# Expression Generation for Solving MWPs Constrained by Extracted Units and Predefined Rules

*Hongyan Deng*

Master of Science
School of Informatics
University of Edinburgh
2022

# Abstract

Systems designed to solving Math Word Problems (MWPs) automatically accept math problems described with natural language, and convert or map the inputs to an equivalent structured expression to deliver final answers. Recently, researchers shift their direction and focus on applying deep learning architectures, such as seq2seq and transformer models, on MWPs solving without manual feature engineering. Despite deep neural networks (DNNs) could provide fair results on this task considering their strong generalisation ability, they are proved by evidence [35] to rely on shallow heuristics to perform well without truly "understand" the problems. Motivated by this issue, we proposed to include unit constraints and inference while training DNNs to generate equation. With the help of extracted units and context information relative to quantities, the model might gain a better "understanding" to both the question and premise part of a MWP. Nevertheless, before the development of our designed system, we have to build our own information extraction system in light of the unsatisfactory performance of the off-the-shelf tools on the MWP datasets and few research regarding the extraction of context information, such as measured entities, properties, and qualifiers.

Therefore, this project could be divided into two phase: 1) The design for deep-learning-based equation generation restricted by extracted units and predefined rules. 2) The design for quantity and unit extraction algorithm along with implementation.

## Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Hongyan Deng*)

## Acknowledgements

I would like to express my appreciations for my supervisor Shay Cohen for his patience and support throughout this project. Many thanks for answering my doubts and offering helps when I was stuck as well as granting me computing resources to conduct experiments. And thanks for all the Cohort team members who are always friendly and be willing to provide helps. I also would like to thank my family and friends for their unconditional love to support me during my master study.

# Table of Contents

# Chapter 1

# Introduction

A math word problem (MWP) describes a math problem with natural language (NL) involving semantic understanding and logic reasoning by applying mathematical axioms and theorems. An automatic MWP solving system is able to take math problems described with unstructured NL as inputs and deliver a numeric result or explanatory answers with NL description by mapping to or generating a corresponding equations. It obtains the benefits that it is not constrained by the input format and the requirement of being familiar with command-driven programming techniques compared with the command-orientated MWP solving software, such as Matlab. Even though there are lots of challenges while investigating MWP solvers regarding the lack of math-related knowledge base, the ambiguity of problem descriptions, and the misinterpretation of a generated internal structure representation [36], this field still attracts ever-increasing research attentions considering its application values. Lately, applying deep neural networks (DNNs), such as transformer, GPT-3 [6], and seq2seq model, to address MWPs are becoming ubiquitous. Benefit from the ability of generalisation for deep learning architectures, an equivalent equation could be generated from a given problem text prepared for further inference and final results delivery.

Another intriguing area is about extracting quantities, unit of measurements, and their related context, which belongs to the domain of information extraction (IE). Although extracting quantitative information is challenged by the inconsistent and ambiguous way of writings, motivated by the need of accessing numerical information in unstructured scientific paper and reports for data collection and data analysis, lots of research have been taken in this field. For instance, the approach of rule-based system [10] performing pattern-matching to decide the validity of extracted candidates, and ontology-based information extraction (OBIE) system [52] which integrates natural language into formal ontologies. According to [18], many research focus on identifying quantities, yet paying little attention on extracting measured entities, properties, units, and relative contexts, which are considered to be not only significant for constructing knowledge graphs and collecting dataset, but also essential for undertaking our project.

## 1.1 Motivation

Notwithstanding the approaches adopting DNNs could receive fair results and accept wider range of problem types without manual feature engineering, Patel et al. provide evidence to demonstrate that the reason for deep learning approaches performing well on MWP solving is their reliance on shallow heuristics. Without revealing the question part in a given MWP or word-order information, they still could get an answer for the problem by merely associating words in the problem text with equations, which indicates that DNNs are unable to "understand" the problems. Thus, their actual ability to address MWPs should be questionable. To help the deep learning architectures better "understand" the input MWPs, we are motivated to design a deep-leaning-based equation generation system restricted by the extracted units and the related context information of quantities.

Developing such unit-driven system requires the essential operation of extracting quantities, units, and their relative context information. Although a certain amount of research have been taken to investigate quantity and unit extraction, they are oriented towards scientific discourse and few of them concentrate on extracting related context of quantities, such as measured entities, properties and qualifiers. Those additional information help to generate ratio unable to be extracted explicitly. For example, given sentence "each day 2 tablets", it should indicate a ratio with unit "tablets per day", which cannot be extracted explicitly. Moreover, off-the-shelf tools and libraries for quantity and unit extraction unable to offer us satisfactory results (further discussed in details in Section 2.1.1). To remedy those defects, we are also motivated to construct a more suitable information extraction system.

## 1.2 Objectives

As we explained our motivations for constructing the template expression prediction system with unit constraints, and the necessity of building a tailor-made unit extraction system in Section 1.1, our main objectives are as follow:

1. Build a system to extract quantities and units automatically given plain texts of MWPs as inputs.

2. Design the methodology of the deep learning architecture which could predict templates and operators constrained by units.

## 1.3 Contributions

Our contributions of this project could be mainly concluded in the following list:

1. We designed the methods of predicting templates and operators with unit constraints in details.

2. We designed the approaches of extracting quantities and their relative information along with defining new annotation concepts followed by MeasEval task [18].

3. We fine-tuned an NER-orientated BERT module to extract quantities and units span types.

4. We provided detailed analysis and discussion to our experimental results as well as the suggestions of improvement.

# Chapter 2

# Background

## 2.1 Measurement Extraction

Identifying and extracting quantities and units are one of the preliminary works for constructing MWPs solvers in this project. In fact, they are mainly applied in scientific literature to distill quantitative information for further data analysis or database construction in specific scientific domains. Automatically extracting quantitative information from unstructured text is a challenging task in light of the non-standardised or ambiguous expressions of unit and measurements [4]. Although there are tools and research available to provide methods for automatic quantity and unit extraction, the additional information, such as qualifiers and modified entities, are necessary for the sake of better understanding a quantity.

Some previous research related to information extraction at linguistic level are rule-based systems involving pattern-matching and formal grammar engines [15], such as General Architecture for Text Engineering (GATE) [10]. They can only provide relatively simple semantic analysis. Some other research are focusing on integrating ontologies for natural language, which incorporates the semantic information under the current context and some extra knowledge of terms [52]. The Ontology of Units of Measure and Related Concepts (OM) [37] is one of the ontologies involving unit and measurement. Reviewed by Steinberg et al. [45], most of the approaches for ontology concentrate on the classification of measurement types or the conversion among distinguished measurement systems, however, few is about improving extraction quality. Notwithstanding Maksimov et al. [32] proposed methods of extracting not only quantities and units but also their properties by using ontology, there is still a gap between extracting all necessary context information for a quantity and existing literature.

### 2.1.1 Off-the-Shelf Tools

There is few open resource off-the-shelf library available for extracting quantities and unit of measurements from unstructured text. *Quantulum3* [1] package is one of the few. It could generate a list of quantities with corresponding values and units. In addition, it could disambiguate units with identical appearance by applying K-Nearest Neighbours on GloVe word embedding and relative Wikipedia pages. For instance, the unit “pounds” could represent either weight or sterling. *Quantulum3* uses learned vector representation to disambiguate its meanings with included context information.

Despite few resource is accessible for quantity extraction, the number of libraries manipulating units and quantities is considerable. They allow customised unit definition and manipulation to deliver a calculation result. *Numericalunits1.25* [2] could be referenced under any programming language to perform numerical calculation, dimensional analysis, and unit conversions with defined variables or physical constants. Compared with *Numericalunits1.25*, *Pint* [3] provides several different ways of defining physical quantities: either using class constructor to specify the units and magnitudes separately, or using string parser to identify the magnitudes and units in strings. It also obtains a default list of units to support unit definition and operation. The list also could be customised or appending customised units to the existing registry to capture more unit identification and manipulation. Apart from the libraries mentioned above, *Astropy* [4] and *Natu* [5] support more sophisticated manipulation, such as converting to Python scalars and supporting NumPy array calculation for list of quantities.

