Automatic Generation of Elementary Level Mathematical Questions

P. L. V. S. Keerthisrini

188100U

Thesis/Dissertation submitted in partial fulfillment of the requirements for the degree Master of Science in Computer Science and Engineering

Department of Computer Science & Engineering

University of Moratuwa

Sri Lanka

January 2020

DECLARATION

I declare that this is my own work and this dissertation does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature: Date:

The above candidate has carried out research for the Masters thesis/Dissertation under my supervision.

Signature of the Supervisor: Date:

ACKNOWLEDGEMENTS

I am sincerely grateful for the advice and guidance of my supervisor Dr.Surangika Ranathunga. Without her help and encouragement this project would not have been completed. I would like to thank her for taking time out of her busy schedule to be available anytime that was needed with help and advice.

I would also like to thank my progress review committee, Professor. Sanath Jayasena and Dr. Amal Raj. Their valuable insights and guidance helped me immensely.

I would like to thank the entire staff of the Department of Computer Science and Engineering, both academic and non-academic for all their help during the course of this work and for providing me with the resources necessary to conduct my research.

The funding for this project was provided by the Senate Research grant, University of Moratuwa.

Finally, I wish to convey my sincere appreciation to Mr. Ruchira Jayathunga, Ms. Kulakshi Fernando, Ms. Ishadi Jayasinghe, Ms. Rameela Azeez, my family and all my friends for all the love and support.

Thank you!

ABSTRACT

Mathematical Word Problems (MWPs) play a vital role in mathematics education. An MWP is a combination of not only the numerical quantities, units, and variables, but also textual content. Therefore, in order to understand a particular MWP, a student requires knowledge in mathematics as well as in literacy. This makes it difficult to solve MWPs when compared with other types of mathematics problems. Therefore, students require a large number of similar questions to practice. On the other hand, the composition of numerical quantities, units, and mathematical operations impel the problems to possess specific constraints. Therefore, due to the inherent nature of MWPs, tutors find it difficult to produce a lot of similar yet creative questions. Therefore, there is a timely requirement of a platform that can automatically generate accurate and constraint-wise satisfied MWPs.

Due to the template-based nature of existing approaches for automatically generating MWPs, they tend to limit the creativity and novelty of the generated MWPs. Regarding the generation of MWPs in multiple languages, language-specific morphological and syntactic features paves way for extra constraints. Existing template-oriented techniques for MWP generation cannot identify constraints that are language-dependant, especially in morphologically rich yet low resource languages such as Sinhala and Tamil.

Utilizing deep neural language generation mechanisms, we deliver a solution for the aforementioned restrictions. This thesis elaborates an approach by which a Long Short Term Memory (LSTM) network which can generate simple MWPs while fulfilling above-mentioned constraints. The methodology inputs a blend of character embeddings, word embeddings, and Part of Speech (POS) tag embeddings to the LSTM network and the attention is produced for units and numerical values. We used our model to generate MWPS in three languages, English, Sinhala, and Tamil. Irrespective of the language, the model was capable of generating single and multi sentenced MWPs with an average BLEU score of more than 20%.

Keywords: Multi-lingual Mathematical Word Problem generation; Natural Language Generation; Neural Networks; Embeddings;

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

MWP	Mathematical Word Problem
RNN	Recurrent Neural Network
LSTM	Long Short Memory Network
CL-LSTM	Character Level Long Short Memory Network
WL-LSTM	Word Level Long Short Memory Network
RL	Reinforcement Learning
GAN	Generative Adversarial Network
MLE	Maximum Likelihood Estimation
OOV	Out-Of-Vocabulary

Chapter 1

INTRODUCTION

1.1 Background

A Mathematical Word Problem (MWP) is 'a mathematical exercise, where significant background information on the problem is presented as text rather than in mathematical notation' [4]. Solving an MWP requires knowledge in mathematics as well as in comprehension. This makes it difficult to solve MWPs when compared with other types of mathematics problems. Therefore, students require a lot of similar questions to practice. But due to the inherent nature of MWPs, tutors find it difficult to produce a lot of similar, yet creative MWPs. Therefore, there is a timely requirement of a platform that can automatically generate MWPs.

Natural Language Generation (NLG) is a proliferating technique that facilitates machines to generate meaningful and human-readable natural language text. NLG is popular among many domains such as spoken dialogue systems [5], story generation [6], lyric generation [7], question generation [8] and news generation [9]. NLG can be performed by template-based approaches or by machine learning approaches in which a model is trained using existing data. Currently, the latter approach is mushrooming with the advancements available in Deep Learning (DL).

1.2 Problem & Motivation

MWP generation contrasts from general NLG, due to the fact that they contain many inherent features. These features pave the way for many constraints as listed below,

1. Constraints related to the quantities used.

- eg: Kamal had 10 balloons and he gave Fred 4 of the balloons, how many balloons does he now have?

- In this question, the first numeric figure must be higher than the second numeric figure.

2. Constraints related to the units applicable.

- eg: Amal made bread and he used 12l water and 15kg flour. How much less water than flour did Amal use?
- Here, appropriate units should be used (Example: l for water and kg for flour).

3. Combination of ingredients/ materials should be chosen appropriately.

- eg: Mia built a house and she used 9 kg cement and 4 L water. How much more cement than water did Mia use?
- Regarding this example cement and water are a couple of materials required for building houses.

4. MWPs should not invalidate mathematical concepts.

- eg: The total of two integers is 52. Dividing the larger number by the smaller number yields 1 with a remainder of 8. What are the 2 integers?
- The numerical values chosen in the above question should be able to produce a couple of simultaneous equations, which once solved will give two integers as the answers.

Therefore, the generation should be handled carefully for its own inherent constraints.

Furthermore, MWPs possess language specific constraints. For an example, consider the MWP, 'In a car parking area there are 40 cars as blue and red cars. 15 of them are red. How many blue cars are there?'. This problem can be demonstrated in Sinhala and Tamil languages as follows,

In Sinhala language:

- වාහන නැවැත්වීමේ ස්ථානයක නිල් සහ රතු කාර් වලින් කාර් 40 ක් ඇත. ඒවායින් 15 ක් රතු ය. නිල් කාර් කීයක් තිබේද?
- Transliteration: vāhana nævætvīmē sthānayaka nil saha ratu kār valin kār 40 k æta. ēvāyin 15 k ratu ya. nil kār kīyak tibēda?

In Tamil language:

- கார் பார்க்கிங் பகுதியில் நீல மற்றும் சிவப்பு கார்களில் 40 கார்கள் உள்ளன. அவற்றில் 15 சிவப்பு. எத்தனை நீல நிற கார்கள் உள்ளன?

- Transliteration: Kār pārkkin pakutiyil nīla marrum civappu kārkaḷil 40 kārkaḷ uḷḷana. Avarril 15 civappu. Ettanai nīla nira kārkaḷ uḷḷana?

Therefore, we can easily visualize that the language structures are different. For an example, '40 cars' in English is represented as ' කාර් 40 ක් ' and '40 கார்கள்' in Sinhala and Tamil, respectively. Therefore, there is a requirement of a language independent model for automatic MWP generation.

Already available methodologies for MWP generation are semi or fully template-based [10; 11; 12; 13; 14; 15] which force the generated problems to follow specific patterns, compromising the novelty and creativity of the generated problems and making them language-specific.

As mentioned earlier, deep learning has been used in many other domains such as spoken dialogue systems [5], story generation [6], lyric generation [7], question generation [8] and news generation [9]. Any mechanism to employ deep learning approaches to satisfy the aforementioned MWP constraints has not been explored.

1.3 Objectives

The objectives of this research are to,

- Implement a language-independent deep learning based MWP generation mechanism that is tailored to satisfy the MWP specific constraints, and

- Enhance the performance of the implemented system by introducing linguistic input features.

1.4 Methodology

Following were carried out,

- Explore different Neural models for NLG and experiment MWP generation using those existing models and evaluate the outputs that each model produce, and choose

the best model for MWP generation.

- Apply attention mechanism on numerical values and units to improve constraint satisfaction of the generated problems.
- Incorporate input features such as Part of Speech (POS) embeddings, character embeddings, and pre-trained word embeddings for the DL model we used for MWP generation to enhance the accuracy of its outputs.
- Deliver an end to end system incorporating the aforementioned features.

1.5 Contribution

Following contributions are reported,

- Delivered the first DL-based approach for MWP generation that satisfies constraints.
- Created MWP datasets in English, Sinhala and Tamil languages. We have created two types of MWPs as simple MWPs and algebraic MWPs and all these datasets are publicly released [1].

1.6 Publications

- Vijni Liyanage, Surangika Ranathunga, "A Multi-language Platform for Generating Algebraic Mathematical Word Problems", In 2019 ICIIS (14th IEEE International Conference on Industrial & Information Systems) **Published**
- Vijni Liyanage, Surangika Ranathunga, "Multi-lingual Mathematical Word Problem Generation using Long Short Term Memory Networks with Enhanced Input Features", In 2020 LREC (12th edition of the Language Resources and Evaluation Conference) **Published**
- Vijni Liyanage, Surangika Ranathunga, "Algebraic Mathematical Word Problem Generation using Neural Language Generation Models", In 2019 NeuralIPS workshop WiML (14th Women in Machine Learning workshop)

[1] `https://github.com/vijini/MWP_generation.git`

- Was **awarded** with a travel grant

Accepted abstract

1.7 Organization

The rest of the thesis is arranged as follows. Background is presented under Chapter 2. Chapter 3 presents the past work done in the area of NLG and MWP generation. Chapter 4 presents the methodology we used in our NLG model. Chapter 5 describes the evaluation metrics and associated results. We conclude this document in Chapter 6 with a discussion on the results obtained.

Chapter 2

BACKGROUND

2.1 Overview

Automatic MWP generation can be regarded as a sub-area under Natural Language Generation. NLG is a domain that uses many of the advancements in deep learning techniques. First, we discuss different generic models for neural text generation such as Auto-regressive models, Reinforcement Learning models, and Generative Adversarial Networks. Then the limitations of these models are discussed. Finally, the optimization techniques that can be applied to the models are discussed.

2.2 Auto-regressive Models

Recurrent Neural Networks (RNN) [16] are referred to as maximum likelihood estimation (MLE)-based or auto-regressive models. A simple type of densely connected neural network can be considered as a recurrent neural network at its most basic level. The RNNs are distinguished from normal feedforward networks with the introduction of time. Normal feed-forward neural networks consume inputs and transform them into an output with a supervised learning mechanism. Basically, a feed-forward neural network maps patterns between inputs and outputs. A trained set of weights is used to accurately map the relationship between inputs and outputs. A feed-forward neural network considers only the inputs in the current example at a particular time. Therefore, it does not consider the order in time or keep track of previous outputs.

A Recurrent Neural Network (RNN) on the other hand considers the outputs they have produced previously in addition to the current input. They have a feedback loop to consider their past behavior. In order to keep track of previous outputs, they have introduced a novel concept called a 'hidden' state. The hidden states in RNNs facilitate modeling of long-term dependencies.

The mathematical explanation of the hidden state is depicted in the equation 2.1. Here

h_t represents the hidden state at current time t. It is a sigmoid function (ϕ) of the current input (x_t), which is multiplied by the applicable weight matrix W, added with the hidden state of the previous time step (h_{t-1}), which is multiplied by its hidden state to hidden state matrix U.

$$h_t = \phi(Wx_t + Uh_{t-1}) \tag{2.1}$$

Since the feedback loop occurs at every time step within the sequence, each hidden state refers to the previous time step, making each hidden state to contain not only the previous state but all the previous states up to a memory level the model can keep track of. To further understand this procedure, let's focus on the illustration provided in Figure 2.1. Consider the text string "A girl walked into a shop and she said, 'Can I have a pen please?'. The shop owner said, 'Certainly ....". There are many options that can fill this blank such as Ma'am, Miss, and so on. By learning through the fed sequence patterns, an RNN is capable of predicting the next word of the sequence.

