation in the form of postfix expression. We use multilingual word embeddings



GENERATOR MODEL
Postprocessing
temp_a temp_b * temp_c * → 2 4 * 8 * = 64
Postfix Expression
temp_a temp_b * .... </s>
Attention
Bi-LSTM
LSTM
Multilingual Word Embedding
2(temp_a) shelves ...... </s>
(Text Sequence: 2(temp_a) shelves have 4(temp_b) layers, and each layer contains 8(temp_c) books. How many books are on these shelves?)
Preprocessing: Convert Quantities → temp_i
(Input Problem: 2 shelves have 4 layers, and each layer contains 8 books. How many books are on these shelves?)
Equation: x = 2 * 4 * 8
Ans: 64

Figure 1: Framework of Language Agnostic Generator Model



PREDICTOR MODEL
Postprocessing
temp_a temp_b * temp_c * → 2 4 * 8 * = 64
Recursive NN
Equation Structure
Postfix Expression
RecurCell
softmax
Quantity Representation
Self Attention
Bi-LSTM
Multilingual Word Embedding
2(temp_a) shelves have 4(temp_b) layers ....
Preprocessing: Convert Quantities → temp_i
(Text Sequence: 2(temp_a) shelves have 4(temp_b) layers, and each layer contains 8(temp_c) books. How many books are on these shelves?)
(Input Problem: 2 shelves have 4 layers, and each layer contains 8 books. How many books are on these shelves?)
Equation: x = 2 * 4 * 8
Ans: 64

Figure 2: Framework of Language Agnostic Predictor Model

in the quantity embedding network to vectorize quantities language agnostically.

2. *Predictor Model*: The RNN in this model is aimed at predicting unknown operators and building a complete mathematical expression, given a MWP and its corresponding post-fix equation template.

3. *GenPred Model*: This model combines the generator and predictor to tackle both equation template generation and answer prediction.

For clarity, we define an equation template as a general form of equations. For a problem P with equation E_p, we obtain the equation template by mapping numbers in E_p to a list of literals, $temp_i$.

The diagrams in Figures 1, 2, and 3 detail the high-level architecture of each Multilingual MWP Solver. The following sections go into detail about each of the models, and their respective input limitations.

3.2.1 Pre- & Postprocessing

All models have a preprocessing and postprocessing component. During preprocessing, we clean the data and templatize the raw input text. After filtering the data to include single variable equations, we focus on cleaning the problem text. Once the cleaned text contains relevant words and numbers, we templatize by replacing detected quantities in text and equation with literals, $temp\_i$. We also filter out complex problems where the number of literals in the problem text is less than the number of literals in the equation. This preprocessing procedure is intended to reduce the equation template space and improve interpretability in the RNN stage. The postprocessing stage involves substituting the generated post-fix expression with the quantities in the MWP, and calculating an answer.

3.2.2 Generator Model

The primary goal of this model is to learn the mapping between an input problem's text and post-fix equation. This model is drawn from Stage 1 of the T-RNN model proposed by Wang et. al. The input to this model is the templatized text sequence, where quantities in the original MWP are replaced with $temp_i$. We employ a seq2seq model, which consists of a multilingual word embedding layer, an encoder, and a decoder. The Bi-LSTM encoding layer vectorizes the templatized text. The decoder takes the vector representation to generate the post-fix expression. The output of the encoder model is passed to the attention layer to effectively drive quantity embedding. The output of the attention layer is in the form of a postfix expression. After postprocessing, this expression is augmented with the known mathematical quantities from the MWP text, which is used to compute the final answer.