These off-the-shelf quantity and unit manipulation tools are adequate for unit inference and calculation while solving MWPs, however, none of the existing libraries, *Pint* or *Quantulum3*, could deliver unit extraction with satisfactory results. They are unable to deal with units excluding the range of the build-in unit list. For example, sometimes they are unable to capture quantities of count and factor type by prompting “beyond registry list” error or generating “dimensionless” unit [6] Occasionally, they even fail to identify ratio unit without the signifying of keyword “per” and also produce “dimensionless” unit for the rate. Managing “dimensionless” unit to refine unit extraction is problematic, which motivate us to build a unit extraction model more suitable for

[1]https://pypi.org/project/quantulum3/
[2]https://pypi.org/project/numericalunits/
[3]https://github.com/hgrecco/pint
[4]https://docs.astropy.org/en/stable/units/index.html
[5]https://github.com/kdavies4/natu
[6]Dimension is the dimensionality of a unit containing the name and the power of entities related to the unit.

MWP solving.

### 2.1.2 MeasEval Task

Owing to the limited data related to quantity and unit extraction accessible to the public, one openly available database found to be valuable and practical for the extraction task is the dataset attached in the MeasEval task. MeasEval task [18] belongs to SemEval task classified in counts, measurement, attributes of quantities, and relative context extraction for scientific literature. Although extracting measurements from text could be relatively easy [14], only recent research start to focus on identifying name entity and property for quantities and measurements [24]. It is a challenging task due to the ambiguity and inconsistency of quantity and unit representation, and the variability of information location related to the measurement [18]. It also obtains various application fields including the creation of Knowledge Graphs to aggregate quantitative information for assorted subjects [1], the construction of database by excavating valuable quantities and measurements automatically for biomedicine discipline [17] and so on.

The data for MeasEval task contains plain text of 110 licensed articles across 10 subject fields [7] and annotated data composed with basic annotation sets: *Quantities*, *MeasuredEntities*, *MeasuredProperties*, and *Qualifiers*. To indicate the property of a *Quantity*, quantity modifiers are utilised to denote whether it is a count (*IsCount*) or a measurement with unit (*Unit*). Modifiers also could mark additional information for the quantity value, such as *IsApproximate* and *IsRang*. The basic annotation sets could be connected by relations. *MeasuredEntity* or *MeasuredProperty* could create relationship with *Quantity* by *HasQuantity* tag. *MeasuredEntity* could link to *MeasuredProperty* by creating *HasProperty* relation. The relations of annotation model is demonstrated in Figure 2.1, and the details of annotation guidelines are available on GitHub [8].

According to the reported results of paper submissions, most of the systems are developed based on BERT architecture [12] or its derivatives such as RoBERTa [31], BioBERT [29], and SciBERT [3]. Most of the main submissions apply a pipline architecture with BIO encoding scheme for sequence tagging, and half of them use a Conditional Random Fields (CRF) model to promote prediction accuracy. The unit and quantity modifiers are annotated by applying another BERT, or a character-level BiL-

[7]https://github.com/elsevierlabs/OA-STM-Corpus

[8]https://github.com/harperco/MeasEval/tree/main/annotationGuidelines

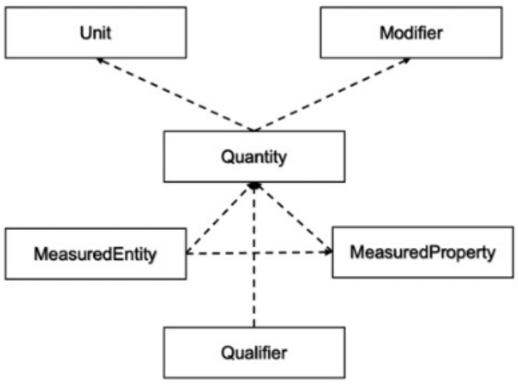

Figure 2.1: The relations of annotation model [18]. All relationships between annotation sets are optional.

STMs. The prevalent training method is to build multi-task sequence tagging model for *MeasuredEntity*, *MeasuredProperty*, and *Qualifier* annotations given the original sentence and annotated *Quantity* spans. Take the best-performed system LIORI [11] as an example. They first apply LUKE [53] to fine-tune an NER model to extract *Quantity*. Then, use pretrained model RoBERTa to perform muilti-task tagging for other spans.They extract units by applying Question Answering style sequence tagging.

### 2.1.3 Name Entity Recognition for Quantity Tagging

The task of extracting and annotating spans could be considered as a part of Name Entity Recognition (NER) task. The NER dataset Ontonotes v5 [51] encompasses the annotated spans for quantities and their measurements. Such annotated span types are analogous compared with the annotation sets in MeasEval task introduced in the previous section only with less fine-grained tags. Investigating approaches and architectures related to NER task is beneficial for quantity and unit extraction.

LUKE [53] acquires the SoTA results for NER task in 2020, which reaches 94.3 F1 scores CoNLL 03 [39] dataset. It produces contextualised representation for each word and entity considered as individual tokens in the given input text. LUKE applies pretrained model BERT to predict randomly masked words and entities in the annotated corpus as its downstream task. It also introduces entity-aware self-attention mechanism which takes the type of tokens into consideration while calculating attention scores. Last year, the proposed Automated Concatenation of Embeddings (ACE) [48] technique surpasses LUCK to reach the SoTA result with 94.6 F1 scores. Inspired by the discovery of the recent works [2] that model's performance could be further improved by concatenating non-contextualised word embeddings, pretrained contextualised embeddings, and character embeddings [13], they proposed to automatically find a better concatenated embedding for a specific structure prediction task motivated by the difficulty of embedding selection considering the everincreasing number of embedding types. They use a controller to sample embedding concatenations concurrently based on the current belief of the effectiveness for each embedding type. The belief and other parameters of a controller are optimised by performing reinforcement learning.

Apart from the SoTA models introduced before, the utilised features and architectures for NER are also significant supported by the experimental results from Schweter and Akbik [42]. They verified again that compared with sentence-level features extracted from the given text, document-level features could prominently improve the performance on NER task. They proposed that document-level features could be extracted by transformer-based models via providing a sentence itself along with 64 subtokens from its left and right context. They also compared two approaches of using transformer to perform NER task: either fine-tuning the transformer and adding a linear layer [12], or applying transformer to extract features for LSTM-CRF architecture [23]. They concluded based on their experiments that the fine-tuning approach surpass the feature-based approach. Furthermore, including CRF decoder for fine-tuning does not promote results significantly.

In addition, there are some tools and libraries to perform NER task that are also favourable for our experiments running. Take *spaCy* [9] and *flair* [10] as examples. They not only supports tokenization for multiple languages, but also contains pretrained piplines for NER, parsing and text classification tasks. Some state-of-art research are build based on them, such as the development of Automated Concatenation of Embeddings (ACE) technique [48].

## 2.2 Math Word Problem Solvers

The approaches of solving MWPs could be mainly categorised into symbolic, statistical, and deep learning approaches. Research concerning transforming the input problems into internal high-level structures, such as trees and stacks, through pattern matching or semantic parsing should be classified as symbolic approaches. They de-

[9]https://github.com/explosion/spaCy
[10]https://github.com/flairNLP/flair

liver the final answers from the corresponding equations derived by the structured representation. Comparatively, the statistical approaches could learning the mappings from the input text to the equation templates without transforming into the internal structures. They primarily apply supervised learning with feature engineering to maximise the joint log-linear probability of the matching template equations. Nonetheless, they are inadequate to solve problems with unseen template equation types and complicated logic relations. Moreover, hand-crafted feature engineering is tedious and gruelling when the problem difficulty upgrades and the number of features substantially increases. Deep learning approaches are recently introduced since 2017. They also attempt to map the given problem text to an equation by utilising the SoTA deep neural architectures, such as transformer and sequence-to-sequence model. Unlike the statistical methods, the step of manually selecting features is omitted with the complementary of generalisation ability of deep neural networks (DNNs).