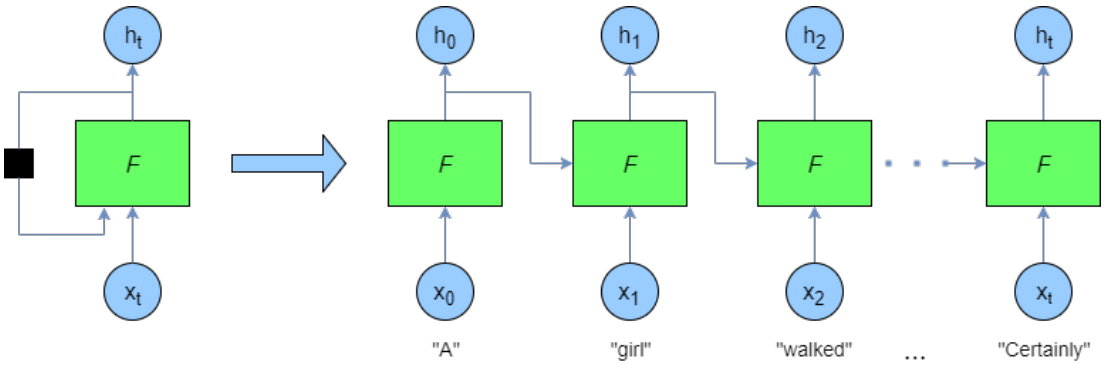

Figure 2.1: Example of a Recurrent Neural Network
Source: Keras lstm tutorial by Andy Thomas [1]

Although RNN is considered a step forward from the normal feed-forward neural network, it has some limitations. Ideally, we would require to have relationships of data with larger distances of time in between. But when there are more time steps available within the neural network, the back-propagation gradient may gather and shatter or diminish to zero. Back-propagation [17] is a training mechanism used in neural networks, which lets the neural network to learn by back-propagating the errors. Back-propagation adjusts the weights of the model such that the difference of the actual output and the desired output

is minimized. A cost function is defined to identify how much the actual output deviates from the desired output. Therefore, back-propagation aims to minimize this cost function by adjusting the weights and biases of the network. At each hidden layer, a partial derivative is calculated as $\partial E/\partial W$. Then these derivatives are used by the gradient descent learning rule to adjust the weights so that the error E is minimized. With the chain rule, the final partial derivative at each layer can be expressed as a product of previous partial derivatives. When these are lower than one, the final value will be vanishing. This issue is known as the vanishing gradient issue, which has been resolved once the Long Short Term Memory networks (LSTM) [18] are introduced.

LSTM network was introduced as a gradient-based model to resolve the aforementioned issues. Originally the LSTM was introduced with only the input and output gates and back-propagation was used as the training mechanism. Later Gers et.al [19] introduced a novel concept called the 'Forget gate', which enables the LSTM to reset its state. Forget gate decides the states that should be remembered or unremembered. This helps the models to reduce the multiplicative effect of little gradients. The structure of the LSTM is depicted in figure 2.2.

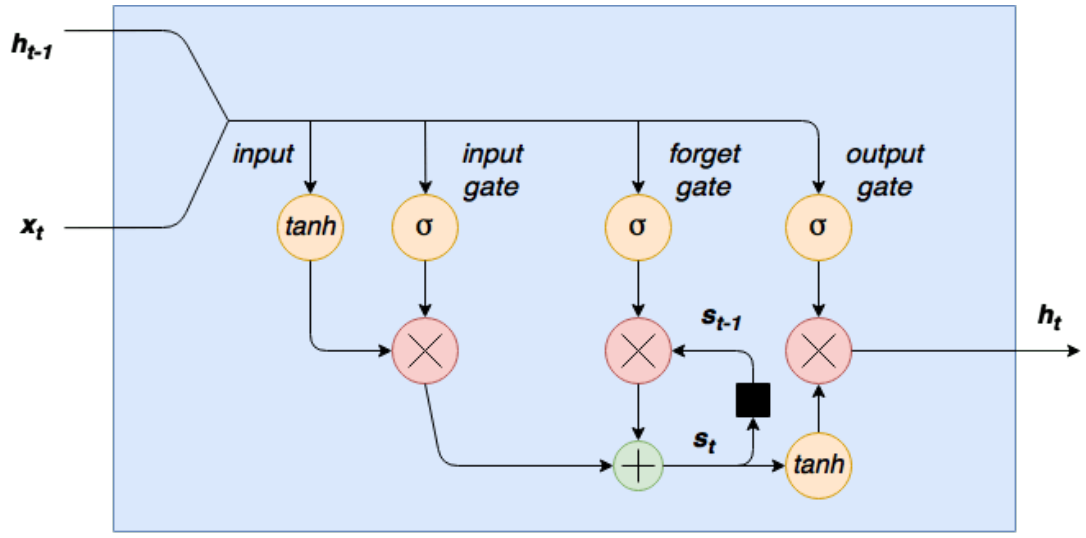

Figure 2.2: LSTM cell diagram Image
Source: Keras lstm tutorial by Andy Thomas [1]

A tanh function is used to compress the input to a value between -1 and +1 as shown in equation 2.2.

$$g = tanh(b^g + x_t U^g + h_{t-1} V^g) \tag{2.2}$$

Here U^g and V^g are weights of the current input and the previous output respectively. Input bias is represented as b^g. Equation 2.3 represents the output of the input gate, which is multiplied element-wise with the input equation. This allows the gate to determine which inputs should be switched on and off. Therefore, the output at the input stage is represented by $g\ o\ i$, where the operator o expresses element-wise multiplication.

$$i = \sigma(b^i + x_t U^i + h_{t-1} V^i) \tag{2.3}$$

Here σ denotes the sigmoid activation.

A new inner state is introduced as S_t. This state is delayed by a one-time step, S_{t-1}, and added to the output at the input stage, which is $g\ o\ i$. There is a forget gate, which is represented in equation 2.4. This is also element-wise multiplied with S_{t-1}, to determine which previous states should be remembered or forgotten. Then the forget gate output is added to the input and the output of this operation is expressed as S_t. Therefore, the current state is defined in equation 2.5.

$$f = \sigma(b^f + x_t U^f + h_{t-1} V^f) \tag{2.4}$$

$$S_t = S_{t-1}\ o\ f + \ g\ o\ i \tag{2.5}$$

Finally, there is the output gate, which is expressed in equation 2.6. This is multiplied by the compressed state S_t. Therefore, the final output of the cell is depicted in equation 2.7.

$$O = \sigma(b^o + x_t U^o + h_{t-1} V^o) \tag{2.6}$$

$$h_t = tanh(S_t)\ o\ O \tag{2.7}$$

2.3 Reinforcement Learning

Reinforcement Learning [20] is a widely used machine learning technique by which an agent should identify what action needs to be taken next, given the facts of the environment. The behavior of the agent is decided in terms of the rewards and the punishments (negative rewards) it receives. In simple terms, reinforcement learning is a goal oriented algorithm in which the aim is to maximize the objective function. Here the agent is not instructed of what action should be taken next. Instead, the agent should discover what action it needs to take in order to get the highest reward. Therefore, reinforcement learning is different from supervised learning, because supervised learning [21] is a process of learning from a training set using labeled data. On the other hand, we cannot state that reinforcement learning is unsupervised learning, because unsupervised learning [22] is a process to identify the structures hidden among unlabelled data.

Other than the agent and the environment, reinforcement learning has some elements such as a policy, a reward, a value function, and a model of the environment. A policy is used to define the behavior of a particular agent at a given time. Therefore, a policy can be considered as a mapping between the recognized states of the environment and the actions. A reward means the goal in a reinforcement learning environment. The agent has an objective to maximize its rewards. A value function is used to determine the value of being in a state. The value of a state means the cumulative reward the agent is capable of gaining at that particular state. Finally, the model of the environment is used to gather inferences on the behavior of the environment. That is, when the state and action are provided, the model will be able to predict the next state and the reward.

Reinforcement learning can be considered as a continuation of state-action pairs that occur sequentially. An example of an objective function can be demonstrated as given in equation 2.8. As shown in the equation, the reward function r can be calculated for t number of time steps. Here $x(t)$ provides the state at time step t, while $a(t)$ provides the action at that particular time step.

$$\sum_{t=0}^{t=\infty} \gamma^t \, r\,(\,x(t),\, a(t)) \tag{2.8}$$

Reinforcement learning is a process where the agent goes through sequences of state-action pairs. Depending on the rewards it gains and adapting to the predictions made by policy function, the agent tries to predict the best path it can take. After some time, reinforcement learning repeats the actions that lead to rewards by skipping other alternatives. There is a trade-off between the exploitation of known rewards and the exploration of new actions to be taken.

Reinforcement learning uses a Markov model [23], which is a mathematical model that assumes the future states only depend on the current state, not the previous states. Therefore, when there is a requirement for sequential data modeling, reinforcement learning will have its own limitations. Furthermore, one of the major problems with reinforcement learning is that it requires a large amount of data and days of computations, which will make the modeling expensive.

2.4 Generative Adversarial Networks

Generative Adversarial Networks (GAN) [24] is a novel paradigm introduced under deep learning. A GAN is comprised of two competing neural networks, the Generator and the Discriminator. The Generator is responsible for producing the outputs, while the Discriminator is responsible for evaluating the generated data for authenticity. That is, distinguishing the generated output data from the original data.

$$V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[logD(x)] \; + \; \mathbb{E}_{z\sim p_z(Z)}[log(1-D(G(Z)))] \tag{2.9}$$

As presented in the equation 2.9, $V(D,G)$ represents the value function of the GAN. Here G and D stand for the Generator and the Discriminator, respectively. $Pdata(x)$ stands for distribution of original data, while $P(z)$ stands for the distribution of the generated data. $D(x)$ and $G(z)$ represent Discriminator and Generator networks, respectively. There is a mini-max game in GANs, where the Discriminator is trying to maximize its reward, while the Generator is trying to minimize the Discriminator's reward.

The training stage of a GAN is comprised of two phases. Within the first phase, the Discriminator is trained on real data, and it checks whether it can identify them as real and trained on generated data and checks whether it can identify them as fake. Within

this phase, the Generator is not doing any task. The second phase of training is where the Generator is trained, and let it try to fool the discriminator by generating data that are similar to real data. This process is continued until the generator is capable of generating data that are as similar to the real data.

The generator produces the same output repeatedly, once it figures out that output satisfies the discriminator. This issue is termed as 'Mode Collapse' [25]. Since the two networks are trying to optimize two opposing objectives, it is difficult to keep the two networks in balance, thereby making the training process slow. Also, if the generator is performing so well, then the training of the generator can fail as a result of vanishing gradients.

Chapter 3

LITERATURE SURVEY

Under this section, we provide a thorough analysis of previous research that has contributed to the domains of Natural Language Generation (NLG) and Mathematical Word Problem (MWP) generation.

3.1 Natural Language Generation

Natural Language Generation (NLG) is a process by which sequences of natural utterances are automatically created from provided structured data. NLG is comprised of six basic tasks, which are listed as below [26],

- Content determination: This step identifies which content is required to be included in the generated text.
- Discourse planning: Under this stage, a structure with the format and order of the content is defined.
- Sentence aggregation: This step determines how the texts should be combined to form the sentences.
- Lexicalization: This stage is used to identify the specific textual representations such as words or phrases that need to be selected to indicate the concepts in the considered domain.
- Referring expression generation: Domain attributes are recognized by selecting suitable words and phrases under this stage.
- Linguistic realization: As the final step, syntactic, orthographical, and morphological precisions are ensured by the application of grammar rules.

Primarily, NLG can be categorized under two main criteria as knowledge-intensive approaches and knowledge-light systems [27; 28]. Knowledge-intensive approaches require

a large amount of human effort, while knowledge-light approaches use statistical models or deep learning techniques for NLG. Moreover, NLG is considered as a Statistical Machine Translation task as well.

3.1.1 Knowledge Intensive Approaches

Knowledge-intensive methods require domain experts to extract explicit rules and to collect the correct knowledge [29]. Therefore, these approaches highly depend on human knowledge and effort. Knowledge-intensive approaches can be further classified as template-based and rule-based approaches. Some of the examples include SunTime [30], FoG [31] and PLANDOC [32] systems. Due to the restrictive nature of the knowledge-intensive systems, knowledge-light approaches have become popular over time.

3.1.2 Knowledge-light Approaches

Fundamentally, knowledge-light approaches are either statistical models or deep learning-based models. Statistical NLG systems such as Nitrogen [33; 34] and Oxygen [35], statistical n-gram based NLG systems [36], neural network based NLG systems [37], and case based NLG systems [38] can be considered as some categories of the knowledge-light approaches.

3.1.2.1 Statistical Approaches

Statistical NLG incorporates the concept of Markov chains [39] for the generation task. Here only the current word is considered to determine the next word by a probability mechanism. Therefore, by considering only the current word, this modeling technique ignores the whole preceding context. An alternative approach is proposed with the use of neural networks, which allows the model to consider the whole preceding sequence of words in a sentence when identifying the next word. Since the deep learning-based approach allows us to model non-linear relationships, it is currently considered as the most promising approach in the domain of NLG.

3.1.2.2 Deep Learning Approaches

In the recent past, deep learning-based techniques for Natural Language Generation (NLG) have become popular among many research domains including spoken dialogue systems [5], story generation [6], lyric generation [7], question generation [8] and news generation [9]. NLG models became popular because they facilitate the generation of human-readable natural language text, from structured data provided, with less human involvement.

Deep learning based NLG can be performed using three main approaches [40], [41]; 1. Maximum likelihood estimation (MLE)-based or Auto-regressive models [42], 2. Reinforcement learning (RL)-based models [43] and 3. Generative Adversarial Networks (GANs) [24].

