# *EquGener*: A Reasoning Network for Word Problem Solving by Generating Arithmetic Equations

**Pruthwik Mishra**
MT & NLP Lab, KCIS
LTRC, IIIT-Hyderabad

**Litton J Kurisinkel**
IREL Lab, KCIS
LTRC, IIIT-Hyderabad

**Dipti Misra Sharma**
MT & NLP Lab, KCIS
LTRC, IIIT-Hyderabad

**Vasudeva Varma**
IREL Lab, KCIS
LTRC, IIIT-Hyderabad

## Abstract

Word problem solving has always been a challenging task as it involves reasoning across sentences, identification of operations and their order of application on relevant operands. Most of the earlier systems attempted to solve word problems with tailored features for handling each category of problems. In this paper, we present a new approach to solve simple arithmetic problems. Through this work we introduce a novel method where we first learn a dense representation of the problem description conditioned on the question in hand. We leverage this representation to generate the operands and operators in the appropriate order. Our approach improves upon the state-of-the-art system by 3% in one benchmark dataset while ensuring comparable accuracies in other datasets.

## 1 Introduction

According to (Koncel-Kedziorski et al., 2015; Hosseini et al., 2014), a word problem narrates a partial world state consisting of entities, entity holders, quantities and other participants. It either changes the state of entities or expounds the relationships between them. At the end, the problem queries about a quantity which can be obtained by arithmetically combining several quantities mentioned in the question as shown in Table 1. Natural language understanding plays a key role in solving any Mathematical Word Problem (MWP). Automating such problem solving requires better reasoning across sentences, also involves complex inference using world knowledge, co-reference resolution and a large vocabulary. Simple keyword or pattern matching is less equipped to take up such a challenge (Bakman, 2007).

| There are 112 short trees and 119 tall trees currently in the park . Park workers will plant 105 short trees today . How many short trees will the park have when the workers are finished ? |
|---|
| Answer : 112 + 105 |

Table 1: Mathematical Word Problem Example

Previous systems either rely heavily on specific set of problem abstractions based on verb categories (Hosseini et al., 2014) or learning equations from pre-defined set of templates (Kushman et al., 2014). Deep neural solvers (Wang et al., 2017) proposed a combination of sequence-to-sequence model and information retrieval system. However, an ideal equation generation system for a word problem should be able to identify components of the equation and form the equation in an orderly fashion independent of syntax or vocabulary of the sentences.

In this work, we introduce *EquGener* - an equation generator using a memory network with an equation decoder. Intuitively, a human math solver collects relevant details from the problem description which equip her to solve the problem (Dellarosa, 1986). Driven by this intuition, we learn a dense semantic representation of the problem description consisting of multiple sentences, conditioned on the question in hand. An equation necessary to solve the question


32nd Pacific Asia Conference on Language, Information and Computation
Hong Kong, 1-3 December 2018

is decoded out of the learned dense representation. We propose a novel architecture for solving word problems using an end-to-end memory network with an equation decoder. Unlike most of the previous works which are equipped to handle only a subset of operations, our system handles problems involving all kinds of arithmetic operations.
Our code is shared at `http://somewhereontheweb`

## 2 Approach

### 2.1 Base model

The encoder-decoder architecture (Sutskever et al., 2014) has proved beneficial in many applications. But, this architecture has its limitation in handling long input sequences. This limitation results from a fixed size internal representation of the encoded sequence where the target sequence is decoded from this representation. Attention mechanism (Luong et al., 2015) has been widely used where the network learns the relative importance on which parts to attend to. In this architecture, the input sequence is encoded as a sequence of vectors and the decoder has access to all these vectors instead of a single vector. We modeled the input sequence as a sequence of word vectors. Each word vector is a concatenated vector representation of pre-trained glove (Pennington et al., 2014) embeddings and the embeddings learned by the network from the training corpus. The equation generation for a word problem requires the identification of words which indicate the presence of operands and operators. So an attention based encoder-decoder has been used as a baseline for our equation generation system. Both the encoder and decoder employ Long-term short-term memory (LSTM) to represent the input and target sequence respectively.