To adapt this model towards multilingual MWPs, we integrate a multilingual word embedding layer by implementing a Universal Sentence Encoder (USE) to support transfer learning (Yang et al., 2019). We initially experimented with different language agnostic word embedding layers such as MUSE and LASER (Artetxe and Schwenk, 2019). We find that USE is best suited for our model, and consistently produces strong results. USE outputs 512 dimensional embedding representation of the

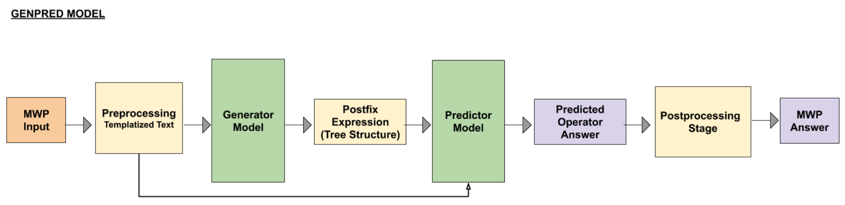

GENPRED MODEL
MWP Input
Preprocessing Templatized Text
Generator Model
Postfix Expression (Tree Structure)
Predictor Model
Predicted Operator Answer
Postprocessing Stage
MWP Answer

Figure 3: Framework of Language Agnostic GenPred Model

input string and is pre-trained on 16 languages, including Chinese and English.

3.2.3 Predictor Model

This model is derived from Stage 2 of the T-RNN model proposed by Wang et al (Wang et al., 2019). The input to this model includes both the templatized text sequence and the post-fix expression, according to the tree structure. Rather than generate the post-fix expression within the model, we provide it as input, such that it's only task is to infer the unknown operators with the use of a quantity embedding network and RNN. This is further elaborated in Figure 2. The templatized text is fed into the quantity embedding network, which is used to generate quantity embeddings. Again, we utilize a language agnostic word embedding layer, which vectorizes the multilingual question text and inputs the word vectors into the quantity embedding network. To more precisely capture the relationship between question words and the corresponding equation template, we pass the embeddings to an attention layer. Once we've obtained the vectorized quantities, we direct the postfix expression input to RNN to infer the unknown operators. RNN recursively applies a softmax function to estimate the probability of an operator until all operators are predicted.

3.2.4 GenPred Model

GenPred is a hybrid model that combines both the Generator and Predictor Model, with the input restricted to the templatized problem text. Here, we leave it to the Generator Model to generate the postfix equation template with placeholders for the operators. This template is leveraged by the Predictor Model to infer the actual operators. In this model, we utilize USE in both sub-models to derive multilingual word and quantity embeddings.

3.3 Baseline Models

We compare each model's performance to two baseline models. Given that the problem of multilingual MWP solving has not been tackled before,

Hyperparameter	GP Generator	Generator	Predictor
Batch Size	64	64	64
Epochs	33	50	20
Embedding Size	512	512	512
Encoder Hidden Size	512	512	160
Decoder Hidden Size	1024	1024	N/A
#Encoder Layers	2	2	2
#Decoder Layers	2	2	N/A
Teacher forcing ratio	0.83	0.5	N/A
Optimizer	Adam	Adam	SGD
Learning rate	0.01	0.01	0.01

Table 2: Tuned Hyperparameters

we introduce a new baseline which randomly generates answers given a template equation structure. The input to this model is the known quantities in the MWP text. It assumes that the order of the quantities remains the same and randomly predicts the unknown operators in the equation. This random component replaces the prediction made by the RNN in our paper's models.

For example, suppose we have the equation, $x = 2 + 3 * 1$ and its corresponding known quantities, $[2, 3, 1]$. The baseline sees this list of known quantities without knowing the operators between them. It then randomly chooses operators between each pair of quantities, and calculates the final answer.

To incorporate the commutative operations, we come up with a second baseline by extending the first one. Here we generate a permutation of the number list and then predict the operators in between and solve it to get an answer.

4 Results

4.1 Experiment Setup

4.1.1 Model Configuration

We split our LAMP32K dataset into train, test and validation sets in the ratio of 60:20:20. We used the validation vs train loss and accuracy curves to tune our hyperparameters. Table 2 shows the best set of hyperparameters for all the three models. We used the same set of hyperparameters for both Predictor model and the prediction module of the GenPred model. Figures 4, 5, 6, 7, 8 and 9 show the loss and accuracy curves for all the models.