Recurrent Neural Networks

The research done by Xu and Rudnicky [44] can be considered as the first attempt to replace statistical language modeling using artificial neural networks. They have experimented with language modeling with a single input word without hidden units, which has hindered the model identifying unigram and bigram statistics. The turning point in the history of neural network-based NLG has happened with the introduction of the research done by Bengio et al. [45]. They identified a solution to the curse of dimensionality with artificial neural networks, in which a neural network is trained with a sequence of words as input and to generate the next word of the sequence as output. Their model has been capable of conquering the state of the art n-gram models while allowing the model to consider longer contexts. All the other advanced neural models have considered the research of Bengio et al. [45] as the baseline for their models.

Merity et al. [46] have experimented with two categories of LSTMs, Character Level LSTM (CL-LSTM) that produces succeeding character and Word Level LSTM (WL-LSTM) that produces succeeding word. Here they have identified several issues with WL-LSTM such as increased computational expenditure as a result of huge vocabulary sizes and the requirement to substitute rare words with Out-Of-Vocabulary (OOV) tokens. On the other hand, since the number of tokens in CL-LSTMs increases, they are slow to process than WL-LSTMs. But Merity et al. [46] show that a Softmax function that is adjustable can improve both CL-LSTM and WL-LSTM, empowering them to achieve

state-of-the-art results and have mentioned that CL-LSTMs can produce better results than WL-LSTMs.

DropConnect [47] is an optimization technique which can be applied on the weight matrices of recurrent layers to reduce the overfitting of a neural network model such as LSTM. Many of the previous works have used a regularization mechanism known as Dropout, where a sub class of activations are made zero. But, DropConnect only makes a randomly chosen sub class of weights to zero. Therefore, DropConnect provides advantages over Dropout, because it can work without altering the formulation of RNN. Moreover, Average Stochastic Gradient Descent (ASGD) [48] is a mechanism which can be used to enhance the training process of the NLG model.

Reinforcement Learning

Reinforcement Learning (RL) can be used for NLG when NLG is considered as a statistical planning problem [49]. With respect to text generation, writing words is considered as the actions, while already written words are considered as the states. Actions will be repeated until the agent reaches the end of the sentence. After the sentence is completed, the agent receives a reward. Due to the large size of the vocabulary, choosing the best word to write is a difficult task. Policy gradient [50] is a technique that is used in reinforcement learning to handle huge action spaces like text generation. Here, a policy identifies the next action to perform (next word to write) for every state. A major limitation in this sort of high dimensional spaces is the difficulty of identifying next word to write, when there is no reward for the model.

GANs

When GANs are used for NLG, the output of the generator has to be fed to the discriminator, to minimize 1-D(G(z)), which is the loss function. Therefore, in order to reach the generator, the gradients have to undergo the picking generation, which cannot be differentiated. This paves way for an issue since back-propagation depends on the differentiability of all the layers in the network.

As a solution to the aforementioned issue, two categories of GANs for NLG are introduced [51],

- GANs that utilize algorithms in reinforcement learning (RL) for natural language

generation such as SeqGAN [52], MaliGAN [53], RankGAN [54], MaskGAN [55] and LeakGAN [56]. GAN models under this category have complex designs which make their training slow.

- GANs that do not use reinforcement learning such as TextGAN[57] and GSGAN[58], which use the soft-argmax [59] and Gumbel-softmax [60] tricks, respectively, to deal with discrete data. Since they tend to keep the original GAN loss function, they tend to face the gradient-vanishing issue.

RL with a policy gradient concept is used with GANs to overcome the above-mentioned issue. In this case, the discriminator is considered as a reward function for the agent. On the other hand, a policy gradient is used to decide the action that needs to be picked at a particular time. The policy gradient creates a distribution of probabilities over all the actions, which allows the model to pick the most suitable next word. Discriminator network considers a sentence as the input and produces a value that shows the realistic nature of the generated sentence. This output value can be used as the corresponding reward for the input sentence and it will be fed back to the generator to update its policy.

Since the policy gradient is determined by considering just a few samples, there can be a high gradient variance from one episode to another. Therefore, this paves way for an unstable training process, and the convergence can be very slow. On the other hand, the policy gradient tends to converge to a local maximum. Moreover possessing large state-action space can make it hard to explore the whole area.

An alternative was proposed using Gumbel softmax distribution [60]. Gumbel softmax distribution is used as a continuous approximation to a multi-nominal distribution parameterized in terms of the softmax function. Here a word is sampled from a multi-nominal probability distribution that is generated by the softmax function.

3.1.3 Statistical Machine Translation

NLG is considered similar to statistical machine translation (SMT) since both the processes follow the same process, which is to generate textual representations from an existing textual representation. Some research [61; 62] identified NLG as another machine translation task. Such models used SMT for generation tasks.

Bels et. al (2009) [62] has employed a phrase-based statistical machine translation model (PB-SMT) [63] for NLG. Later the Mountain system [61] used the MOSES SMT toolkit [64], which is an advanced implementation of PBSMT to generate text. Despite SMT's contribution to NLG, these systems were not as efficient as statistical NLG platforms such as Oxygen [35] and Nitrogen [33; 34].

3.1.4 Semi-automatic Approaches

Several approaches can be considered as semi-automatic because they contain features of both knowledge-intensive and knowledge light systems. One example of such approaches is an NLG system, which uses a Probabilistic synchronous context-free grammar (PSCFG) [65].

3.2 Evaluation Metrics

3.2.1 Bi-lingual Evaluation Understudy

Bi-lingual Evaluation Understudy (BLEU) [66] is a speedy, automatic evaluation metric which does not depend on the language. It counts the number of matches comparing the altered n-grams of the produced text (candidate) with that of the reference dataset, which means it is a metric that is scored at the word level. BLEU scores are calculated as BLEU-1, BLEU-2, and so on, such that an average BLEU score can be calculated finally.

Consider the following example,

- Reference sentence: Bill has 3 balloons.
- Generated sentence: Bill Bill Bill Bill.

Here, to compare the above two sentences, one can look at the generated sentence and assign each word 1, if that word appears in the reference sentence, or else 0. Then, you can get the total and divide it by the number of words in the generated sentence, which is termed as "unigram precision". But, in this particular example, since the word "Bill" has occurred 4 times, we will get a score of 1. But, it provides a misleading score. Therefore, to prevent this issue, the BLEU metric modifies the score a bit by capping the number of times to count each word based on the highest number of times it appears in any reference

sentence. Accordingly, the BLEU score will be 0.25. Specifically, we call this BLEU-1 score, since we are considering one word at a time. But, if the words are jumbled, still we can get a high BLEU score, which is not good. This issue can be mitigated by counting not just individual words, but words that occur next to each other. These are called n-grams, where n is the number of words per group. Depending on the number of words we consider, the BLEU score changes appropriately. For instance, if we use BLEU-2 for the above example, the score will be calculated considering the bi-grams. Here the BLEU-2 score will be zero since there is no single pair of words in the generated sentence that matches with a pair of words in the reference sentence.

3.2.2 Quality vs Diversity Trade-off of Deep Learning Models

Cífka et al. [67], has introduced a novel method in which the trade-off between the two dimensions quality and diversity of deep learning models is mapped. The dimension quality takes the accuracy of a particular sentence that is been generated by the model into account. Diversity is the dimension that compares the generated sentences with each other for their similarities. But as depicted in Figure 3.1 left side graph, using two different matrices for the evaluation makes it difficult to compare the results and arrive at a conclusion. Here the performance of the two models is plotted and the two models perform better in the two dimensions. Therefore, it is hard to arrive at a conclusion. Therefore, one of the latest research [2] identified the importance of the softmax temperature [68] for the trade-off between the two dimensions. When the temperature is high, a particular model is capable to generate an output of high diversity but low quality. This is because high temperature increases entropy increasing the diversity and decreases the probability decreasing the quality. In this research [2] as depicted in Figure 3.1 (middle), the quality vs diversity for two models have been plotted for a range of temperatures. This procedure is termed as *temperature sweep*. As per Figure 3.1 (right) we can see that the blue model dominates the red. Therefore, this research shows that for any desired diversity level, at a particular temperature the blue model outperforms the red model in terms of quality (and vice versa). They [2] have used temperature sweep to evaluate MLE models and GAN models and have identified that a well temperature-tuned MLE model outperforms GANs.

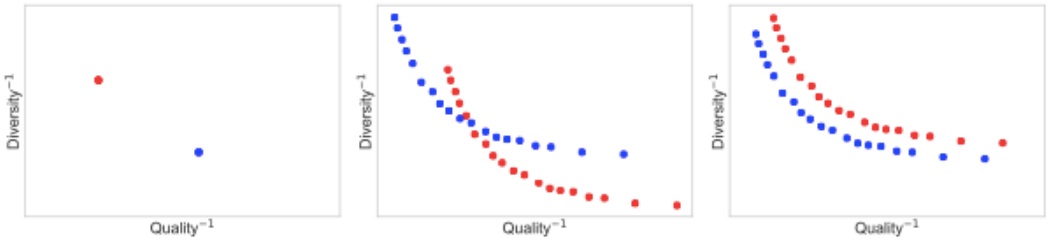

Figure 3.1: The quality versus diversity trade-off with temperature sweep
Source: Caccia et al.(2018) [2]

3.3 Mathematical Word Problem Generation

None of the aforementioned Natural Language Generation (NLG) techniques has experimented with MWPs. Since MWPs should be properly examined for the constraints related to Mathematical concepts, numerical values, units, and variable handling, constraint-based language generation is required. Existing approaches [13; 69; 14] for MWP generation are deprived of full automation since they are knowledge-heavy techniques. Therefore, the MWPs generated by such models follow similar patterns, lacking creativity and novelty.

A template-based database approach has been considered as the most common approach for automatic generation of MWPs [15; 10; 11]. Although this approach has been capable of generating MWPs, it has limitations in terms of customizability because the generated problems follow a template. Some of the template-based approaches are elaborated in the following paragraphs.

Deane and Sheehan [15] have introduced an NLG approach that uses frame semantics to generate MWPs automatically. Their templates are based on a distance-rate-time model, in which blanks should be filled by choosing items from lists. Here the connection of the mathematical content and the verbal content is demonstrated neither at the level of the individual words nor at the predetermined verbal template level. Instead, the relationship is represented within an inclusive concept like motion, which contains several items such as vehicle, distance, movement, or time. They have introduced the concept of Frame semantics as the method of representing verbal content by providing the basis for NLG and also as the generic conceptual structure for MWPs. A frame is defined as a ‘data structure to represent a standardized situation’ [70]. Filmore [71] introduced the concept of Frame semantics in which they identified the relationship between a noun and a verb

follows a set of fundamental patterns. Deane and Sheehan [15] visualized the relevance of the frame semantics for the vocabulary patterns in MWPs. That is they identified that every word problem belongs to a specific semantic frame.

For example, consider the semantic case motion. Here the following vocabulary can be identified at the most abstract level,

- The theme: the entity that is capable of moving,
- The source: the starting point of the theme's movement,
- The goal: Destination of the theme,
- The route: Mapping from the source to the goal,
- The distance: the length of the route

Then the structure should be analyzed considering the motion-time relationship as given below,

- The start time: time the source is left by the theme,
- The end time: time the goal is reached by the theme,
- The duration: time to reach the goal,
- The motion event: motion which occurs between start time and end time,
- The rate or speed: calculated by dividing distance from duration.

Thereafter the actions by which the agent's movement occurs should be considered,

- The agent: Motion occurs as a result of this,
- The instrument: The item that affects motion

Although this approach proves to be a step forward for the domain of NLG and automatic MWP generation, this research highly depends on the templates and frame semantics, thus making the generated problems to follow the same pattern.

Polozov et al. [10] have followed the same architecture and have used Answer Set Programming (ASP) [72] for MWP generating logic to assure a set of educational and

chronicle requirements. Polozov et al. are capable of generating personalized MWPs while taking into account tutor requirements (in terms of difficulty level), as well as student requirements (in terms of the character and story themes). In their approach first, they consider the set of requirements R. This can be further categorized as tutor requirements (R_T) and student requirements (R_S). Tutor requirements basically consider the pedagogical constraints within the MWPs. For example, a tutor would consider which mathematical operations should the MWP possess. On the other hand, the requirements of students can have varieties as follows,

- Setting requirements: This defines the background on which the MWP should be based on. Example: Fantasy or Fiction
- Character requirements: This defines the characters that should be included in the MWP.
- Relationship requirements: This defines the relationship between different characters.

There are two phases defined in the system presented by Polozov et al. [10], the phase that generates logic and the phase that produces language. A logical depiction is constructed under the logical phase while considering the defined requirements R. This will generate a logical graph including actors, actions, and entities. Depending on the tutor requirements (R_T), an equation E is created. Then every variable in the equation is defined based on the student requirements (R_S). This way relevant actors and actions for the plot are selected.