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

The $j^{th}$ hidden state of the encoder is computed as equation 1 using an LSTM. The decoder is initialized with the hidden and cell states obtained at the last time-step of the encoder. Global attention model (Luong et al., 2015) considers all the hidden states of the encoder to derive the context vector. Each hidden state of the input word sequence and the hidden state of the equation are compared to arrive at the alignment.

$$\begin{aligned} \mathrm{a_t} &= \mathrm{align}(\mathrm{h_t}, \tilde{\mathrm{h_s}}) \\ &= \frac{\exp({\mathrm{h_t}}^{\mathrm{T}}.\tilde{\mathrm{h_s}})}{\Sigma_{\mathrm{s'}} \exp({\mathrm{h_t}}^{\mathrm{T}}.\tilde{\mathrm{h_s'}})} \end{aligned} \tag{2}$$

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

$$\mathrm{c_t} = \Sigma_t \mathrm{a_t} \times \mathrm{h_t} \tag{3}$$

The attentional hidden state in the decoder side is obtained by concatenating the context vector from the input word sequence and equation hidden state.

$$\tilde{\mathrm{h_t}} = \mathbf{W_c}[\mathrm{c_t}; \mathrm{h_t}] \tag{4}$$

### 2.2 Memory Network Based Encoder

End-to-End memory networks (Sukhbaatar et al., 2015) succeeds in representing sentences as well as captures the salience or the intent of the question in Question Answering systems. We used a variant of memory network to solve arithmetic word problems. *EquGener* learns dense continuous representations of the question sentence and the supporting sentences featuring in the problem text. The word representations in the supporting sentences act as memories and these are weighted as per the question. The relevant memories are assigned higher weights than the irrelevant ones. This weighted combination of memory vectors is then learned by the encoder to obtain a hidden representation of the word sequence appearing in the supporting sentences conditioned on the question words. The decoder then generates the equation conditioned on the encoded hidden representation. Two kinds of memory network settings are generally followed : (1) explicit identification of supporting sentences where the equation components lies (2) without any information regarding supporting sentences. We used the later configuration for our system which required less supervision than the former. Considering our input to be a sequence of words $w_1, w_2, w_3, .., w_n$, this words are transformed or embedded into a lower dimension space d which can be achieved via an embedding matrix A of size $d \times V$ where V is the vocabulary size. Each word is represented as a vector concatenating its glove embedding (Pennington et al.,

2014) and its transformed inputs. These are called the memories in our memory network. *EquGener* learned the words which had to be attended to based only on the question text. The input to our system is a sequence of words. As the word embeddings learned from a small training corpus were not reliable, we augmented the word representations with pre-trained glove embeddings. This strategy helped us in handling words which were not available in pre-trained word vectors like $num_i$ in the problem text.

$$\mathrm{m_i} = [\mathrm{w_p}; \mathrm{w_e}] \tag{5}$$

where $w_p$ and $w_e$ denotes the pre-trained embeddings and learned word embeddings respectively. The question sentence in a word problem is also embedded using a matrix B of similar dimension to A. A is used to embed the words appearing in supporting sentences while embedding B is used for the words in the question. This represents the internal state **u** for the question. The match between internal state and memory elements helps in predicting the components involved in an equation. The match is found using a dot product between u and $m_i$ and applying a softmax function over it.

$$\begin{aligned} \mathrm{p_i} &= \mathrm{softmax}(\mathrm{u^T}.\mathrm{m_i}) \\ \mathrm{softmax}(\mathrm{x_i}) &= \frac{\mathrm{e^{x_i}}}{\sum_{\mathrm{j=1}}^{\mathrm{n}} \mathrm{e^{x_j}}} \end{aligned} \tag{6}$$

where n refers to the total number of words in the supporting sentences
There is a probability score for each word appearing in memories. Each memory $m_i$ has a corresponding output vector $c_i$ which is obtained through another embedding matrix C. The output vector is a weighted sum of $p_i$ and $c_i$s. We used a slight modification to this formulation used in memory network (Sukhbaatar et al., 2015) which is given below in equation 7. We describe two formulations for single layer and multi-layer memory network.