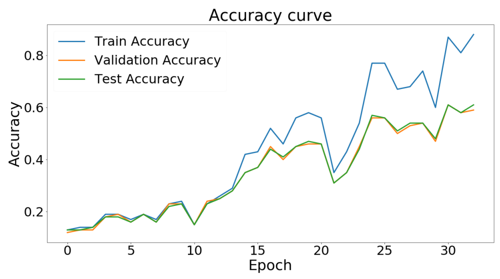


Figure 4: GenPred Model Accuracy

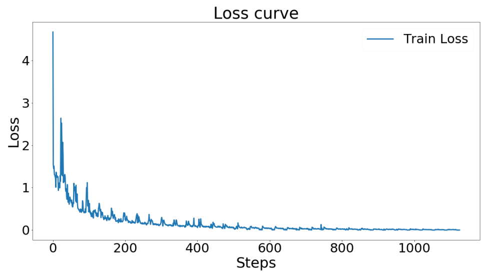


Figure 5: GenPred Model Loss

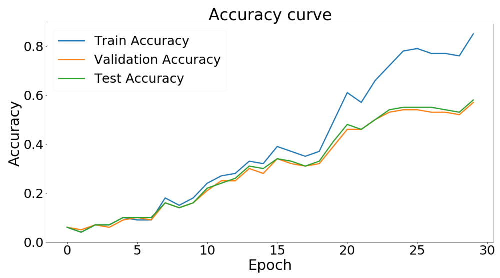


Figure 6: Generator Model Accuracy

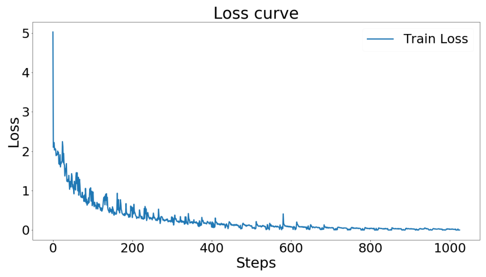


Figure 7: Generator Model Loss

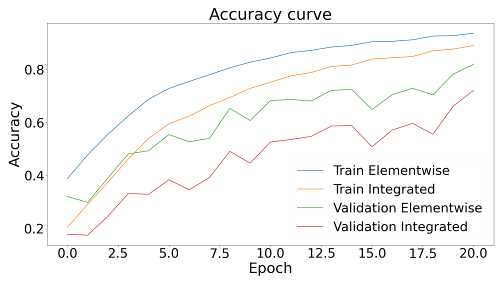


Figure 8: Predictor Model Accuracy

4.1.2 Evaluation Metric

We use Equation Integrated Accuracy and Answer Accuracy as the evaluation metrics. Equation integrated accuracy is one when the generated equation

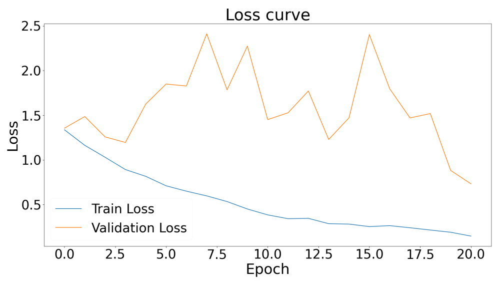


Figure 9: Predictor Model Loss

matches the actual equation entirely. Otherwise, it's 0. Answer accuracy is similarly calculated by comparing actual answer and predicted answer. For the Generator model, answer accuracy may be more than the Equation integrated accuracy as an equation can have several equivalent forms. For the Predictor model, equation integrated accuracy and the answer accuracy remain the same as the model predicts only the operators given an equation template. This implies that the correct/incorrect prediction of operators will result in an equation integrated accuracy of one/zero and hence, the answer accuracy will also be one/zero. For the GenPred model, we can evaluate the generator module using only the equation integrated accuracy since this phase generates equation templates without the actual operators. Similar to the Predictor model, equation integrated accuracy and answer accuracy will be equal for the prediction module of GenPred model.