Under the NLG phase, the logical graph is converted to a textual representation using pre-defined templates are used and this makes the generated narratives to look alike. Therefore, they apply some post-processing techniques to make these MWPs look natural. First, there is a step to arrange sentences by which the sentences generated by the templates are arranged into a linear story referring to the casual and the temporal mappings within sentences. Then in the next step, a determination of references is done by which each entity that is referenced within the narrative is realized into a textual representation. Some of the examples for the entities are articles, pronouns, etc. Here they use a reference resolution

algorithm to make sure that every reference gets a non-repetitive representation.

Although this approach facilitates the generation of personalized MWPs, still it depends upon the templates. Therefore, we cannot conclude that this approach as a contribution to fully automatic MWP generation.

A template oriented mechanism which is semi-automated was proposed by Singh et al. [11] for algebra proof problems. This paradigm is capable of generating similar problems to a proof problem by an equation. First they syntactically generalize the problem a query. Their methodology includes three stages,

- Generating Query: Under this a query is generated for each MWP.
- Running Query: Under this a list of similar MWPs are produced. This helps in eliminating invalid problems, trivial problems, and equivalent problems out of the problems that are generated by the query generation phase. They use optimization algorithms such as polynomial identity testing [73] for this task.
- Adjusting Query: The query will be altered, if the produced MWP is unacceptable.

The first two steps are automated, while the last one is done manually. We will consider the following time series example given in equation 3.1.

$$\lim_{n\to\infty}\sum_{i=0}^{n}\frac{2i^2+i+1}{5^i}=\frac{5}{2} \tag{3.1}$$

Then a query problem is generated for this problem as given in equation 3.2.

$$\lim_{n\to\infty}\sum_{i=0}^{n}\frac{C_o i^2+C_1 i+C_2}{5^i}=\frac{C_4}{C_5} \tag{3.2}$$

Then the required query constraints are defined as below,

- $C_5 \neq 0 \wedge C_4 \neq 0 \wedge \gcd(C_4, C_5) = 1$
- $\gcd(C_0, C_1, C_2) = 1$
- $C_0 \neq 0$

Then random numbers can be assigned for these constants and generate required similar examples.

This approach makes sure new questions are formed for a given question. But we can see that the generated problems look alike. On the other hand, the system should define the constraints for every query generalization. Although they state that there is a narrative generation part, the generated problems are simply the proof problems but cannot be considered as MWPs.

Koncel et al. [12] proposed a theme rewriting approach by which the same problem is rewritten using a more interesting theme such as Star Wars. Still, this approach is incapable of generating fresh MWPs. Therefore, all these stated theme-based techniques are constricted to produce MWPs depending on the templates, making the produced MWPs to possess pre-defined patterns compromising the creativity.

Comparatively, a limited number of research has tried to generate fresh MWPs. We consider such contributions in the following paragraphs.

William [74] has used Web Ontology language (OWL) to represent MWPs. If a particular ontology has OWL Data Property Assertion statements (axioms) where integers are used to define literal values, this research is capable of creating MWPs from that. For example consider the following example for a travel ontology,

DataPropertyAssertion(
DataProperty (hasPopulation)
NamedIndividual (benbecula)
Literal (DataTypeIRI = integer 1219)

The above ontology represents that an individual named benbecula has the data property hasPopulation with an integer value of 1219. Then using SWAT tools [75], the appropriate English sentences are assigned. Then the extracted OWL statements are refactored and aggregated to compose the MWP. This research has pointed out five basic elements that facilitate automatic tuning,

- Ability to read the output.
- Incorporation of distracting numbers.

- Inclusion of extra details on the textual representation.
- The sequence of numbers.
- Conceptual complexity of MWPs.

Since the level of complication is only varied by altering the generated MWPs or by incorporating the aforementioned distractions to the MWP, this approach is capable of only producing a limited number of MWPs.

Wand and Su [76] introduced a novel concept in which expression trees are used to generate narratives to compose the MWPs. There are two phases in this research one phase for the equation generator and another phase for the narrative generator. Under the equation generator phase, there are four steps included as stated below with the example,

1. Synthesize a seed equation
   X(*du*) = Y(textitdu)
2. Assign *du* the dimensional unit *meters*
   X(*meters*) = Y(*meters*)
3. Unroll X via X_1 (*meters/ hour*) * X_2 (*hour*) = X(*meters*)
   X_1 (*meters/ hour*) * X_2 (*hour*) = Y(*meters*)
4. Unroll Y via Y_1 (*meters/ lap*) * Y_2 (*lap*) = Y(*meters*)
   X_1 (*meters/ hour*) * X_2 (*hour*) = Y_1 (*meters/ lap*) * Y_2 (*lap*)

Then under the narrative generation phase, there are also four steps as mentioned below,

1. Apply the binary expression tree: The synthesized equation is transformed into a Binary Expression Tree (BET).
2. Supplement the keywords: Each Atomic Expression Tree (AET) of the BET is traversed. Then a sub-story is assigned to each AET. Then those sub-stories are concatenated.
3. Perform auxiliary tasks: This is the stage where known values are assigned for three slots, keeping one slot blank. When assigning values, a suitable range of values is

chosen for each variable. Any other constraint in terms of values is also handled at this step.

4. Combine all the sub-stories: Since all the keywords and values are allocated, now the relevant sub-stories can be assigned.

Here the sub-stories are chosen from a database of templates. This system is capable of producing original, varied and and configurable MWPs and can be considered as an improvement when compared with the other existing systems. But, this technique strongly based on units to produce the BETs, while the templates are utilized to obtain sub-stories from produced Atomic Expression Trees.

3.4 Summary

We haven't found a prevailing system which utilizes state-of-the-art NLG techniques (discussed in section 3.1) to produce MWPs. Even Wang and Su [13] are using previously designed skeletons that are required to be filled up with different numerical values and units that are extracted from a synthesized equation. This makes the generated problems to follow similar patterns or structures. Moreover, the generation of MWPs depends on the languages that the templates are written with.

Chapter 4

METHODOLOGY

4.1 Introduction

Recent research [2; 77] have proved a temperature-tuned MLE (Maximum Likelihood Estimation) model can perform better than considered GAN models such as TextGAN [57] and LeakGAN [56]. Moreover, reinforcement learning techniques fail in identifying the next action to choose (the next word to write) when the model cannot receive a reward.

Despite above observations, we experimented with the generation of MWPs in three languages of English, Sinhala, and Tamil with both GAN and MLE models. We utilized TextGAN [57] to produce the MWPs, that includes Gumbel-softmax and soft-argmax trick, instead of RL based GANS which usually yield high-variance gradient estimates, known to be challenging for optimization [78] and possesses lengthy training periods [51].
Within the auto-regressive technique, our approach utilized both word-level (WL) and character-level (CL) LSTMs to produce MWPs. Being provided under Chapter 5, CL-LSTM could generate better results, when compared with WL-LSTM. Moreover, the CL-LSTM required a relatively small time for training. Thus, we selected CL-LSTM from all the considered models for additional enhancements.

4.2 TextGAN model

Initially, we utilized the TextGAN [57] for the generation of MWPs. It took days for the training but still could not produce meaningful MWPs. This might be due to the fact that our datasets were comparatively small and normally GANs require large datasets to learn properly. Therefore, we understood that proceeding with GANs is not suitable for our research due to the limitations in terms of datasets.

4.3 Vanilla MLE Model

The character level LSTM was utilized to train all the English, Sinhala and Tamil datasets. Per each dataset, the input-output pairs are encoded as integers. Afterwards, recognized input-output mappings are reshaped, normalized, and are one hot encoded. One hot encoding [79] is a process by which categorical variables are converted into a form that could be provided to ML algorithms. The considered character level LSTM model works as a sequential model. It incorporates two regularization techniques, the Dropout technique, which arbitrarily chooses activations and sets them to zero and the DropConnect technique, which arbitrarily chooses weights in hidden layers and sets them to zero. According to the weight improvements, the training of the model will be done within 15-20 epochs.

The generation stage initiates by arbitrarily choosing a seed text from the dataset. The seed text is of a length of 20 - 30 characters (Example: 'Ann made cookies and she'), which is equivalent to the previously chosen sequence length of the patterns. The model generates the rest of the characters that are required to form the problem starting from the endmost character in the seed sentence. The model can produce multiple word problems at an instance, based on the span of characters specified in the code.

Within NLG stage, a naive procedure makes use of greedy sampling, which is a technique that every time selects the most probable character from the softmax output of the model. However, it can destroy the creativity and freshness of the generated strings, since a greedy approach tends to produce repetitive and predictable text sequences. Thus, there is a vital requirement to incorporate randomness in the sampling procedure of the probability distribution for the succeeding character. This approach is known as Stochastic Sampling [80]. But, excessive randomness or entropy can generate characters which do not build meaningful sentences. Hence, to adjust the randomness of the text generation process, our methodology utilizes a concept known as the softmax temperature [68], which is capable of characterizing the entropy of the probability distribution utilized for sampling. When a particular temperature value is provided, a novel probability distribution is established from the initial distribution by re-weighting it. Thus, by incorporating temperature parameter concept, our model can generate creative yet realistic text sequences.

Generation of unique and fresh problems is facilitated by randomly selecting the seed

text and applying temperature tuning. Once a higher value is chosen as the temperature tuning parameter, a unique MWP will be generated. Moreover, the randomness of the outputs is facilitated by the Dropout regularization applied in the LSTM which reduces the over-fitting issue. With all these techniques, our model accomplishes the generation of innovative and fresh MWPs, when compared with the available template-oriented techniques.

As stated in section 1.2, several MWPs possess numerical constraints. For instance, examine the MWP 'Bill has 19 pens and Charlie has 16 less pens than Bill, how many pens does Charlie have?'. In this question, the first numerical figure should be higher than the second numerical figure. Several MWPs like the one shown below have breached the constraints related to numerical values.

- Ann made buns and she used 3 kg of sugar and 10 kg of flour, how much more sugar than flour did Ann use

In this aforementioned MWP, an issue regarding quantities exists. Although the MWP mentions 'more sugar than flour', the amount of sugar is less than the amount of flour.

4.4 Improvement with POS-tag based Post Processing Mechanism

Recent research [81], [82], have shown that the accuracy of NLG performed by neural networks can be further enhanced by input features like POS tags.
Hence, to recognize and satisfy constraints found in MWPs, our approach uses a lexical categorization based POS filtering technique to correct the mistakes found in English MWPs. The post-processing POS mechanism shown in Algorithm 1 is used to identify the numeric values, units, and adjectives such as 'more'or 'less', which are accompanied by the preposition 'than'. Then our algorithm compares the numeric figures with one another, in terms of the adjective and preposition composition. The contradictions identified in the generated MWPs are fixed using the algorithm. Consider the example, 'Ann made cookies and she used 3 kg of sugar and 10 L of flour, how much more sugar than flour did Ann use? '. Here, the units used are unsuitable (L represents the amount of flour). To correct those problems, our algorithm considers successive noun pairs that appear behind

the number tags and identifies whether they are equivalent with one another. POS tag filtering was not used to fix constraints identified in Sinhala or Tamil MWPs since they require language-specific rules.

```
Data: Generated question
Result: Constrained satisfied question
initialization;
tokens = word_tokenize(Data)
nltk.pos_tag(tokens)
if POS_tag_sequence.contains(2CDANDJJR) then
    if JJR = 'more' then
        while first_CD_value  second_CD_value do
            first_CD_value++
        end
        if units_dictionary.contains(NN_Bigrams) then
            Output(Data);
        else
            modify(NN_Bigrams);
            Output(Data);
        end
    end
end
```

Algorithm 1: The Algorithm to solve constraints identified in problems

Once the Algorithm 1 is applied on the generated MWPS, constraints related to units and numerical quantities were fully satisfied in the generated questions. Consider the following instance for a MWP prior to applying constraint satisfaction:

- Ann made cookies and she used 3 kg of sugar and 10 L of flour, how much more sugar than flour did Ann use?

The corrected MWP once constraint satisfaction is applied,

- Ann made cookies and she used 11 kg of sugar and 10 kg of flour, how much more sugar than flour did Ann use?

With this system, we had to identify all the specific constraints and manually define the rules required to resolve the identified constraints. Every time a new constraint was found, we had to change the rules and check for the results. The constraints found in the datasets are language dependent (refer Section 1.2 for the identified constraints) as well. Consider the following example for POS tagged sentences in different languages,

- If| IN Saran| NNP buys| VBD 16kg| CD of| IN rice| NN and| CC gives| VBZ 6kg| CD of| IN it| PRP to| TO his| PRP brother| NN, how| WRB much| JJ rice| NN does| VBZ he| PRP have| VB ?