### 2.2.1 Symbolic and Statistic Approaches

The symbolic approaches obtain longer generations back to the last century compared with the statistical approaches. Systems such as STUDENT [5], CARPS [7] are firstly introduced to solve elementary algebraic math problem with rule-based designs. They are only capable of addressing problems with a limited number of types due to the constraint of rules and formats in the knowledge base [34]. In 2015, Shuming Shi et al. [43] introduced SigmaDolphin system and DOL (dolphin language) which is a build-in data structure with open domain and type consistency. They applied semantic parser to convert the input text into high-level structures represented by DOL and derived expressions to generate the final results from DOL representation. EUCLID system [21] shares resemble traits with SigmaDolphin considering lazy evaluation and tree internal structures. It is able to handle problems containing more complex semantics information and sentence structures by propagating any uncertain semantic parsing through a nondeterministic tree transducer until the uncertainty has been resolved. In order to solve more difficult MWPs, Matsuzaki et al. [33] presented a hybrid end-to-end system aiming at solving pre-university math problems. After applying dependency parsing and coreference resolution on the input problem text, they used combinatory categorial grammar (CCG) to convert natural language text into logical forms and create a ranked list for the related sentence-level logical forms. The final results are generated by computer algebra system (CAS) after inferring and rewriting logical forms by applying designed axioms.

As for the statistical methods, Hossein et al. [22] presented ARIS in 2014 to solve elementary MWPs only with addition and subtraction operators. By mapping the verbs in the input text to the designed verb categories, the correspondence between discrete object and numeric could be caught. The final results are acquired by observing the increase or decrease of a quantity in a container after grounding objects and quantities in the input text to entities and containers. Kushman [28] introduced an algorithm of learning to align numbers and variables in the problem text into equation templates which is not limited with addition and subtraction compared with Hossein's algorithm. Through optimising a join log-linear distribution, it learns to line up variables and quantities in the input text to an expression template. This system is defective while there is a significant gap between the problem types of the training sets and the test sets. In order to conquer this issue, Koncel-Kedziorski et al. [25] proposed ALGES system to generating expression trees and solve algebraic word problems. Roy and Roth [38] proposed their methods to solve arithmetic word problems by performing feature selection from the input and decomposing the target problems into classification tasks. Both ALGES system and Roy's methods use estimated local likelihood to predict an internal operator by a local classifier and further affect the global scores of an expression tree generation.

### 2.2.2 Deep Learning Methods

Apart from the symbolic and statistical approaches, deep-learning-based MWP solvers started to gain increasing attentions from researchers since 2017. They obtain the main advantages of learning effective feature representation without human interference and strong generalisation ability allowing them to be adopted in various fields. Wang et al. [49] proposed Deep Neural Solver (DNS) in 2017, which is a pioneering design to apply DNNs on MWPs solving. DNS utilises seq2seq model to translate the inputs MWP text into an equivalent equation. They also introduced significant number identification (SNI) model attempting to improve the equation generation quality by discriminating numeric will be used later in equation construction. Additionally, they proposed a hybrid model combining the similarity-based retrieval system and the seq2seq model. Choosing the retrieval system when the similarity score is higher than a preset threshold, otherwise, choose the default seq2seq model. Their hybrid model with SNI achieves the best performance with 64.7% accuracy compared with the results of signal model. After the introduction of DNS system, Seq2SeqET [8] further extends their works of translating the input problem text into an expression tree by applying a seq2seq model. With the generated expression tree, the final numerical answer of a MWP could be inferred. To remove duplicated generation expression trees, equation normalisation was introduced to unify the representation of expression trees. Based on Seq2SeqET system, T-RNN [47] is proposed considered to be an improved version of Seq2SeqET. Except for the expression tree generation by applying seq2seq model, Wang et al. included quantity representation to help operator prediction. After the prediction for template expression, the answer module aims to learn quantity representation from BiLSTM model and predict encapsulated operators by using learnt quantity representation and recurrent NNs.

Saxton et al. [40] provided a comprehensive evaluation and analysis of two prevailing deep learning architectures, transformer and recurrent NN, to solve MWPs. According to their results, transformer performs better than recurrent architecture on both of their self-constructed evaluation dataset. Plausible assumptions to explain the better performance of transformer are its attributes of more carried calculations, shallower architecture, and internal sequential memory. They also observed with experimental results that the more operators involved in the calculation of arithmetic word problems, the less accurate the generated results would be. The accuracy reduces from 90% to 50% when there are multiplication and division involved other than addition and subtraction. They hypothesised that this is caused by no shortcuts or generated intermediate values to evaluate blend operator calculations. Patel et al. [35] also further investigate this issue. They also raise their doubts for the capability of DNNs to solve MWPs since they believe deep leaning architectures depend on shallow heuristics to receive fair performance on homogeneous datasets. Therefore, we are motivated to alleviate this issue.

# Chapter 3

# Methodology

As mentioned in Section 1.2, our task is unfolded into two main parts. In this section, we first need to investigate how to identify quantities and measurements in a given plain text. After we generated units for each quantity, the identified quantities and their corresponding generated units will be utilised in the structure of MWP solver. We will also illustrate the details of the solver architecture in the second section.

## 3.1 Measurement Extraction

Considering the limitation of the functionalities of the off-the-shelf quantity and measurement extraction tools and rareness of the openly accessible datasets related to unit and quantity extraction as we discussed in Section 2.1.1 and Section 2.1.2, we decided to construct a more suitable model of measurement extraction for our designed MWP solver. Based on the methodology and data of the up-to-date and openly accessible task MeasEval [18], we proposed to extract the defined entities in a given text and further generate a measurement for a corresponding extracted quantity.

### 3.1.1 Entity Extraction

Despite the designed concepts for entities are informative and the task is significant for creating Knowledge Graphs for extracted information of scientific literature, using all entity concepts from MeasEval task is unnecessary for our project since generating Knowledge Graphs is not one of our objectives. Thus, we decided to utilise the concepts of basic annotation set: *Quantity*, *MeasuredEntity*, *MeasuredProperty*, *Qualifier*, and *Unit*. Their definitions are illustrated in Table 3.1.

| Span Type | Definition |
|---|---|
| Quantity | A *Quantity* could be either a count only including a value, or a measurement composed with a value and an unit. It could obtain optional modifiers, such as *IsCount*. It can also appear individually without identifying other span type, such as *MeasuredEntity*. **Example**: The car's speed can be ***120 km per hour*** while being driven in the highway. |
| MeasuredEntity | Ideally, each *Quantity* should be associated with a *MeasuredEntity* which represents an entity being related to certain measurement. **Example**: The ***car***'s speed can be 120 km per hour while being driven in the highway. |
| MeasuredProperty | It is an optional span type that could be associated with *Quantity* and *MeasuredEntity*. It describes a property that a *Quantity* span could obtain. **Example**: The car's ***speed*** can be 120 km per hour while being driven in the highway. |
| Qualifier | It is also an optional span type which describes certain situations might impact the Quantity. **Example**: The car's speed can be 120 km per hour ***while being driven in the highway***. |
| Unit | An *Unit* span refers to the unit of measurement for the *Quantity*, which is usually a required span type associated with a *Quantity*. Generally, it is inside the span of an *Quantity*. **Example**: The car's speed can be 120 ***km per hour*** while being driven in the highway. |

Table 3.1: The definitions of the basic annotation set.