4.2 Result Comparison and Analysis

We compare performances across the three model architectures by evaluating AA and EIA. Table 3 provides a comprehensive summary of each model's relative performance on the train and the test sets within LAMP32K. Here, EIA refers to Equation Integrated Accuracy and AA refers to Answer Accuracy.

The Predictor Model exhibits the strongest performance on multilingual data. It has the highest test accuracy at 72%. It exceeds the Generator Model's performance by 10.2% in terms of answer accuracy. Similarly, it outperforms the GenPred model by 57%. This model also performs significantly better than both baseline models, improving answer accuracy by an average of 65.65%. Figure 8 illustrates the Predictor Model's equation elementwise and integrated accuracy. Note that, within the model, elementwise test accuracy exceeds integrated test accuracy. We attribute the Predictor model's superior performance to the multifaceted input it accepts. It receives both templatized text, and the corresponding post-fix expression derived from the tree structure as input. In other words, the model is performing a diluted multi-class classification task, as it predicts 1 of 4 possible operators for each unknown.

The Generator Model displays the second highest performance on the test set, with an EIA of 58% and AA of 61.8%. As expected, the answer accuracy is more than the equation integrated accuracy since an equation can have several valid forms. Figure 12 shows one such scenario. This model exceeds the baseline by 50%. The model's answer accuracy is less compared to that of the Predictor model as the equation structure is not available and the model has to solve more difficult task of generating operators, operands and a valid postfix equation using these.

The GenPred Model has the weakest performance with an average AA and EIA of 15%. It exceeds the baseline model accuracy by approximately 8.5%, which is 34.6% below the average accuracy improvement over the baseline across all three models. One reason for the low accuracy is error accumulation. Given that GenPred combines two model components, it has the capacity to reflect errors across both. For example, if the Generator module in GenPred were to produce an invalid equation template, this error would carry over to the Prediction module, leading to an error in answer prediction. To further this point, there may be cases that work best in the Generator module but perform worse in the Prediction module, and vice versa. As a result, low accuracy may be exacerbated. We investigate one such case in Section 4.3.

We note early on that there have been no proposed models targeted at the task of multilingual MWP solving. However, as mentioned in Section 2, T-RNN proposed by Wang et. al. is the state-of-the-art model for single-language MWP solvers on large-scale datasets. The Generator, Predictor, and GenPred models are all derived from various components of the T-RNN model. More specifically, Generator is derived from the Template Prediction module (Stage 1) of T-RNN, and Predictor is derived from the Answer Generation module (Stage 2) of T-RNN. The key difference is that our models are equipped to handle multi-lingual data with the incorporation of the language agnostic word embedding layer, USE.

Model	Train EIA	Train AA	Test EIA	Test AA
Baseline 1	N/A	6.5%	N/A	6.5%
Baseline 2	N/A	6.2%	N/A	6.2%
Generator	85%	87%	58%	61.8%
Predictor	89%	89%	72%	72%
GP Generator	95%	N/A	61%	N/A
GP Predictor	19%	19%	15%	15%

Table 3: Train and Test Accuracies

We compare the accuracies achieved by our Generator and Predictor Models with the Template Prediction and Answer Generation modules in T-RNN, respectively. We keep in mind, however, that these two sets of models operate in different linguistic contexts.

Table 4 displays the answer accuracy of each of these models. While we cannot construct a direct 1:1 comparison between the two sets of models given that they were trained and tested on different datasets, we can identify a few similarities. First, Generator and Predictor do not achieve as high of an accuracy as their counterparts. The Template Prediction module in T-RNN outperforms Generator by 7.3%. The Answer Generation module in T-RNN outperforms Predictor by 22.4%. However, in both sets of models, the Generator/Template Prediction component underperforms in comparison to the Predictor/Answer Generation. This cross-applies to our earlier analysis, where we point out that Predictor/Answer Generation receive two inputs – the templatized text sequence *and* post-fix expression.