- සරන්| NNP සහල්| NNC 16kg| NUM ක්| RP මිලදී| VNF ගෙන| VNF ,| PUNC එයින්| PRP 6kg| NUM ක්| RP මල්ලිට| NNC දුන්නේ| VP නම්| POST ,| PUNC ඔහු| PRP සතුව| VNF ඉතිරි| JJ සහල්| NNC කොපමණද| VP ?| PUNC

  (Saran sahal 16kg k miladī gena, eyin 6kg k mallīṭa dunnē nam, ohu satuva itiri sahal kopamaṇada?)

- சரன்| NN 16kg| QC அரிசி| NN வேண்டி| VM அதில்| PRP 6kg| QC தம்பிக்கு| PRP கொடுத்தால்| VM ,| SYM அவனிடம்| PRP மீதமுள்ள| JJ அரிசி| NN எவ்வளவு| RB ?| SYM

  (Caran 16kg arici vēṇṭi atil 6kg tampikku koṭuttāl, avaniṭam mītamuḷḷa arici ev-vaḷavu?)

The POS tag sequences representing the material type, its quantity, and unit combination are VBD+CD+IN+NN for English MWP, NNC+NUM+RP for Sinhala MWP, and QC+NN for Tamil MWP. Therefore, it can be seen that even the same MWP translated in different languages, does have different structures, thereby making the POS tag mappings language-specific. Therefore, it is required to separately define post-processing POS tag algorithms in a language-specific manner. This approach is represented in Figure 4.1.

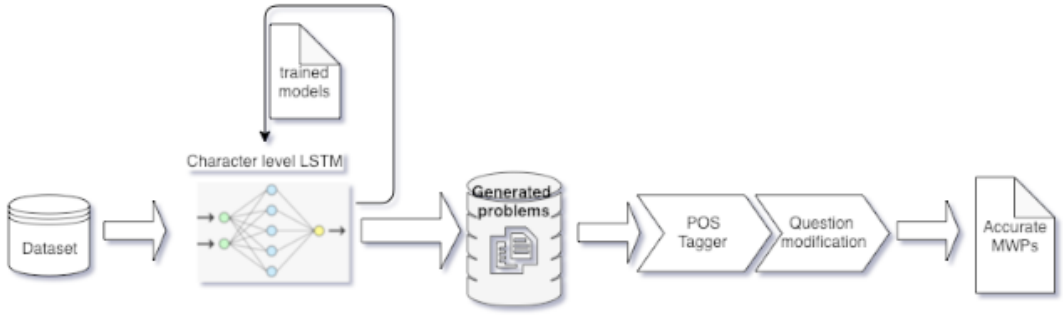

Figure 4.1: Architecture diagram of our first approach

4.5 End to End MLE Model with Attention and Different Embeddings

To eliminate the aforementioned limitations in our previous rule-based POS tag mechanism, we introduced a fully automated neural model for MWP generation in this research. In our novel approach, we have incorporated different forms of embeddings such as character, word, and POS as input features to the LSTM model that is responsible for generating MWPs. Moreover, the attention mechanism is applied to the LSTM model to resolve constraints related to units and numerical values.

4.5.1 Different forms of Embeddings

Recently embeddings play a vital role in the domain of deep learning. Since machine learning models cannot understand the text, they have to be converted to numbers. These numerical representations are known as "Embeddings". In our research, a combination of POS tags embeddings, word embeddings, and character embeddings was used as the summation of input features (I) as shown in the equation 4.1:

$$I = \sum_{j=1}^{M} \| W_j P_k (\sum_{i=1}^{N} \| C_i) \tag{4.1}$$

where $\|$ is the vector concatenation. W_j and P_k are the word embedding and POS tag embedding of each word respectively. C_i is the character embedding of each character in a particular word. i, j, and k represent the sum of characters in every word, the number of words in every sentence, and the number of POS tag embeddings defined for the dataset respectively. All these embeddings are thoroughly explained in the subsections 4.5.1.1, 4.5.1.2 and 4.5.1.3.

4.5.1.1 Character Embeddings

Since the introduction of character CNN by Xiang and Yann [3], the application of character embeddings has proved to be a prominent task in NLP. Figure4.2 demonstrates the general architecture of how CNN is used to create character embeddings.

As the first step in assigning character embeddings to a dataset, a list of characters should be defined. For instance, if we consider a dataset in English, there will be 52

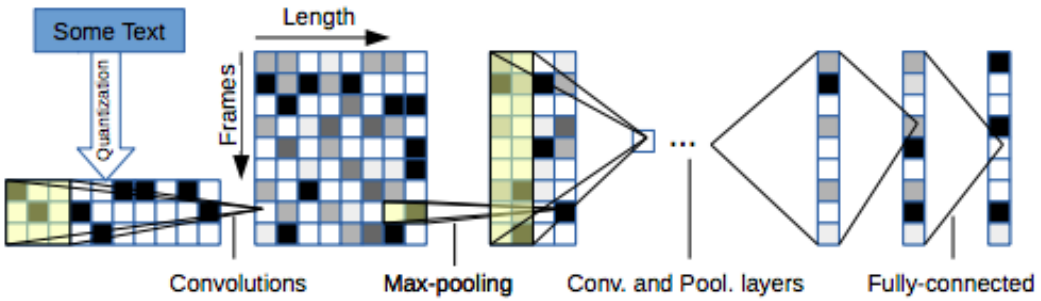

Figure 4.2: Capture from Text Understanding from Scratch [3]

English characters (including capital and simple letters), 10 numbers (0-9), 20 special characters, and one unknown character (UNK), forming a total of 83 characters. Then these characters will be transferred as one-hot encoding and a sequence of vectors will be obtained. Then a one-dimensional convolutional layer is applied to these sequences of encoded characters. The convolutional layer can be considered as a process in which several scanners are sliding through a word, character by character. These scanners are capable of focusing on multiple characters at a time. The scanners slide through the sequences of characters and extract information from focused characters. Likewise, information from several scanners is combined to form the overall representation of a word. Then a max-pooling layer is used to obtain a fixed-dimensional representation of each word. Finally, there are fully connected layers to regularize the model with techniques such as DropOut and DropConnect.

4.5.1.2 Word Embeddings

Word embeddings are assigning real-valued vector representations for words in a dataset. Words that are utilized in similar contexts receive similar representations because the distributional representations are learned based on the usage of words. There are two methods in creating embeddings, either we can freshly define embeddings to a dataset that is available for a specific task or we can use pre-trained word embeddings. The most popular pre-trained embeddings are Word2vec [83] and GloVe [84].

We used one hot encoding as well as word embeddings in our research. Although utilizing pre-trained embeddings is a time saving and easy approach, their application to a

specific task may not yield high results. In our research also we initially used Word2vec pre-trained embedding for our English datasets and since the results were low, we decided to freshly define embeddings for our datasets. For this task, we utilized the Keras embedding layer to our LSTM network which is responsible for MWP generation. Before feeding the dataset to the embedding layer, data had to be encoded and for this, we used the Tokenizer API in Keras to create one-hot encoding. Here, each embedding is formed by a set of zeros, in which 1 is assigned to the corresponding dimension. A simple one-hot word embedding for a small vocabulary of five words is shown in Figure 4.3.

	1	2	3	4	5
Bill	1	0	0	0	0
made	0	1	0	0	0
a	0	0	1	0	0
cake	0	0	0	1	0
and	0	0	0	0	1

Figure 4.3: One hot encoding example

These one hot encoded-words are fed to the embedding layer and the layer defines random weights for words initially, which are modified later during training. Once the embedding layer finishes learning weights for words, the training model can be saved for later use or directly fed to the next layer in the network. Since the output of the Embedding layer is two-dimensional, before connecting it directly to the next layer in the LSTM network, we had to flatten the output layer to a one-dimensional layer using the Flatten layer.

4.5.1.3 POS Embeddings

POS tagging is a technique by which tokens in a dataset are marked in relation to a predefined POS tag set, based on the context and meaning of the tokens. POS tags include nouns, verbs, adjectives, pronouns, and their sub-categories. POS tags have been used as input features in some research done for Neural Machine Translation [81], text-based question generation [8], and answer generation for MWPs [85]. Rajpirathap and Ranathunga [85] have used a POS tag-based feature extraction mechanism to identify whether the first

numerical value is larger than the second value. This further highlights that POS is a suitable mechanism to resolve constraints related to numerical quantities.

The POS tag mechanism defined with lexical categorization [86] was used to apply POS tags for the English MWP datasets. The POS tag set and the POS tagger introduced by Fernando et al. [87] was used to tag Sinhala language datasets while the POS tagset used by Thayaparan et al.[88] was used to tag the Tamil datasets. The POS tags are fed to the Bi-LSTM network via the embedding layer.

4.5.2 Attention Mechanism

Attention is a mechanism that is capable of allowing the decoding function to focus on specific areas in the input depending on the decoding requirement. Vaswani et al. [89] have stated attention as an integral part of sequence modeling because attention facilitates modeling of dependencies irrespective of the distance between input or output sequences. The application of attention for neural text generation has been popular [90]. In our research, we incorporated attention to improving our model by enabling the attention mechanism on numerical values and units of MWPs. Consider the MWP, 'Ann baked bread and she used 0.625kg flour and 1.25kg salt. How much less flour than salt did Ann use?'.

Here, our approach applies the attention of the two numerical values (0.625 & 1.25) and their associated units (kg).

We used the attention layer defined in Keras to apply attention to the MWPs. First, we calculated a set of attention weights for our model. Then, those weights are multiplied by output vectors of the encoder to create a weighted combination. Therefore, the result contains information about the required specific part of the input sequence (i.e. units and numerical values), and this supports the decoder to select the right output words. The attention that was applied on our model can be visualized in the Figure 4.4. As depicted in the Figure 4.4, the units focus on the type of material used and the types of materials focus on the product produced. Moreover, the word sequence 'more flour than sugar ' focuses on the numerical values.

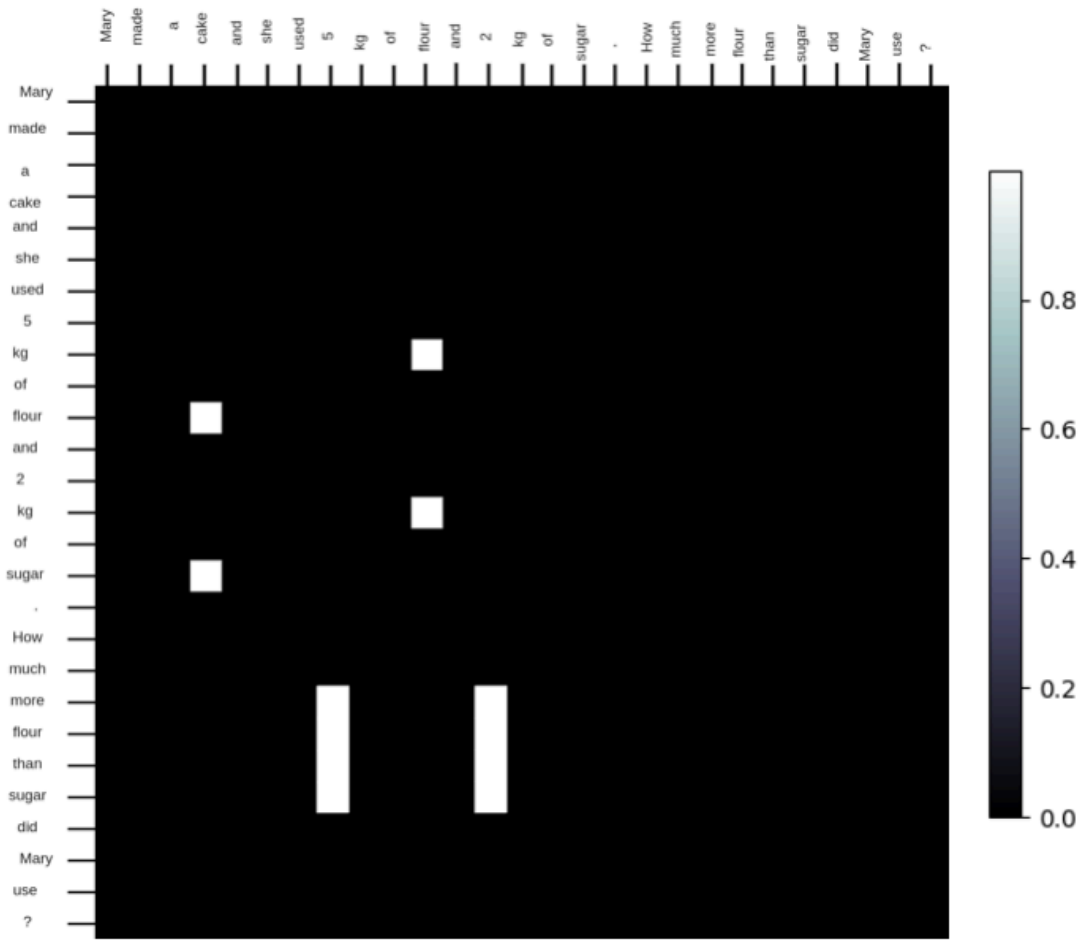

Figure 4.4: Heat map regarding the applied attention mechanism

4.5.3 Additional Improvements

4.5.3.1 Temperature Tuning

Further, we applied Temperature tuning as a mechanism to vary the creativity and novelty of the MWPs produced. Softmax temperature [68] is a hyper-parameter that is used in neural models to control the entropy of the probability distribution. As provided in Chapter 5, if the temperature parameter is set to a higher value, the self BLEU score is a low value, which means the outputs differ from the input dataset. Therefore, the creativity of the generated MWPs will be high.