Moreover, we only use the quantity modifiers which are necessary for our project: *IsCount* and *IsList*. Additionally, we add a quantity modifier *IsRatio* to signify whether a Quantity denoting a ratio under the context. Their definitions are shown in Table 3.2.

| Modifier | Definition |
|---|---|
| IsCount | If a Quantity is a count type, it represents that both *Unit* and *MeasuredProperty* spans should not appear. **Example**: There are ***four*** apples on the table. {**Q**: four; **ME**: apples} |
| IsList | If there is a sequence of quantities, those quantities should be annotated as *IsList* only when they embellish the same *MeasuredEntity* with the same *MeasuredProperty* modifier and *Unit* span. **Example**: The concentrations of iron element in each control group are ***1.2, 2.2, and 0.5 mg***. {**Q**: 1.2, 2.2, and 0.5 mg; **ME**: control group; **MP**: concentrations of iron element} |
| IsRatio | It signifies whether a *Quantity* is ratio under the current context. **Example**: The strong wind obtains the speed of **20 m per s**. {**Q**: 20 m per s; **ME**: wind; **MP**: speed} **Example**: It has ***60 cm*** distance between each scan. {**Q**: 60 cm; **ME**: scan; **MP**: distance} |

Table 3.2: The definitions of the quantity modifiers.

The motivation of including *IsRatio* modifier is that MeasEval task is unable to infer the correct rate under certain circumstances as it is only an information extraction task. Inferring and generating a ratio from the sentence context is not one of its objectives. Although sometimes the rate is interchangeable to *Unit* when the ratio relation is explicitly expressed in the text and is annotated as *Unit* span, MeasEval task might not capture the actual unit of a quantity since it does not consider the context. For instance, for sentence "each node has 18 cores", its annotated tags in MeasEval are

"18 cores" for *Quantity* span with *Unit* "cores" and "node" for *MeasuredEntity* span. Nevertheless, under such context, the actual unit for the quantity should be "cores per node". With annotating the quantity modifier *IsRatio*, the *Unit* of a quantity will be considered as ratio, which is essential for generating the correct unit of measurement.

### 3.1.2 Unit Generation

In light of solving MWPs values more on the quantity and unit of measurement, only extracting unit from the input text is insufficient. That is the reason why the unit generation phase is included in this part. The annotated entities in the previous section are utilised to generate units, which involve *Quantity* along with quantity modifiers and *MeasuredEntity*.

The basic logic of unit generation is displayed in Algorithm 1. If the *Quantity* is a ratio and its unit of measurement does not contain keyword "per", this means its unit is required to be extracted under the sentence context. For *Quantity* who has identified unit, the string for its *Unit* and *MeasuredEntity* should be concatenated with keyword "per". For example, given a sentence "Each house is spaced 2 m apart.", we could extract *Quantity* "2 m" with *Unit* "m" and *MeasuredEntity* "house". The generated unit should be "2 m per house". As for *Quantity* who is a count type without unit of measurement, the unit should be generated by connecting the string of *MeasuredEntity* and noun or noun phrase after keywords "per" or "each". For instance, given a sentence "The number of supermarkets is 2 per block.", the extracted *Quantity* is "2" with modifier *IsCount* of True, and *MeasuredEntity* is "supermarkets". So the generated unit should be "2 supermarkets per block".

## 3.2 Designed Structure for Math Word Problem Solver

For our proposed methods, the architecture [47] of utilising recurrent NNs to predict equation templates and further predicting operators by learnt quantity representation is our main reference. There are two stages in the designed architecture presented by Wang et al. [47]: the structure prediction stage which generates tree-structured template equations by seq2seq model, and the answer module which predicts the unknown operators for the generated templates. Building on their approaches, we further introduce to apply quantity and unit extraction on their structure prediction stage, and perform unit constraints on their answer module to improve the accuracy of operator

**Algorithm 1** Unit Generation

```
Input: $Quantity$, Quantity Modifiers, and $MeasuredEntity$
Output: generated unit
  if $Quantity.IsRatio$ == True and $Quantity.Unit$ doesn't contains keyword "per" then
    if $Quantity.IsCount$ == False then
      generated unit = $Quantity.Unit$ + "per" + $MeasuredEntity$
    else
      generated unit = $MeasuredEntity$ +"per" + noun phrase after keywords "per" or "each"
    end if
  else
    generated unit = $Quantity.Unit$
  end if
  return generated unit
```

prediction.

### 3.2.1 Generation for Template Expressions

To capture the term dependency in both directions and prevent gradient vanishing, we also use Bi-LSTM [41] as the encoder to learn the representation of the input text and use LSTM [20] as the decoder to predict the next term and generate the whole sequence. Given the input MWPs text, $X = \{x_1, x_2, ..., x_n\}$, the hidden state of $i^{th}$ token could be obtained by concatenating two sequences of directional hidden states after passing the pretrained word embedding of the input sequence, $W = \{w_1, w_2, ..., w_n\}$, into the encoder:

$$\mathbf{h}_i^e = Concatenate(\overrightarrow{\mathbf{h}_i^e}, \overleftarrow{\mathbf{h}_i^e})$$

$$BiLSTM(\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_n) = \{\overrightarrow{\mathbf{h}_1^e}, ..., \overrightarrow{\mathbf{h}_n^e}\}, \{\overleftarrow{\mathbf{h}_1^e}, ..., \overleftarrow{\mathbf{h}_n^e}\}$$

By applying self-attention [46], the hidden state $h_j^d$ of $j^{th}$ time step in the decoder could be obtained:

$$\mathbf{a}_{ij} = \mathbf{h}_{j-1}^d \cdot \mathbf{h}_i^e$$

$$\alpha_{ij} = softmax(\mathbf{a}_{ij})$$

$$\mathbf{c}_j = \sum_{i=1}^{n} \alpha_{ij}\mathbf{h}_i^e$$

$$\mathbf{h}_j^d = f(\mathbf{h}_{j-1}^d, \mathbf{c}_j, y_{j-1})$$

where $a_{ij}$ is the attention weight of the hidden state in the previous time step of the decoder, which measures the similarity between decoder's previous hidden states $h_{j-1}^d$ and encoder's each hidden states $h_i^e$. Then, the attention weight $a_{ij}$ is normalised to obtain a distribution over elements of input $X$. The context vector $\mathbf{c}_j$ is the weighted average of the source hidden states. After getting the context vector, the vector of the destination hidden state $\mathbf{h}_j^d$ at time step $j$ could be obtained by combining decoder's previous hidden state $\mathbf{h}_{j-1}^d$, the previous output $y_{j-1}$, and the context vector $\mathbf{c}_j$ and passing into a non-linear function.

The probability distribution of the output $y_t$ at the current time step $t$ is computed by a softmax function over the vocabulary candidates. Subsequently, the probability of the generated sequence should be

$$p(y_1,...,y_m|x_1,...,x_n) = \prod_{t=1}^{m} p(y_t|\mathbf{h}^e, y_1,...,y_{t-1})$$

where $\mathbf{h}^e$ is the last hidden state of the encoder which represents the input problem text $X$ with the fixed dimension.

The generated sequence should be a **suffixed** template expression containing **abstracted** quantities and operators. All quantities are represented as $n_i$ where $i$ is the index of a number being detected in the problem text. All operators are expressed as $\langle op \rangle$ which encapsulates the details of an operator for the prediction task in the next phase. After the generation of template equation, we proposed to perform **unit generation** (discussed in Section 3.1) for all quantities included in the expression for further inference and performing unit constraints in the next phase. The visual demonstration is shown in Figure 3.1.