This difference in accuracies can be explained in a couple of different ways. First, there is a difference in dataset size and composition. LAMP32K is a composite of both Math23K and Dolphin-S, and consequently contains multilingual MWPs. Furthermore, there are procedural differences that may contribute to this difference. In T-RNN, Wang et. al. mention that, during the Template Prediction stage, they terminate the MWP solver if it happens to output an invalid equation template(Wang et al., 2019). Although this occurs 0.7% of the time in Math23K, this additional step can bolster the accuracy of the Template Prediction module. Moreover, in our model development, we do not discard invalid models for either Math23K or Dolphin-S in LAMP32K.

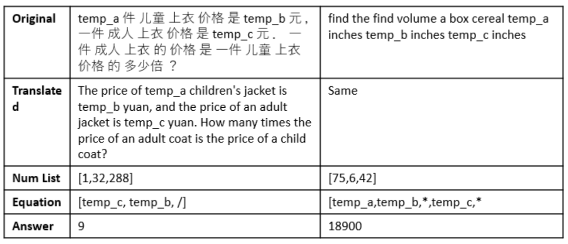
4.3 Error Analysis

To understand the Generator model's weak performance, we conduct an error analysis and offer an explanation as to why the model underperforms.

	Math23K	LAMP32K
Generator	-	61.8
Predictor	-	72
Template Prediction	69.1	-
Answer Generation	94.4	

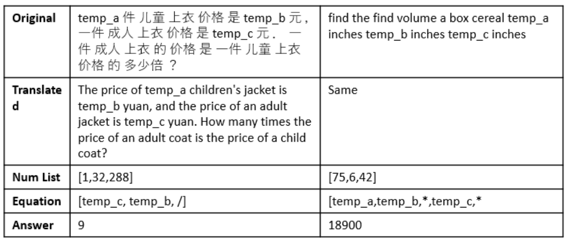
Table 4: MWP Solving Accuracy Comparison

Figures 10, 11 and 12 show sample outputs generated by the Generator model. In the Chinese example in Figure 10, we observe that the number list in the textual question is more than the numbers actually used in the equation to solve the question. The model is able learn the significant numbers and generate the equation correctly. From the failure cases of figure 11, we observe that the textual question is involved and the equation is complex, lengthy and has multiple operators. Even though the model is able to generate a valid postfix equation, it failed to predict the correct equation. In figure 12, we observe that the equation generated is equivalent to the actual equation even though it's not an exact match. However, the answer accuracy is one.



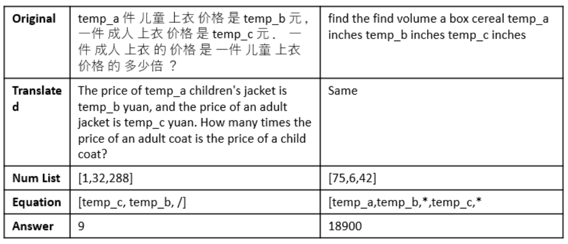
Original	temp_a 件 儿童 上衣 价格 是 temp_b 元，一件 成人 上衣 价格 是 temp_c 元． 一件 成人 上衣 的 价格 是 一件 儿童 上衣 价格 的 多少倍 ？	find the find volume a box cereal temp_a inches temp_b inches temp_c inches
Translated	The price of temp_a children's jacket is temp_b yuan, and the price of an adult jacket is temp_c yuan. How many times the price of an adult coat is the price of a child coat?	Same
Num List	[1,32,288]	[75,6,42]
Equation	[temp_c, temp_b, /]	[temp_a,temp_b,*,temp_c,*
Answer	9	18900

Figure 10: Generator Model- Success Example



Original	temp_a 件 儿童 上衣 价格 是 temp_b 元，一件 成人 上衣 价格 是 temp_c 元． 一件 成人 上衣 的 价格 是 一件 儿童 上衣 价格 的 多少倍 ？	find the find volume a box cereal temp_a inches temp_b inches temp_c inches
Translated	The price of temp_a children's jacket is temp_b yuan, and the price of an adult jacket is temp_c yuan. How many times the price of an adult coat is the price of a child coat?	Same
Num List	[1,32,288]	[75,6,42]
Equation	[temp_c, temp_b, /]	[temp_a,temp_b,*,temp_c,*
Answer	9	18900