4.5.4 Architecture Diagram

Overall architecture diagram our approach is presented in Figure 4.5.

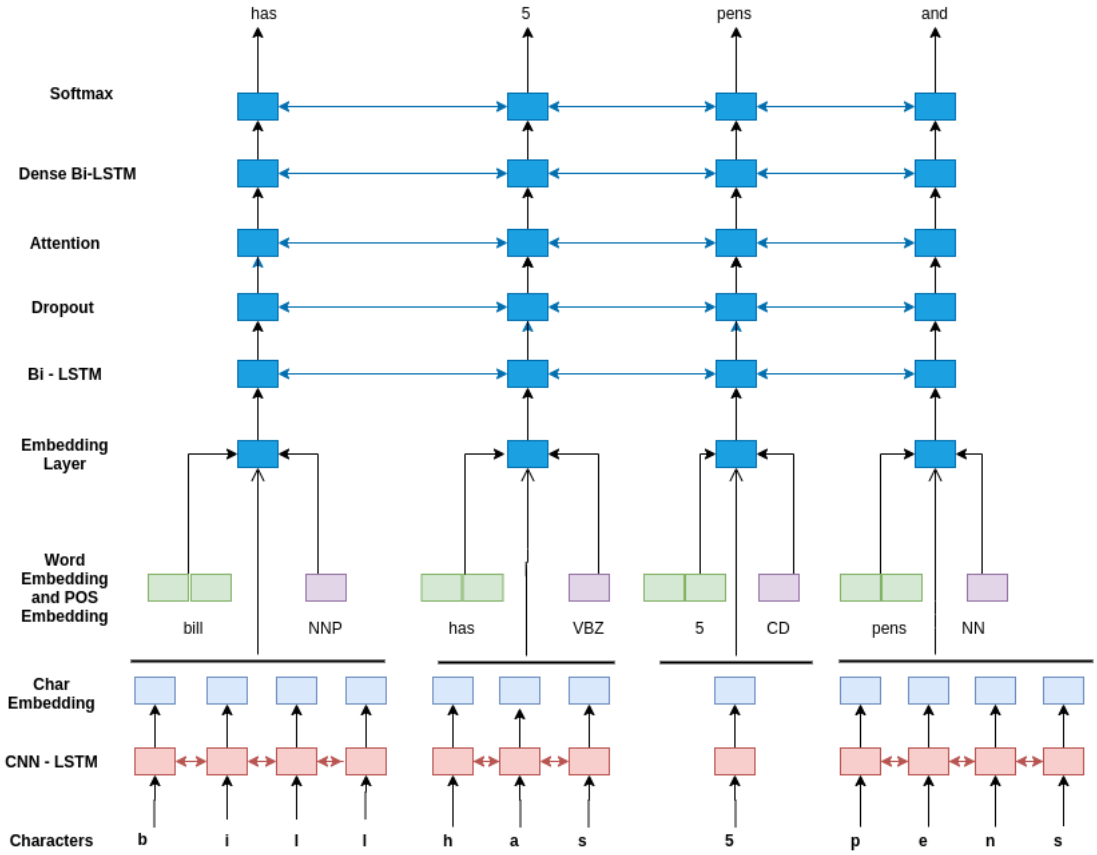

Figure 4.5: Architecture diagram of our current approach

Chapter 5

EVALUATION AND RESULTS

5.1 Introduction

Our model was evaluated by humans as well as using the BLEU score metric. Using human evaluation basically, the efficiency of the model was evaluated. We could generate 100 fresh problems for 1000 questions that were fed to the model. Then the accuracy, creativity, and efficiency were assessed in terms of the aforementioned evaluation methods. We determined that our system can generate MWPs more effectively when compared with the manual creation of MWPs. The two types of BLEU scores self vs test BLEU scores were calculated to check the novelty and accuracy of the generated problems respectively.

5.2 Dataset

As shown in Table 5.1, we used MWPs belonging to three languages, namely, English, Sinhala, and Tamil. All the questions belong to the elementary level, where each question requires simple one or two mathematical operations such as addition, subtraction, multiplication, or division. 1,878 questions of the algebraic datasets were extracted from the SigmaDolphin dataset [91]. The rest of the 472 questions, as well as Sinhala and Tamil questions, were created with the help of some undergraduate students. They have referred GCE ordinary level past papers to find similar questions and have altered them appropriately.

5.3 Human Evaluation

A group of five tutors was used to measure the improvement of our model in terms of efficiency when compared with the manual generation of Mathematical Word Problems (MWPs). We asked the tutors to correct any mistakes found in the MWPs generated by our model and measure the time taken. For each language, each tutor was asked to generate ten MWPs. Then the same tutors were asked to manually generate ten similar MWPs

Language	Question type	No. of questions	single multi-sentenced	Examples
English	Simple	1350	Single	Mary has 20 rupees and Rosy has 7 less rupees than Mary, how many rupees does Rosy have?
English	Simple	1350	Multi	Eve ran 10 miles and walked 6 miles. How much farther did Eve run than walk?
English	Algebraic	2350	Single	Find two consecutive odd integers such that three times the smaller one exceeds two times the larger one by 7.
English	Algebraic	2350	Multi	The sum of two numbers is 91. The larger number is 1 more than 4 times the smaller number. Find the numbers.
Sinhala	Simple	1000	Single	විමල්ට වයස අවුරුදු 25 ක් වන අතර ඔහුගේ මිතුරාට ඔහුට වඩා අවුරුදු 2 ක් අඩුවෙන් වයස ඇත, මිතුරාට වයස කොපමණද?
Sinhala	Simple	1000	Multi	රවි ලුණු ගෙඩි කිලෝ 9ක් ගෙනාවේය. සගයා ලුණු කිලෝ 4 ගෙනාවාය. තාත්තා ලුණු කිලෝ 9ක් ගෙනාවේය. මුලු ලුණු කිලෝ ගණන කීයද?
Tamil	Simple	1000	Single	நிமலாவிடம் 34 பைகள் உள்ளன, கமலாவிடம் நிமலாவை விட 9 பைகள் குறைவாகவும் உள்ளன, கமலாவிடம் எத்தனை பைகள் உள்ளன?
Tamil	Simple	1000	Multi	கவியின் வயது சமன் மனுவின் வயது.மதுவின் வயது சமன் றம்யாவின் வயது. கவியின் வயது 25 எனில் றம்யாவின் வயது யாது?

Table 5.1: Datasets created

and measure the time taken. This way we could compare the average times taken for manual generation of MWPs in each language and the times taken for the correction of automatically generated MWPs. The results are provided in Table 5.2. According to the

average times marked in the table, when compared with the manual generation, the MWP generation of our system had improvements of efficiencies by 87.98%, 88.66%, 91.07%, and 86.75% regarding the generation of simple English MWPs, complex English MWPs, simple Sinhala MWPs, and simple Tamil MWPs, respectively.

	TTG 10 SE MWPs	TTE 10 SE MWPs	TTG 10 CE MWPs	TTE 10 CE MWPs	TTG 10 SS MWPs	TTE 10 SS MWPs	TTG 10 ST MWPs	TTE 10 ST MWPs
Tutor 1	18	2	23	2	15	2.5	20	3.5
Tutor 2	20	2.2	27	3	25	3	19	4
Tutor 3	15	1	28.5	3.5	17.5	1.5	25	3
Tutor 4	15	2.5	22	2.4	28	1	23	2.5
Tutor 5	21	3	23	3.1	26.5	2	30	2.5
Average	17.8	**2.14**	24.7	**2.8**	22.4	**2**	23.4	**3.1**

Table 5.2: Human evaluation results in terms of TTG (Time To Generate) 10 fresh MWPs VS TTE (Time To Edit) 10 MWPs that are generated by our model
SE: Simple English, CE: Complex English, SS: Simple Sinhala, ST: Simple Tamil

5.4 Machine-based Evaluation

To evaluate the accuracy of the generated MWPs we used BLEU. We calculated the BLEU scores at each stage of our model regarding all the datasets. The results are depicted in Table 5.3, Table 5.4, Table 5.5 and Table 5.6 for simple English, complex English, simple Sinhala and simple Tamil datasets, respectively.

Model	BLEU 2	BLEU 3	BLEU 4	BLEU 5	BLEU Avg
Baseline	27.04	21.62	13.21	7.33	17.30
WP	26.98	25.02	12.91	5.87	17.70
WPC	29.37	**28.22**	13.76	9.41	20.19
WPC A	**32.93**	28.01	**16.23**	**14.71**	**22.97**

Table 5.3: BLEU Scores Generated By Various Models Concerning the Creation of simple English MWPs. WP: Word + POS embeddings, WPC: Word + POS + Character embeddings, A: Attention

With the help of the temperature sweep mechanism, we evaluated the generated MWPs concerning the quality versus diversity trade-off using test BLEU score and self BLEU

Model	BLEU 2	BLEU 3	BLEU 4	BLEU 5	BLEU Avg
Baseline	37.04	23.89	18.75	17.20	24.22
WP	43.23	25.31	9.72	9.33	21.90
WPC	45.43	31.93	26.23	19.11	30.68
WPC A	**47.84**	**37.03**	**29.42**	**19.83**	**33.53**

Table 5.4: BLEU Scores Generated By Various Models Concerning the Creation of Complex English MWPs

Model	BLEU 2	BLEU 3	BLEU 4	BLEU 5	BLEU Avg
Baseline	29.83	19.20	16.72	8.74	18.62
WP	35.73	23.42	16.88	11.73	21.94
WPC	35.23	23.56	17.72	10.02	21.63
WPC A	**39.21**	**25.51**	**21.30**	**11.95**	**24.49**

Table 5.5: BLEU Scores Generated By Various Models Concerning the Creation of simple Sinhala MWPs

Model	BLEU 2	BLEU 3	BLEU 4	BLEU 5	BLEU Avg
Baseline	22.91	12.13	7.97	0.02	10.76
WP	25.32	17.48	13.12	5.31	15.31
WPC	24.12	18.59	17.28	4.93	16.23
WPC A	**29.15**	**22.43**	**18.25**	**13.12**	**20.74**

Table 5.6: BLEU Scores Generated By Various Models Concerning the Creation of simple Tamil MWPs

score respectively. We could discover temperatures for each dataset, which optimized the test versus self BLEU score trade-off values. The graphs depicting the Negative Test-BLEU VS Self-BLEU graph for simple English, complex English, simple Sinhala and simple Tamil datasets are provide in Figure 5.1, Figure 5.2, Figure 5.3 and Figure 5.4, respectively.