### 3.2.2 Operator Prediction

Wang et al. [47] referred this phase as Answer Generation Module. Again, they applied BiLSTM with self-attention to gain the quantity representation vectors. Then, they used a Recurrent NN to infer the unknown operator $\langle op \rangle$ in the generated template equations. Followed by their approach, given the input text $X = \{x_1, x_2, ..., x_n\}$ and corresponding word embedding $W = \{w_1, w_2, ..., w_n\}$, the hidden features $H = \{h_1, h_2, ..., h_n\}$ are obtained from the BiLSTM encoder, which is the same procedure

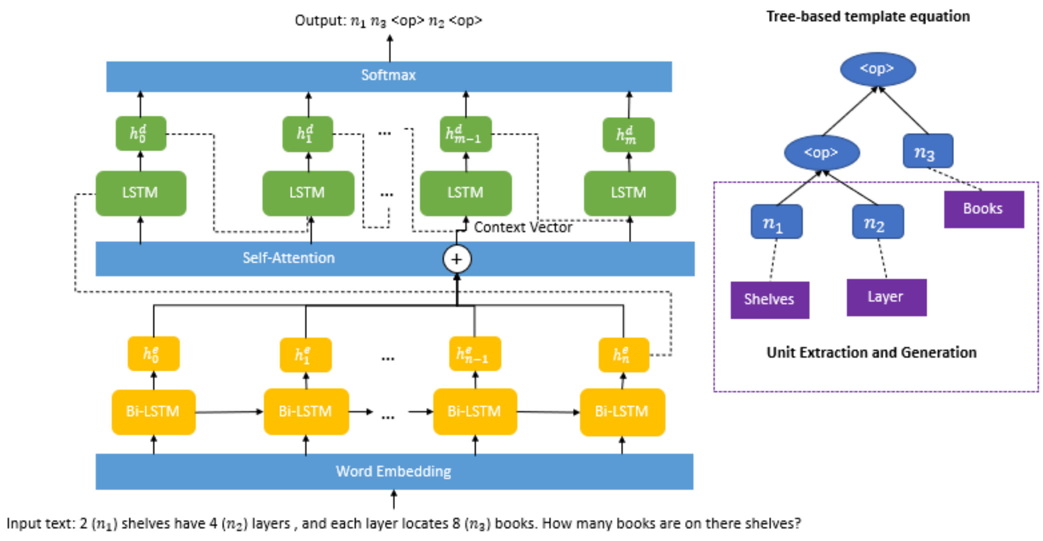
Output: $n_1$ $n_3$ <op> $n_2$ <op>
Softmax
$h_0^d$
$h_1^d$
$h_{m-1}^d$
$h_m^d$
LSTM
LSTM
LSTM
LSTM
Context Vector
Self-Attention
+
$h_0^e$
$h_1^e$
$h_{n-1}^e$
$h_n^e$
Bi-LSTM
Bi-LSTM
Bi-LSTM
Bi-LSTM
Word Embedding
Input text: 2 ($n_1$) shelves have 4 ($n_2$) layers , and each layer locates 8 ($n_3$) books. How many books are on there shelves?
Tree-based template equation
<op>
<op>
$n_3$
$n_1$
$n_2$
Books
Shelves
Layer
Unit Extraction and Generation

Figure 3.1: The architecture of generating template equation. Given the text inputs, each token is encoded by BiLSTM with self-attention, and decoded to generate a template with suffix format, which could be converted into a tree. After the template generation, corresponding units will be extracted and further generated for each quantity inside the template.

introduced in the previous section. Indicating by their sequence positions, the quantities generated in the template equation could be located in the hidden feature list, since each $h_i$ represents the $i^{th}$ token in the input sequence and $n_j$ denotes the $j^{th}$ quantity appearing in the input text. Therefore, it is straightforward to get hidden vectors for the quantities $H^q = \{h^q_1, h^q_2, ..., h^q_k\}$ from $H$ according to the index information provided by $n_j$.

The quantity representation is learnt by conducting self-attention again:

$$Q = W_q \oplus Q'$$
$$Q' = tanh([C, H^q]W_1 + b_1)$$
$$C = AH$$
$$A = softmax(H^q H^T)$$

where $W_q$ is the quantity embedding extracted from the text embedding $W$. $Q'$ is a temporary quantity representation which performs non-linear transformation on the concatenated context vector $C$ and quantity hidden vector $H^q$.

Thereafter, we define **unit category**, which applies *WordNet* [1] to find the lexical category for the units of two operands. This is motivated by the compatibility of addition and subtraction operation. Despite those two operators are normally performed strictly with the same units, we should allow the operation to be carried on when the units of operands share the same category. For instance, given the question “Given 3 apples and 2 oranges, how many fruits in total?”, quantity “3” with unit “apple” and quantity “2” with unit “orange” should allow addition and subtraction since their units belong to the same “fruit” category. This process is defined as **soft constraints** because it offers “suggestions”, fetching unit category, to permit the manipulation while it fails to satisfy the hard-code. Furthermore, we could predefine some rules to constrain the operator prediction. One basic rule is:

*Only two operands with the same unit or their units belonging to the same category can apply addition or subtraction, otherwise, apply multiplication or division.*

Inspired by the idea of redesigning the activation function of seq2seq model regarding incorrect generation while using softmax function [50], we also determine to redesign the activation function for this architecture. The original softmax function is

[1] https://wordnet.princeton.edu/

redesigned by adding a binary vector $\rho_t$ whose each element refers to an operator prediction in the output vocabulary where “1” and “0” indicate a possible or impossible prediction correspondingly. To determine whether it is possible to predict an operator, we need to utilise our predefined rules and generated units for quantities in the template expression. The prediction probability distribution of an operator at time step $t$ is expressed as:

$$P(o \,|\, q_p) = \frac{\rho_t \odot e^{W_3 q_c}}{\sum \rho_t \odot e^{W_3 q_c}}$$

$$q_p = tanh(W_2([q_l, q_r]) + b_2)$$

where $q_p$ is the representation of the parent node. It could be calculated by passing the concatenated vector of the left and right quantity representation to a non-linear projection function. $W_2$ and $W_3$ are weights to be learnt in the RNN model.

For instance, given a generated suffixed expression $n_1 n_3 \langle op \rangle n_2 \langle op \rangle$, after translating it into an expression with infix form $(n_1 \langle op \rangle n_3) \langle op \rangle n_2$ by using a stack and acquiring the learnt quantity representation of $n_1$, $n_2$ and $n_3$ from the encoder, quantity representation for $n_1$ and $n_3$ should be the quantity representation of the left and right node $q_l$ and $q_r$ accordingly in the sub-tree to infer the quantity representation $q_p$ and new quantity $n_p^1$ for their least common ancestor (i.e. an internal parent node for the sub-tree). At this stage, we proposed to perform **unit inference** for the predicted new quantity $n_p^1$ for further operator prediction and final result generation.

Unit inference phase is to derive the unit of a parent node for a left and right node, which is also designed as a rule-based module. We define several basic rules for inference:

Rule 1: If the predicted operator is addition or subtraction, and if two operands have the same unit, the inferred unit should remain the same;
Rule 2: If the predicted operator is addition or subtraction, and if two operands don’t have the same unit but belong to the same unit category, the inferred unit should be their unit category;
Rule 3: If the predicted operator is multiplication, the inferred unit should offset the rate unit, such as *“m/s* $\times$ *s”* should get the result with unit *“m”*;
Rule 4: If the predicted operator is division, the inferred unit should be newly created. For example, a quantity with unit *“km”* and a quantity with unit “h” share the predicted operator $\div$ inferring the result with unit *“km/h”*.

Subsequently, with the predicted operator representation and new quantity for the internal node, the representation and updated quantity of the successive operator in the template expression could be predicted as well as inferred unit. Iteratively, we could acquire the predicted operator in the root node and deliver a final result for the whole expression tree. The visual demonstration of this module is illustrated in Figure 3.2.