Figure 11: Generator Model- Failure Example



Original	temp_a 件 儿童 上衣 价格 是 temp_b 元，一件 成人 上衣 价格 是 temp_c 元． 一件 成人 上衣 的 价格 是 一件 儿童 上衣 价格 的 多少倍 ？	find the find volume a box cereal temp_a inches temp_b inches temp_c inches
Translated	The price of temp_a children's jacket is temp_b yuan, and the price of an adult jacket is temp_c yuan. How many times the price of an adult coat is the price of a child coat?	Same
Num List	[1,32,288]	[75,6,42]
Equation	[temp_c, temp_b, /]	[temp_a,temp_b,*,temp_c,*
Answer	9	18900

Figure 12: Generator Model- Example Output

We have varying length equations with length ranging from 3 to 23. Table 5 shows the variation

Equation Length	Accuracy	Count
3	82.9%	9481
5	62.2%	13851
13	2%	270

Table 5: Equation Length wise Accuracies

Id	Template	Accuracy	Count
T1	temp_a temp_b +	66%	675
T2	temp_a temp_b -	85%	1344
T3	temp_a temp_b *	85%	3292
T4	temp_a temp_b /	82%	4025

Table 6: Template wise Accuracies

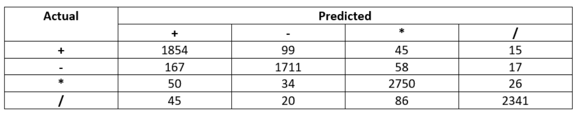
of accuracy with equation length for the Generator model. We observe that the generation of short equations is highly accurate compared to lengthy equations with multiple unknown literals. This may be due to the difficulty of the model in learning long term dependencies. Also, frequency of equations of length 13 is significantly low in the training data compared to that of short equations.

As noted in table 1, there are various types of equation templates in the dataset. Table 6 shows template wise accuracy for equations of length 3 for the Generator model. We observe that the templates with subtraction operator are more accurately generated compared to templates with addition operator. Apart from the availability of training data, our analysis on the textual questions for the template type T2 shows that word indicators of minus operator like subtract, rest, remaining, left are frequently occurring in the questions. But, we didn't observe this trend in the questions of plus operator templates T1. Thus, semantic understanding of the problem text may be weak, and the model may not be able to adequately math the numerical quantities in the problem text.

Further analysis of the Generator model shows that it is facing difficulties in learning examples which have multiple literals in the text whereas the actual equations depend only on a subset of them. As an example, textual question can have multiple literals such as temp_a through temp_j. However, the actual equation may only rely on literals temp_a and temp_b for calculation. In this case, the model has to identify the significant literals among the available literals before generating the equation.

Table 13 shows the confusion matrix of the Predictor model for the test dataset of LAMP32K. The classes are imbalanced with '-' having the least amount of data. We observe that '-'is the most confused class. '+' is mostly confused with '-' and '-' is mostly confused with '+'. We observed that phrases like "how many" and "total" frequently appear in both the classes which might have led to this confusion. Similarly, '*' and '/' are mostly confused with each other. Again, we see that phrases like "how many times" frequently appear in both these classes which may be the reason for this confusion.

From the above two observations, we notice that the Generator model performs the best in predicting templates with '+' operator whereas the Predictor model performs the worst in predicting the operator '-'. This may be one of the reasons for the weak performance of GenPred model that combines both the Generator and Predictor modules when compared to the individual models.