5.5 Discussion

With the results we obtained, it is clear that our system is capable of generating the MWPs with an efficiency of more than 86% in all the three languages. With respect to the accu-

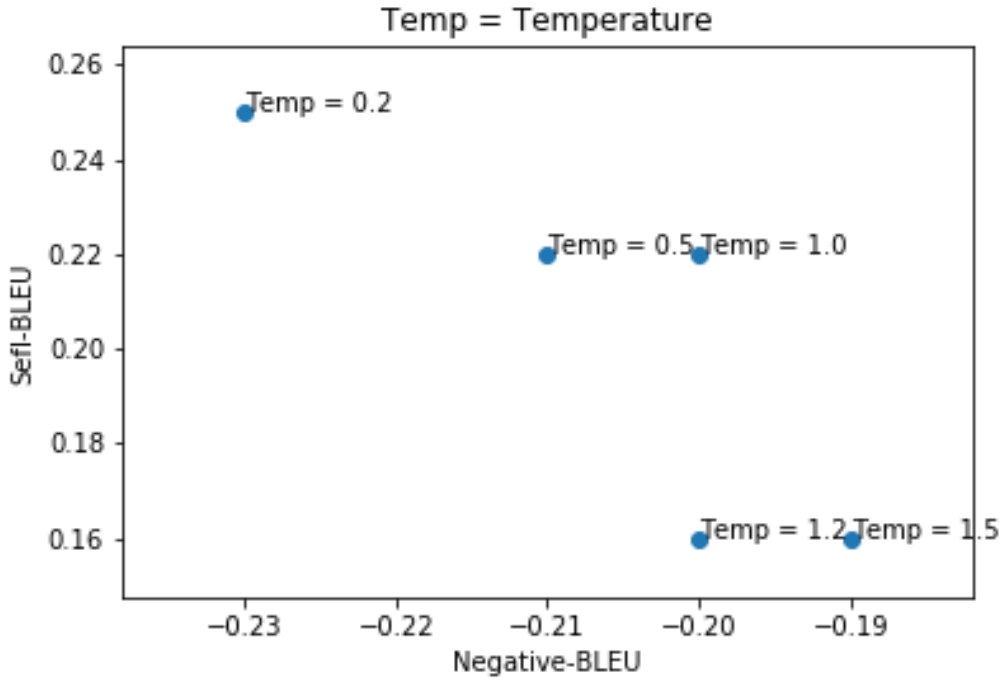

Figure 5.1: Negative Test-BLEU VS Self-BLEU graph for simple MWPs in English

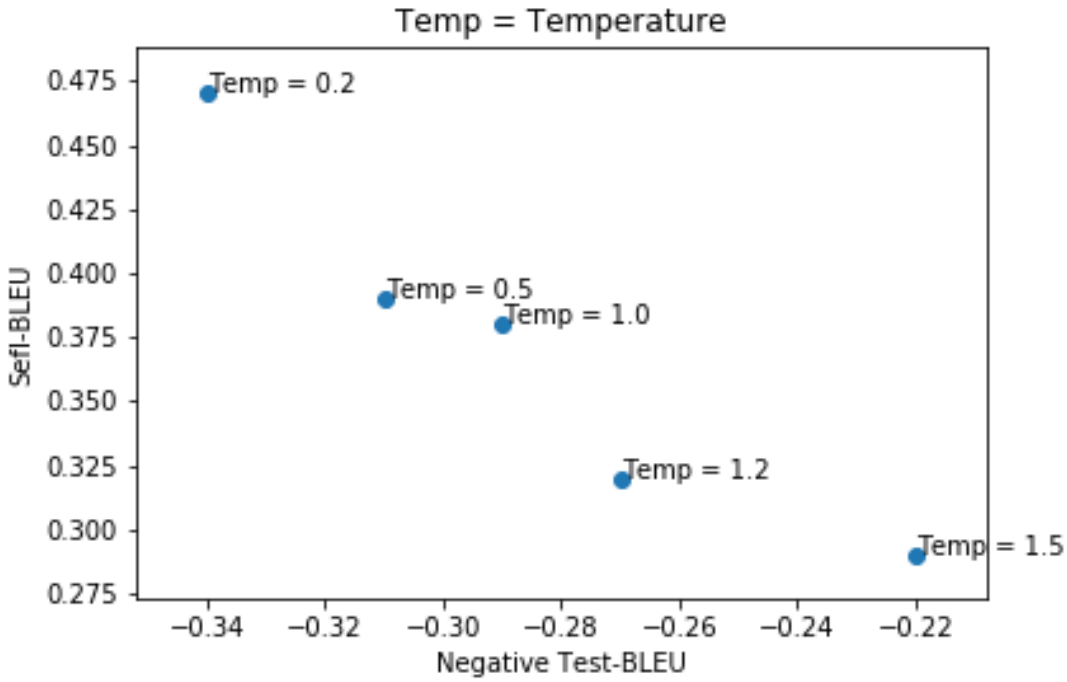

Figure 5.2: Negative Test-BLEU VS Self-BLEU graph for complex MWPs in English

racy of the generated simple English MWPs, there is an improvement of accuracy of 2.3 % in terms of the average BLEU-score when we used word embeddings and POS tag em-

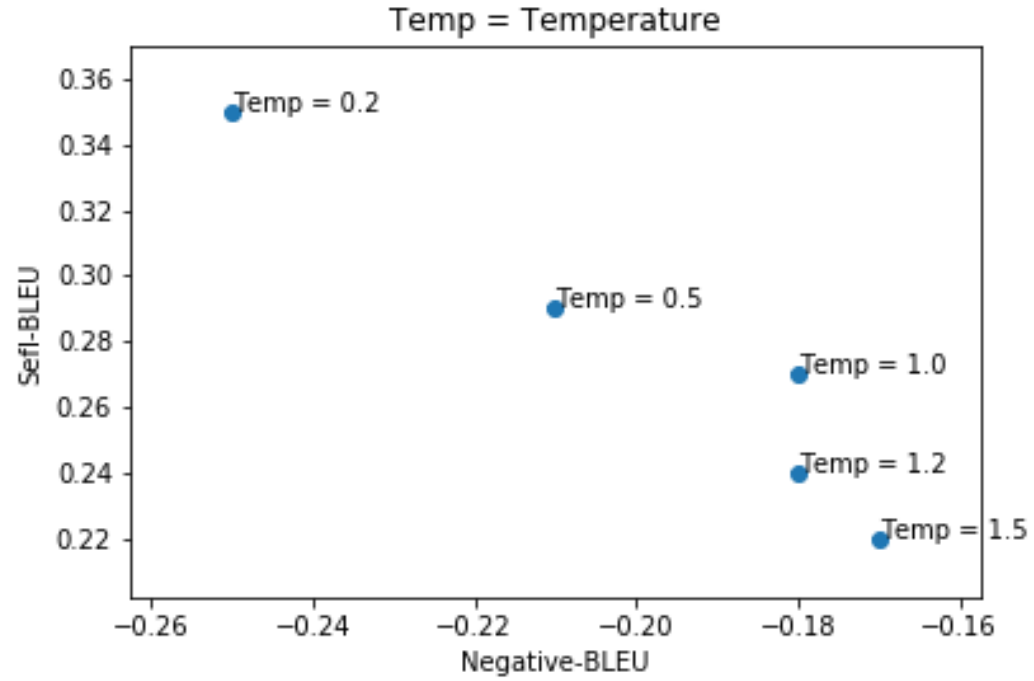

Figure 5.3: Negative Test-BLEU VS Self-BLEU graph for simple MWPs in Sinhala

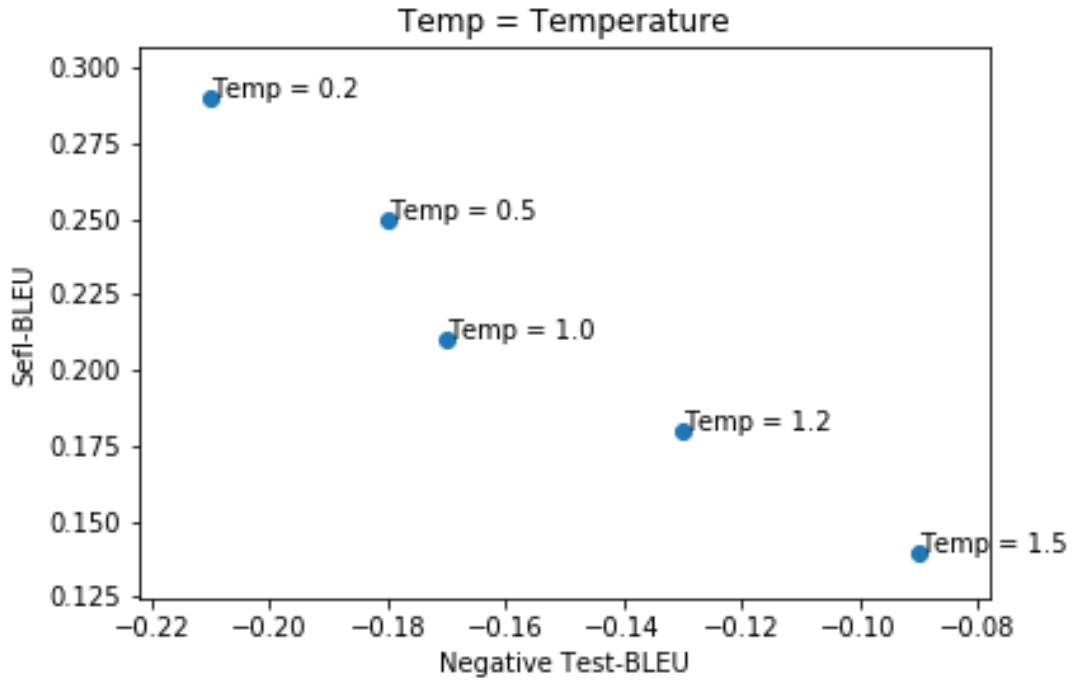

Figure 5.4: Negative Test-BLEU VS Self-BLEU graph for simple MWPs in Tamil

beddings. The average results were further improved when the character embeddings were introduced by 14.1%, than the accuracy that was gained for word embeddings. Finally the

attention on units and numerical values could improve the average results by 13.8% accuracy, when compared with the accuracy at the character embeddings level. Therefore, it is visible that the introduction of character embeddings and attention have facilitated the improvement of results by a higher proportion.

Surprisingly there was a decrease of accuracy by 9.6 % when the word embeddings and POS tag embeddings were incorporated on top of the baseline model regarding the English algebraic dataset. But the introduction of character embeddings has allowed the model to increase its accuracy by 40.1%. Then the introduction of attention has further improved the results by 9.3%. Therefore, it can be seen that the accuracy of the generated complex English MWPs was highly influenced by the introduction of character embeddings. It can be seen that the introduction of word and POS tag embeddings has reduced the results. This might be due to the reason that we used the NLTK POS tagging scheme and the complex MWP dataset contains many variables and values when compared with the contents in other datasets.

The introduction of word and POS tag embeddings could improve the results of the generated Sinhala MWPs by 17.8%, because word & POS embeddings helps the model to learn structures in MWPs. Then the incorporation of character embeddings has reduced the accuracy by just 1.4%. This might be due to the reason that Sinhala language contains many letters that are attached and when these needed to be separated to define character embeddings, the model can undergo issues. Finally the application of attention has improved the results by 13.2%. Therefore, it can be seen that the incorporation of POS and character embeddings has paved way for a large improvement in accuracy.

Concerning the Tamil dataset there is an improvement in accuracy by 42.3% when POS and Word embeddings are introduced. This huge improvement has occurred as a result of structural mapping that is provided by word & POS embeddings. Then the introduction of character embeddings has further improved the results by 6%, since character level mapping has supported the character level generation of MWPs. Finally with the incorporation of attention the results have further improved by 27.8%. This is because attention mechanism is responsible for improving accuracy in terms of units and numerical quantities.

Different datasets show different levels of improvements when the same new fature

is incorporated to the model. For an example, the introduction of word and POS embeddings have influenced on the datasets by increase of 2.3 %, decrease of 9.6 %, increase of 17.8% and increase of 42.3% for simple English, English Algebraic, Sinhala and Tamil datasets. Therefore, it can be seen that the accuracies have been increased with a large proportion for Sinhala and Tamil datasets. This might be due to the fact that Sinhala Tamil are morphologically rich languages, thus letting word embeddings play vital roles in the generation processes to capture the words. Likewise, various languages show different levels of accuracy improvements due to their inherent syntactic and semantic structures.

Chapter 6

CONCLUSION AND FUTURE WORK

Existing systems for the generation of Mathematical Word Problems (MWPs) are either fully or semi template based approaches. The generated MWPs are bereft of novelty and creativity, since they are restricted within a certain template. Moreover the template based nature prevents the existing models in becoming multi-lingual platforms to generate MWPs. On the other hand, none of the previous research has used advances in Natural Language Generation (NLG) for the automatic generation of MWPs.

In our research we have developed a language independent NLG based model to generate MWPs. We created several datasets in three languages to train the model. We analyzed latest NLG approaches such as Maximum Likelihood Estimation (MLE) approaches and Generative Adversarial Networks (GAN) and tested the MWPs generated by each model using the datasets we created. We chose the Long Short Term Memory Network (LSTM) as the baseline of our research, since it could produce more accurate results when compare with the other models.

MWPs are different to other types of text generation approaches since they contain lots of constraints. Therefore, the generation of MWPs should be capable of resolving the constraints found in MWPs. To resolve the constraints and to enhance the correctness of the MWPs that were produced, we used input features such as word embeddings, POS embedings, character embeddings as well as attention on units and numerical vlaues. The word embeddings and the character embeddings were defined by us for our datasets. By analysing the results, we could deduce that character embeddings contribute the model to generate MWPs, more than the word embeddings. This is due to the reasons such as word embeddings suffer from OOV issues and when the vocabulary is too large there are infrequent words, chances of training such words are minimized. But on the other hand, character embeddings are capable of identifying character n grams even within infrequent words. Attention was provided on numerical values and units. Attention was capable of improving constraint satisfaction, because attention mechanism focuses on the numerical

and unit-wise accuracy of each MWP. The aforementioned pre-processing steps facilitated the model to improve its results by satisfying the specific constraints found in MWPs.

MWPs contain several keywords such as ‘many’, ‘more’, ‘less’and ‘than’, that build up relationships between different variables in terms of quantities. These keywords belong to several POS tag classes. Therefore, we hope to integrate attention mechanism on different POS tag classes and inspect for the variations in accuracy level in our future work. And also we hope to utilize the latest advancements found in Reinforcement Learning based mechanisms for the generation of MWPs.