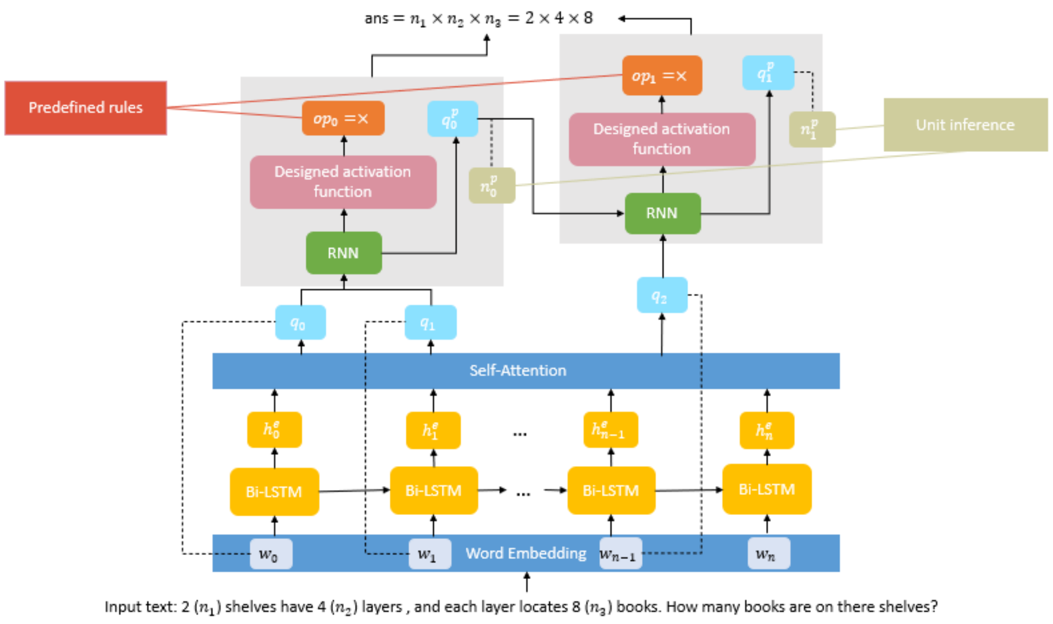
ans = $n_1 \times n_2 \times n_3 = 2 \times 4 \times 8$
Predefined rules
$op_0 = \times$
$q_0^p$
Designed activation function
$n_0^p$
RNN
$op_1 = \times$
$q_1^p$
$n_1^p$
Unit inference
$q_0$
$q_1$
$q_2$
Self-Attention
$h_0^e$
$h_1^e$
$h_{n-1}^e$
$h_n^e$
Bi-LSTM
$w_0$
$w_1$
Word Embedding
$w_{n-1}$
$w_n$
Input text: 2 ($n_1$) shelves have 4 ($n_2$) layers , and each layer locates 8 ($n_3$) books. How many books are on there shelves?

Figure 3.2: The structure of operator prediction. Encoded by BiLSTM, quantity representation could be learnt. Adopt RNN as the decoder to predict operators by using redesigned activation function and predefined rules. The quantity representation vector of the predicted operator also could be learnt from the decoder. The numeric along with its unit could be further inferred by applying the rule-based unit inference system.

As for the learning objective of this module, the parameters in the encoder (BiLSTM) with self-attention and the decoder (recursive NN) are trained altogether via minimising the objective function:

$$J(\theta) = -\frac{1}{r}\sum_{i=1}^{r} logP(o_p(i)\,|\,q_l(i), q_r(i))$$

where $r$ is the number of least common ancestors (inner nodes) in the whole expression tree.

# Chapter 4

# Implementation

As mentioned in Chapter 1, we are motivated to construct a model obtaining the functionalities of identifying quantities and producing their unit of measurements which is suitable for MWP solvers. Building on the methodology in the previous chapter, we decided to firstly focus on implementing quantity and unit extraction. In the following sections, we will explain the details of model implementation and provide the experiment results along with our discussion.

## 4.1 Identifying Question Part

As further explained in Section 3.2, the quantities and unit of measurements should be extracted in both premise and question part of a MWP for the sake of training the generation process of expression template tree by unit constrains and inference. Although for arithmetic word problems of elementary level who assemble datasets such as MAWPS [26] and Math23K [50], the question part is usually an interrogative sentence which located at the last sentence of a MWP, the identification process could become more complicated when the problem difficulty upgrades. In dataset repository MATH [19] which contains middle school or pre-university level MWPs, the question part could be located at other place. For instance, the question part is the first sentence for the MWP "In the diagram, what is the value of $y$? [asy] draw((5,0)–(0,0)–(0,5)); draw((.5,0)–(.5,.5)–(0,.5)); draw(Arc((0,0),1,90,360),Arrows); label("$y^\circ$", (-1,-1), SW); [/asy]". Sometimes, the question is not interrogative but imperative. For example, the question part of a MWP in MATH repository "From $1,2,3,4,5,6,7,8,9$ we randomly draw with replacement numbers $n$ and $m$. Compute the probability that $n \times m = 24$." is an imperative sentence. Thus, hard-coding the question identification process is unwise for this project. The first stage of implementation would be identifying a question part of a MWP by using machine learning and binary classification.

The database we used is combined with both the interrogative and declarative sentence classification dataset available in Kaggle [1], and imperative sentences extracted from AQuA [26] which is a large-scale dataset collecting 34,202 multi-choice math problems. By chunking the tokenised sentence, the sentence is annotated as imperative if it start with a verb phrase. Adding those imperative sentences is to replenish the Kaggle dataset lacking imperative-style questions existing in MWP solving.

The features attempted are TF-IDF and word embedding [2]. After extracting features, we applied decision tree and gradient boosting to train the binary classifier. As shown in Table 4.1, two learning methods obtain similar prediction results, and applying word embedding does not improve results prominently. Considering time and resource consumption, using TF-IDF feature with either one algorithms to perform sentence classification is sufficient.

| | **TF-IDF** | **Embedding** |
|---|---|---|
| **Decision Tree** | 0.93 | 0.94 |
| **Gradient Boosting** | 0.93 | 0.94 |

Table 4.1: The accuracy results for sentence classification with TF-IDF and word embedding features by applying decision tree and gradient boosting algorithm.

## 4.2 Quantity and Unit Extraction

According to the experimental results provided by Schweter and Akbik [42], to achieve SoTA results on NER task, fine-tuning transformer gains slightly advantage compared with utilising extracted features by using transformer, and applying document-level features could significantly improve prediction quality. Our implementation for annotating span types introduced in Section 3.1.1 utilise *flair* library with fine-tuneable transformer embedding to train a sequence tagger.

Firstly, we construct our dataset by reusing openly accessible data from MeasEval task [18]. After retrieving all text files in MeasEval task, we used sentence tokeniser to

[1]The dataset is consisted with parsed SQuAD dataset and SPAADIA dataset. Data Link: https://www.kaggle.com/shahrukhkhan/questions-vs-statementsclassificationdataset

[2]We applied Sentence Transformer with pretrained model “all-MiniLM-L6-v2”. Link: https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

split each sentence and include the span for each sentence within the document. Then read in the annotated files with tsv format for each plain text file, and annotate each token as one of the tags *Quantity*, *MeasuredEntity*, *MeasuredPropoerty*, and *Qualifier* sentence by sentence. We followed the corpus format in *flair* tutorial [3] for NER task by using column format with BIO schema and separating sentences by lines. Then, we split the data for training, evaluating, and testing with ratio 7:1:2.

After data preparation, we performed fine-tuning a transformer for this annotation task. Technically, it is beneficial for assembling everything on a single structure and fine-tuning it entirely [42]. We illustrate some noticeable parameter setting in Table 4.2.

| **Parameter** | **Value** |
|---|---|
| Model | dslim/bert-base-NER |
| Layers | -1 |
| Subtoken pooling | first |
| Use context | True |
| Use CRF | True |
| Learning rate | 5.0e-6 |
| Mini batch size | 4 |
| Max epochs | 20 |
| Scheduler | OneCycleLR |

Table 4.2: The noticeable parameter setting for fine-tuning the architecture.