Actual	Predicted			
	+	-	*	/
+	1854	99	45	15
-	167	1711	58	17
*	50	34	2750	26
/	45	20	86	2341

Figure 13: Predictor Model- Confusion Matrix

4.4 Work Division

Each member's role has been crucial in the course of this project. As we moved forward with model development, results and error analysis, we clearly assigned tasks amongst all members equally. While Ramya worked on embedding the language agnostic part and an intelligent preprocessing, Sheryl worked on the Generator model and Shrija on integrating the generator and predictor. Infact, all members collaborated to develop the codebase. Once done, each took up one model and started tuning the hyperparameters. Sheryl then worked on developing the baseline models, Ramya worked on template error analysis and Shrija on model evaluations. All members collaborated in synthesizing the experiment results and putting together the final report.

5 Conclusion

In this paper, we proposed the integration of a multilingual word embedding layer across three RNN-based models in order to develop robust, language-agnostic MWP solvers. In addition to building a multilingual MWP corpus, LAMP32K, we investigated and compared 3 RNN-based models – Generator, Predictor, and GenPred. At a high-level, our study shows that it is possible to build language-agnostic MWP solvers without compromising on accuracy. With the integration of USE, we show that transfer learning via multilingual word embeddings achieves strong performance on multilingual MWP datasets. Even though the multilingual embeddings themselves are evaluated on retrieval tasks, we conclude, based on our results, that they perform reasonably well on MWPs. To the best of our knowledge, this is the first exploration of universal language agnostic sentence embeddings in the space of RNN-enabled, template-based automatic math word problem solvers.

Our study revealed that, while all three models surpassed the baseline, the Predictor Model had the highest test answer accuracy by a significant margin. However, we note that the task being fulfilled by the Predictor Model is less intricate than the task completed by Generator and GenPred. The difference lies in the input accepted by each model, given that the latter two models only take in the templatized text. We find that our Generator model is better suited for shorter equations with fewer literals. We also conclude that solving MWPs in two different stages is ineffective, given the poor performance of our hybrid model.

While there is no other research that exclusively focuses on multi-lingual MWP solvers, we evaluate the answer accuracies achieved by components of T-RNN, from which Generator, Predictor, and GenPred are derived from. We notice that the Generator and Predictor models attain reasonable accuracies relative to these two components of T-RNN, which have been trained on far less data composed of a single language.

We are confident that this research topic in the problem space of language-agnostic MWP solvers will continue to grow in a direction with singgificant impact. The models we have explored provide a promising framework for further analysis. In the future, we plan to improve these models by optimizing for multi-variable equations with multiple unknowns and cross-apply them across other languages.

References

Mikel Artetxe and Holger Schwenk. 2019. Massively multilingual sentence embeddings for zero-shot cross-lingual transfer and beyond. *Transactions of the Association for Computational Linguistics*, 7:597–610.

Danqing Huang, Shuming Shi, Chin-Yew Lin, Jian Yin, and Wei-Ying Ma. 2016. 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 Computa-*

tional Linguistics (Volume 1: Long Papers), pages 887–896.

Lei Wang, Yan Wang, Deng Cai, Dongxiang Zhang, and Xiaojiang Liu. 2018. Translating a math word problem to an expression tree. *arXiv preprint arXiv:1811.05632*.

Lei Wang, Dongxiang Zhang, Jipeng Zhang, Xing Xu, Lianli Gao, Bing Tian Dai, and Heng Tao Shen. 2019. Template-based math word problem solvers with recursive neural networks. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (Volume 33: No. 01)*, pages 7144–7151.

Yan Wang, Xiaojiang Liu, and Shuming Shi. 2017. Deep neural solver for math word problems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 845–854.

Yinfei Yang, Daniel Cer, Amin Ahmad, Mandy Guo, Jax Law, Noah Constant, Gustavo Hernandez Abrego, Steve Yuan, Chris Tar, Yun-Hsuan Sung, et al. 2019. Multilingual universal sentence encoder for semantic retrieval. *arXiv preprint arXiv:1907.04307*.

Dongxiang Zhang, Lei Wang, Luming Zhang, Bing Tian Dai, and Heng Tao Shen. 2019. The gap of semantic parsing: A survey on automatic math word problem solvers. *IEEE transactions on pattern analysis and machine intelligence*.