[1] Andy Thomas. *How to easily build a powerful deep learning language model*, 2019. `https://adventuresinmachinelearning.com/keras-lstm-tutorial`.

[2] Massimo Caccia, Lucas Caccia, William Fedus, Hugo Larochelle, Joelle Pineau, and Laurent Charlin. Language gans falling short. *arXiv preprint arXiv:1811.02549*, 2018.

[3] Xiang Zhang and Yann LeCun. Text understanding from scratch. *arXiv preprint arXiv:1502.01710*, 2015.

[4] J.C. Moyer, L. Sowder, J. Threadgill-Sowder, and M.B. Moyer. Story problem formats: Drawn versus verbal versus telegraphic. *Journal for Research in Mathematics Education*, pages 342–351, 1984.

[5] TH. Wen, M. Gasic, N. Mrksic, PH. Su, D. Vandyke, and S. Young. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. *arXiv preprint arXiv:1508.01745*, 2015.

[6] M. Roemmele. Writing stories with help from recurrent neural networks. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[7] P. Potash, A. Romanov, and A. Rumshisky. Ghostwriter: Using an lstm for automatic rap lyric generation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1919–1924, 2015.

[8] Q. Zhou, N. Yang, F. Wei, C. Tan, H. Bao, and M. Zhou. Neural question generation from text: A preliminary study. In *National CCF Conference on Natural Language Processing and Chinese Computing*, pages 662–671. Springer, 2017.

[9] L. Leppanen, M. Munezero, M. Granroth-Wilding, and H. Toivonen. Data-driven news generation for automated journalism. In *Proceedings of the 10th International Conference on Natural Language Generation*, pages 188–197, 2017.

[10] Oleksandr Polozov, Eleanor O'Rourke, Adam M Smith, Luke Zettlemoyer, Sumit Gulwani, and Zoran Popović. Personalized mathematical word problem generation. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[11] Rohit Singh, Sumit Gulwani, and Sriram Rajamani. Automatically generating algebra problems. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

[12] Rik Koncel-Kedziorski, Ioannis Konstas, Luke Zettlemoyer, and Hannaneh Hajishirzi. A theme-rewriting approach for generating algebra word problems. *arXiv preprint arXiv:1610.06210*, 2016.

[13] K. Wang and Z. Su. Dimensionally guided synthesis of mathematical word problems. In *IJCAI*, pages 2661–2668, 2016.

[14] S. Williams. Generating mathematical word problems. In *2011 AAAI Fall symposium series*, 2011.

[15] Paul Deane and Kathleen Sheehan. Automatic item generation via frame semantics: Natural language generation of math word problems. 2003.

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

[17] Robert Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier, 1992.

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

[19] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. 1999.

[20] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 2. MIT press Cambridge, 1998.

[21] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[22] Geoffrey E Hinton, Terrence Joseph Sejnowski, Tomaso A Poggio, et al. *Unsupervised learning: foundations of neural computation*. MIT press, 1999.

[23] Shai Fine, Yoram Singer, and Naftali Tishby. The hierarchical hidden markov model: Analysis and applications. *Machine learning*, 32(1):41–62, 1998.

[24] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[25] Tong Che, Yanran Li, Athul Paul Jacob, Yoshua Bengio, and Wenjie Li. Mode regularized generative adversarial networks. *arXiv preprint arXiv:1612.02136*, 2016.

[26] Ehud Reiter and Robert Dale. Building applied natural language generation systems. *Natural Language Engineering*, 3(1):57–87, 1997.

[27] Ibrahim Adeyanju. Generating weather forecast texts with case based reasoning. *arXiv preprint arXiv:1509.01023*, 2015.

[28] Joy Mahapatra, Sudip Kumar Naskar, and Sivaji Bandyopadhyay. Statistical natural language generation from tabular non-textual data. In *Proceedings of the 9th International Natural Language Generation conference*, pages 143–152, 2016.

[29] Ehud Reiter, Somayajulu G Sripada, and Roma Robertson. Acquiring correct knowledge for natural language generation. *Journal of Artificial Intelligence Research*, 18:491–516, 2003.

[30] Ehud Reiter, Somayajulu Sripada, Jim Hunter, Jin Yu, and Ian Davy. Choosing words in computer-generated weather forecasts. *Artificial Intelligence*, 167(1-2):137–169, 2005.

[31] Eli Goldberg, Norbert Driedger, and Richard I Kittredge. Using natural-language processing to produce weather forecasts. *IEEE Expert*, 9(2):45–53, 1994.

[32] Kathleen McKeown, Karen Kukich, and James Shaw. Practical issues in automatic documentation generation. In *Proceedings of the fourth conference on Applied natural language processing*, pages 7–14. Association for Computational Linguistics, 1994.

[33] Irene Langkilde and Kevin Knight. Generation that exploits corpus-based statistical knowledge. In *Proceedings of the 17th international conference on Computational linguistics-Volume 1*, pages 704–710. Association for Computational Linguistics, 1998.

[34] Kevin Knight and Vasileios Hatzivassiloglou. Two-level, many-paths generation. In *Proceedings of the 33rd annual meeting on Association for Computational Linguistics*, pages 252–260. Association for Computational Linguistics, 1995.

[35] John S White. *Envisioning Machine Translation in the Information Future: 4th Conference of the Association for Machine Translation in the Americas, AMTA 2000, Cuernavaca, Mexico, October 10-14, 2000 Proceedings*. Springer Science & Business Media, 2000.

[36] Alice Oh and Alexander Rudnicky. Stochastic language generation for spoken dialogue systems. In *ANLP-NAACL 2000 Workshop: Conversational Systems*, 2000.

[37] Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 1017–1024, 2011.

[38] Shimei Pan and James Shaw. Segue: A hybrid case-based surface natural language generator. In *International Conference on Natural Language Generation*, pages 130–140. Springer, 2004.

[39] John G Kemeny and J Laurie Snell. *Markov chains*. Springer-Verlag, New York, 1976.

[40] Jules Gagnon-Marchand, Hamed Sadeghi, Md Akmal Haidar, and Mehdi Rezagholizadeh. Salsa-text: self attentive latent space based adversarial text generation. In *Canadian Conference on Artificial Intelligence*, pages 119–131. Springer, 2019.

[41] Sidi Lu, Yaoming Zhu, Weinan Zhang, Jun Wang, and Yong Yu. Neural text generation: past, present and beyond. *arXiv preprint arXiv:1803.07133*, 2018.

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

[43] Hongyu Guo. Generating text with deep reinforcement learning. *arXiv preprint arXiv:1510.09202*, 2015.

[44] Wei Xu and Alex Rudnicky. Can artificial neural networks learn language models? In *Sixth international conference on spoken language processing*, 2000.

[45] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[46] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. An analysis of neural language modeling at multiple scales. *arXiv preprint arXiv:1803.08240*, 2018.

[47] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International conference on machine learning*, pages 1058–1066, 2013.

[48] Léon Bottou. Stochastic gradient learning in neural networks. *Proceedings of Neuro-Nımes*, 91(8):12, 1991.

[49] Oliver Lemon. Adaptive natural language generation in dialogue using reinforcement learning. *Proc. SEM-dial*, pages 141–148, 2008.

[50] Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*, 2015.

[51] Liqun Chen, Shuyang Dai, Chenyang Tao, Haichao Zhang, Zhe Gan, Dinghan Shen, Yizhe Zhang, Guoyin Wang, Ruiyi Zhang, and Lawrence Carin. Adversarial text generation via feature-mover's distance. In *Advances in Neural Information Processing Systems*, pages 4666–4677, 2018.

[52] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[53] Tong Che, Yanran Li, Ruixiang Zhang, R Devon Hjelm, Wenjie Li, Yangqiu Song, and Yoshua Bengio. Maximum-likelihood augmented discrete generative adversarial networks. *arXiv preprint arXiv:1702.07983*, 2017.

[54] Kevin Lin, Dianqi Li, Xiaodong He, Zhengyou Zhang, and Ming-Ting Sun. Adversarial ranking for language generation. In *Advances in Neural Information Processing Systems*, pages 3155–3165, 2017.

[55] Jiaxian Guo, Sidi Lu, Han Cai, Weinan Zhang, Yong Yu, and Jun Wang. Long text generation via adversarial training with leaked information. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[56] William Fedus, Ian Goodfellow, and Andrew M Dai. Maskgan: better text generation via filling in the_. *arXiv preprint arXiv:1801.07736*, 2018.

[57] Yizhe Zhang, Zhe Gan, Kai Fan, Zhi Chen, Ricardo Henao, Dinghan Shen, and Lawrence Carin. Adversarial feature matching for text generation. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 4006–4015. JMLR. org, 2017.

[58] Matt J Kusner and José Miguel Hernández-Lobato. Gans for sequences of discrete elements with the gumbel-softmax distribution. *arXiv preprint arXiv:1611.04051*, 2016.

[59] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[60] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[61] Brian Langner and Alan Black. Mountain: A translation-based approach to natural

language generation for dialog systems. *Proc. of IWSDS 2009, Irsee, Germany*, 2009.

[62] Anja Belz and Eric Kow. System building cost vs. output quality in data-to-text generation. In *Proceedings of the 12th European Workshop on Natural Language Generation (ENLG 2009)*, pages 16–24, 2009.

[63] Philipp Koehn, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 48–54. Association for Computational Linguistics, 2003.

[64] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th annual meeting of the association for computational linguistics companion volume proceedings of the demo and poster sessions*, pages 177–180, 2007.

[65] Anja Belz. Automatic generation of weather forecast texts using comprehensive probabilistic generation-space models. *Natural Language Engineering*, 14(4):431–455, 2008.

[66] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

[67] Ondřej Cífka, Aliaksei Severyn, Enrique Alfonseca, and Katja Filippova. Eval all, trust a few, do wrong to none: Comparing sentence generation models. *arXiv preprint arXiv:1804.07972*, 2018.

[68] M. Buscema. Back propagation neural networks. *Substance use & misuse*, 33(2):233–270, 1998.

[69] R. Singh, S. Gulwani, and S. Rajamani. Automatically generating algebra problems. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

[70] Roger C Schank and Robert P Abelson. *Scripts, plans, goals, and understanding: An inquiry into human knowledge structures*. Psychology Press, 2013.

[71] Emmon Bach and Robert Harms. Universals in linguistic theory. 1968.

[72] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Answer set solving in practice. *Synthesis lectures on artificial intelligence and machine learning*, 6(3):1–238, 2012.

[73] Jacob T Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM (JACM)*, 27(4):701–717, 1980.

[74] Sandra Williams. Generating mathematical word problems. In *2011 AAAI Fall symposium series*, 2011.

[75] Marco Ponza, Paolo Ferragina, and Francesco Piccinno. Swat: A system for detecting salient wikipedia entities in texts. *Computational Intelligence*, 2019.

[76] Ke Wang and Zhendong Su. Dimensionally guided synthesis of mathematical word problems. In *IJCAI*, pages 2661–2668, 2016.

[77] Gábor Melis, Chris Dyer, and Phil Blunsom. On the state of the art of evaluation in neural language models. *arXiv preprint arXiv:1707.05589*, 2017.

[78] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

[79] Jason Brownlee. *Why One-Hot Encode Data in Machine Learning*, 2020. `https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning`.

[80] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[81] Rico Sennrich and Barry Haddow. Linguistic input features improve neural machine translation. In *Proceedings of the First Conference on Machine Translation: Volume*

1, Research Papers, pages 83–91, Berlin, Germany, August 2016. Association for Computational Linguistics.

[82] Manex Agirrezabal, Bertol Arrieta, Aitzol Astigarraga, and Mans Hulden. Pos-tag based poetry generation with wordnet. In *Proceedings of the 14th European Workshop on Natural Language Generation*, pages 162–166, 2013.

[83] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[84] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[85] S. Rajpirathap and S. Ranathunga. Model answer generation for word-type questions in elementary mathematics. In *International Conference on Applications of Natural Language to Information Systems*, pages 17–28. Springer, 2019.

[86] Edward Loper and Steven Bird. Nltk: the natural language toolkit. *arXiv preprint cs/0205028*, 2002.

[87] S. Fernando, S. Ranathunga, S. Jayasena, and G. Dias. Comprehensive part-of-speech tag set and svm based pos tagger for sinhala. In *Proceedings of the 6th Workshop on South and Southeast Asian Natural Language Processing (WSSANLP2016)*, pages 173–182, 2016.

[88] M. Thayaparan, S. Ranathunga, and U. Thayasivam. Graph based semi-supervised learning approach for tamil pos tagging. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

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

[90] Z. Xie. Neural text generation: A practical guide. *arXiv preprint arXiv:1711.09534*, 2017.

[91] S. Shi, Y. Wang, C. Lin, X. Liu, and Y. Rui. Automatically solving number word problems by semantic parsing and reasoning. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1132–1142, 2015.