Our motivation of using 'dslim/bert-base-NER' model [4] is that it is a BERT architecture, the 'bert-base-cased' model [5], fine-tuned by NER task on CoNLL-2003 dataset with English version, which achieves SoTA results on NER task. Based on the experiment results provided by Schweter and Akbik [42], adding CRF decoder between the transformer and the linear classifier could slightly improve the results for English. We still decided to add the CRF decoder for small improvement. We used the context for the transformer to obtain the document-level features by including 64 left and right tokens at each side suggested by Schweter and Akbik. We also chose the "first" subtoken pooling strategy [12] to take the representation of the first subtoken to represent the whole token. Moreover, we adopt one-cycle training strategy [44] which is able to

[3]https://github.com/flairNLP/flair/blob/master/resources/docs/TUTORIAL_6_CORPUS.md
[4]https://huggingface.co/dslim/bert-base-NER
[5]https://huggingface.co/bert-base-cased

linearly decrease the learning rate until reaching 0.

## 4.3 Analysis and Discussion

Based on the parameter settings shown in the previous section, we obtain the experiment results for four main span types: *Quantity*, *MeasuredEntity*, *MeasuredProperty*, and *Qualifier*, which is illustrated in Table 4.3. We also compare our F1 scores of four span types with the baseline and the best performer of MeasEval task shown in Table 4.4.

| **Span Type** | **Precision** | **Recall** | **F1-score** | **Support** |
|---|---|---|---|---|
| Quantity | 0.701 | 0.763 | 0.730 | 316 |
| MeasuredEntity | 0.271 | 0.412 | 0.327 | 262 |
| MeasuredProperty | 0.272 | 0.317 | 0.293 | 186 |
| Qualifier | 0.063 | 0.034 | 0.044 | 88 |

Table 4.3: The experiment results of each span type in our fine-tuned architecture.

| System Name | **Quantity** | **MeasuredEntity** | **MeasuredProperty** | **Qualifier** |
|---|---|---|---|---|
| Our System | 0.730 | 0.327 | 0.293 | 0.044 |
| Baseline | 0.827 | 0.053 | 0.064 | 0.005 |
| LIORI | 0.861 | 0.437 | 0.467 | 0.163 |

Table 4.4: The F1 score results compared with the baseline of MeasEval and the overall best performer LIORI [11]. Additionally, the best performance on *MeasuredProperty* span type reaches **0.804** F1 score [16].

We could conclude from the results and comparison that *Quantity* span might be the easiest one to predict while Qualifier is the most challenging tag among four types. Although we get comparable results for *MeasuredEntity*, *MeasuredProperty*, and *Qualifier*, our system fails to outperform the baseline for predicting *Quantity* type. To analyse the reason for the prediction quality not achieving fair performance, one possible assumption is the parameter settings indicating that we still need to refine and adjust the parameters. We also hypothesise it is due to the integrated annotation tag prediction. According to the overview of nine systems participated in MeasEval task [18], most of them applied multi-task sequence tagging for *MeasuredEntity*, *MeasuredProperty*, and *Qualifier* at the final stage by forwarding both the original sentences and annotated Quantity spans as inputs. Proved by the top performance systems, the approach of performing Quantity identification at first followed by multi-task tagging or staged individual subtask prediction is effective for this task. Nonetheless, our system stacked four span types together to perform a single-task prediction, which might explain the unsatisfying results somehow.

As for the prediction of Unit tag, the results are surprisingly good. We also adopt the same fine-tuning module with the same parameters for this task, yet achieves the highest performance compared with other system. We treat it as a staged individual NER tagging task and forward the annotated span of units extracted by ourselves along with the original text in the dataset into the NER-orientated pretrained BERT module. One possible explanation is staged single-task learning might be suitable for this subtask. The results for this task are shown in Table 4.5.

| | **Precision** | **Recall** | **F1-score** | **Support** |
|---|---|---|---|---|
| Our System | 0.901 | 0.898 | **0.900** | 315 |
| Baseline | / | / | 0.561 | / |
| Stanford MLab | / | / | **0.760** | / |
| LIORI | / | / | 0.722 | / |

Table 4.5: The experiment results of *Unit* span type compared with the baseline in MeasEval task, the overall best performer LIORI, and the top scorer Stanford MLab [30] in *Unit* prediction task.

# Chapter 5

# Conclusions

## 5.1 Future Works

Building on our proposed MWP architecture and unit generation approach, we provide some guide for future research and experiments in the following:

1. Supplement dataset: In light of the limited access for the data related to quantity and unit tagging, either we need human-annotators to tag "IsRatio" span type, or we need to do more research to find other datasets appropriate for our task.

2. Improve the module for span type prediction: Followed by the major approach of tag prediction concluded in [18], predict *Quantity* span type at first. Subsequently, adopt multi-task learning for sequence tagging of *MeasuredEntity*, *MeasuredProperty*, and *Qualifier* span types. Adjust parameters to find the settings achieving the best performance.

3. Implement the proposed MWP solvers: For the proposed design of MWP solving system, implement the system, evaluate the system on MAWPS [27], SVAMP [35] and GSM8K [9] dataset, and compare the experiment results with our main reference [47] and SoTA systems, such as trained verifiers based on GPT-3 families to score solutions [9], Graph2Tree with RoBERTa [35].

## 5.2 Summary

Benefited by taking natural language and unstructured text as inputs without programmed and structured format, the systems of automatically solving MWPs attract ever-increasing research attentions. This research field is challenged by the difficulty of transforming natural language into high-level representation to mapping into or generating an equivalent equation which will deliver a final result for the given input problem. There are three mainstream approaches regarding solving MWPs: symbolic, statistical and deep learning methods. Recently, abundant studies take the advantages of deep learning architectures and their strong ability of generalisation to shift the research direction to mainly focus on deep learning approach. Despite utilising deep neural networks could achieve fair results without tedious manual feature engineering or complicated high-level structure designs, a recent study [35] challenges deep learning architectures' capability of addressing MWPs by providing evidence of their high reliance on shallow heuristics. Petal et al. demonstrate with experimental results that SoTA deep architectures could obtain high performance without the revealing of the question part in a given MWP or word-order information.

Motivated by the issues of deep neural networks, we proposed a seq2seq tree-based template expression generations with unit inference and unit constraints. We believe that constrained by unit information of the question and premise part for a given MWP, the architecture could better understand a MWP and make a better decision regarding the operator prediction. Before the implementation of the MWP solving system, we are required to construct a module for quantity and unit extraction for both the question and premise part of MWPs. We found that despite there are a few tools related to quantity and unit extraction, they are unsuitable for our project due to the failure of detecting implicit ratio under a sentence context and the limitation of unit registry predicting plenty of units as "dimensionless". Such that, our main implementation concentrates on extracting units and quantities in a given MWP text. After fine-tuning a NER-orientated BERT module, we received a comparable results for span type prediction. We also discussed our explanation to the experimental results along with some suggestions of module improvement. At last, we listed some future works that could be carried on.

We believe our works could contributes to the research society of solving MWP. The approach of learning with unit constraints could provide some inspiration and direction to the research society and the system of quantity and unit extraction could facilitate researchers who are investigating the topic of information extraction.

# Bibliography

[1] Bilal Abu-Salih. Domain-specific knowledge graphs: A survey. *Journal of Network and Computer Applications*, 185:103076, 2021.

[2] Alan Akbik, Tanja Bergmann, and Roland Vollgraf. Pooled contextualized embeddings for named entity recognition. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 724–728, 2019.

[3] Iz Beltagy, Kyle Lo, and Arman Cohan. Scibert: A pretrained language model for scientific text. *arXiv preprint arXiv:1903.10676*, 2019.

[4] Soumia Lilia Berrahou, Patrice Buche, Juliette Dibie-Barthelemy, and Mathieu Roche. How to extract unit of measure in scientific documents? In *KDIR: Knowledge Discovery and Information Retrieval*, pages 454–459. Springer, 2013.

[5] Daniel G Bobrow. A question-answering system for high school algebra word problems. In *Proceedings of the October 27-29, 1964, fall joint computer conference, part I*, pages 591–614, 1964.

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

[7] Eugene Charniak. Computer solution of calculus word problems. In *Proceedings of the 1st international joint conference on Artificial intelligence*, pages 303–316, 1969.

[8] Ting-Rui Chiang and Yun-Nung Chen. Semantically-aligned equation generation for solving and reasoning math word problems. *arXiv preprint arXiv:1811.00720*, 2018.

[9] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[10] Hamish Cunningham. Gate: A framework and graphical development environment for robust nlp tools and applications. In *Proc. 40th annual meeting of the association for computational linguistics (ACL 2002)*, pages 168–175, 2002.

[11] Adis Davletov, Denis Gordeev, Nikolay Arefyev, and Emil Davletov. Liori at semeval-2021 task 8: Ask transformer for measurements. In *Proceedings of the 15th International Workshop on Semantic Evaluation (SemEval-2021)*, pages 1249–1254, 2021.

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

[13] Cicero Dos Santos and Bianca Zadrozny. Learning character-level representations for part-of-speech tagging. In *International Conference on Machine Learning*, pages 1818–1826. PMLR, 2014.

[14] Luca Foppiano, Thaer M Dieb, Akira Suzuki, and Masashi Ishii. Proposal for automatic extraction framework of superconductors related information from scientific literature. 2019.

[15] Luca Foppiano, Laurent Romary, Masashi Ishii, and Mikiko Tanifuji. Automatic identification and normalisation of physical measurements in scientific literature. In *Proceedings of the ACM Symposium on Document Engineering 2019*, pages 1–4, 2019.

[16] Akash Gangwar, Sabhay Jain, Shubham Sourav, and Ashutosh Modi. Counts@IITK at SemEval-2021 task 8: SciBERT based entity and semantic relation extraction for scientific data. In *Proceedings of the 15th International Workshop on Semantic Evaluation (SemEval-2021)*, pages 1232–1238, Online, August 2021. Association for Computational Linguistics.

[17] Tianyong Hao, Hongfang Liu, and Chunhua Weng. Valx: a system for extracting and structuring numeric lab test comparison statements from text. *Methods of information in medicine*, 55(03):266–275, 2016.

[18] Corey Harper, Jessica Cox, Curt Kohler, Antony Scerri, Ron Daniel Jr., and Paul Groth. SemEval-2021 task 8: MeasEval – extracting counts and measurements and their related contexts. In *Proceedings of the 15th International Workshop on Semantic Evaluation (SemEval-2021)*, pages 306–316, Online, August 2021. Association for Computational Linguistics.

[19] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

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

[21] Mark Hopkins, Cristian Petrescu-Prahova, Roie Levin, Ronan Le Bras, Alvaro Herrasti, and Vidur Joshi. Beyond sentential semantic parsing: Tackling the math sat with a cascade of tree transducers. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 795–804, 2017.

[22] Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. Learning to solve arithmetic word problems with verb categorization. In *EMNLP*, pages 523–533. Citeseer, 2014.

[23] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.

[24] Kyle Hundman and Chris A Mattmann. Measurement context extraction from text: Discovering opportunities and gaps in earth science. *arXiv preprint arXiv:1710.04312*, 2017.

[25] Rik Koncel-Kedziorski, Hannaneh Hajishirzi, Ashish Sabharwal, Oren Etzioni, and Siena Dumas Ang. Parsing algebraic word problems into equations. *Transactions of the Association for Computational Linguistics*, 3:585–597, 2015.

[26] Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. MAWPS: A math word problem repository. In *Proceedings of the*

*2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1152–1157, San Diego, California, June 2016. Association for Computational Linguistics.

[27] Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. Mawps: A math word problem repository. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1152–1157, 2016.

[28] Nate Kushman, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay. Learning to automatically solve algebra word problems. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 271–281, 2014.

[29] Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. Biobert: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240, 2020.

[30] Patrick Liu, Niveditha Iyer, Erik Rozi, and Ethan A. Chi. Stanford mlab at semeval-2021 task 8: 48 hours is all you need. In *SEMEVAL*, 2021.

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

[32] Nikolay Maksimov, Anastasia Gavrilkina, Victoriy Kuzmina, and Ekaterina Borodina. Ontology of properties and its methods of use: Properties and unit extraction from texts. *Procedia Computer Science*, 169:70–75, 2020.

[33] Takuya Matsuzaki, Takumi Ito, Hidenao Iwane, Hirokazu Anai, and Noriko H Arai. Semantic parsing of pre-university math problems. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2131–2141, 2017.

[34] Anirban Mukherjee and Utpal Garain. A review of methods for automatic understanding of natural language mathematical problems. *Artificial Intelligence Review*, 29(2):93–122, 2008.

[35] Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are nlp models really able to solve simple math word problems? *arXiv preprint arXiv:2103.07191*, 2021.

[36] Shuai Peng, Ke Yuan, Liangcai Gao, and Zhi Tang. Mathbert: A pre-trained model for mathematical formula understanding. *arXiv preprint arXiv:2105.00377*, 2021.

[37] Hajo Rijgersberg, Mark Van Assem, and Jan Top. Ontology of units of measure and related concepts. *Semantic Web*, 4(1):3–13, 2013.

[38] Subhro Roy and Dan Roth. Solving general arithmetic word problems. *arXiv preprint arXiv:1608.01413*, 2016.

[39] Erik F Sang and Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. *arXiv preprint cs/0306050*, 2003.

[40] David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models. *arXiv preprint arXiv:1904.01557*, 2019.

[41] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.

[42] Stefan Schweter and Alan Akbik. Flert: Document-level features for named entity recognition. *arXiv preprint arXiv:2011.06993*, 2020.

[43] Shuming Shi, Yuehui Wang, Chin-Yew Lin, Xiaojiang Liu, and Yong 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.

[44] Leslie N Smith. A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay. *arXiv preprint arXiv:1803.09820*, 2018.

[45] Markus D Steinberg, Sirko Schindler, and Jan Martin Keil. Use cases and suitability metrics for unit ontologies. In *OWL: Experiences and Directions–Reasoner Evaluation*, pages 40–54. Springer, 2016.

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

[47] Lei Wang, Dongxiang Zhang, Jipeng Zhang, Xing Xu, Lianli Gao, Bing Tian Dai, and Heng Tao Shen. Template-based math word problem solvers with recursive neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7144–7151, 2019.

[48] Xinyu Wang, Yong Jiang, Nguyen Bach, Tao Wang, Zhongqiang Huang, Fei Huang, and Kewei Tu. Automated concatenation of embeddings for structured prediction. *arXiv preprint arXiv:2010.05006*, 2020.

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

[50] Yan Wang, Xiaojiang Liu, and Shuming Shi. Deep neural solver for math word problems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 845–854, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

[51] Ralph Weischedel, Martha Palmer, Mitchell Marcus, Eduard Hovy, Sameer Pradhan, Lance Ramshaw, Nianwen Xue, Ann Taylor, Jeff Kaufman, Michelle Franchini, et al. Ontonotes release 5.0 ldc2013t19. *Linguistic Data Consortium, Philadelphia, PA*, 23, 2013.

[52] Daya C Wimalasuriya and Dejing Dou. Ontology-based information extraction: An introduction and a survey of current approaches, 2010.

[53] Ikuya Yamada, Akari Asai, Hiroyuki Shindo, Hideaki Takeda, and Yuji Matsumoto. Luke: deep contextualized entity representations with entity-aware self-attention. *arXiv preprint arXiv:2010.01057*, 2020.