#### 2.2.1 Single Layer

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

$$\mathrm{o} = \sum_{\mathrm{i=1}}^{\mathrm{n}} \mathrm{p_i} + \mathrm{c_i} \tag{7}$$

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

$$\mathrm{d} = \mathrm{Dense(o)} \tag{8}$$

$$\mathrm{E} = \mathrm{u} + \mathrm{d} \tag{9}$$

#### 2.2.2 Multi-Layer

In case of multi-layer memory network, the hidden state gets updated in each hop with discovery of new attention points in the memories according to the question. The attention produces a probability distribution of the semantic match between the words present in question and supporting sentences. Same input and output embeddings are used across the layers. The match For a k-hop memory network where the memory layers are stacked on top of each other, the internal state is updated as follows,

$$\mathrm{u_{k+1}} = \mathrm{u_k} + \mathrm{d_k} \tag{10}$$

and at the final hop K,

$$u_{K+1} = u_K + d_K \tag{11}$$

The computation of $d_k$ is same as the single layer case. The hidden state of the encoder is computed as per the equations 11-16. [1]

$$\mathrm{f_t} = \sigma(\mathrm{W_f} \cdot [\mathrm{h_{t-1}}, \mathrm{x_t}] + \mathrm{b_f}) \tag{12}$$

$$\mathrm{i_t} = \sigma(\mathrm{W_i} \cdot [\mathrm{h_{t-1}}, \mathrm{x_t}] + \mathrm{b_i}) \tag{13}$$

$$\tilde{\mathrm{C}}_\mathrm{t} = \tanh(\mathrm{W_C} \cdot [\mathrm{h_{t-1}}, \mathrm{x_t}] + \mathrm{b_C}) \tag{14}$$

$$\mathrm{C_t} = \mathrm{f_t} * \mathrm{C_{t-1}} + \mathrm{i_t} * \tilde{\mathrm{C}}_\mathrm{t} \tag{15}$$

$$\mathrm{o_t} = \sigma(\mathrm{W_o} \cdot [\mathrm{h_{t-1}}, \mathrm{x_t}] + \mathrm{b_o}) \tag{16}$$

$$\mathrm{h_t} = \mathrm{o_t} * \tanh(\mathrm{C_t}) \tag{17}$$

### 2.3 Decoder

The decoder takes the encoder representation learned in the memory network and predicts the operands and operators sequence. The decoder is initialized with the last hidden state and cell state

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


32nd Pacific Asia Conference on Language, Information and Computation
Hong Kong, 1-3 December 2018

from the encoder. The decoder also uses an LSTM to predict the next output. The decoder predicts the output distribution using teacher forcing (Lamb et al., 2016). The hidden and cell states are computed according to the LSTM equations defined in section 2.2. In Figure 1, the output tokens are referred to as Op1, Op2 and Opr which stand for the operands and the operator of the equation.

## 3 Experimental Setup

### 3.1 Data

We used 1314 arithmetic problems with a single operation present in MAWPS Koncel-Kedziorski et al. (2016) as our training set. The operations include all basic mathematical operation addition (+), subtraction (-), multiplication (*) and division (/). The two benchmark datasets for evaluation are MA1 and IXL dataset Mitra and Baral (2016) which are subsets of the AI2 dataset Hosseini et al. (2014) released as a part of project Euclid [2]. We chose problems with only single operation from these two datasets - 103 from MA1 and 81 from IXL dataset.

### 3.2 Preprocessing

Every number appearing in a word problem is replaced by $num_i$ in a random fashion where $i \, \varepsilon \, [1, 6]$. This is done to minimize the sparsity of different kinds of numbers appearing in the problem text. This also assisted in learning better representations of numbers in word problems. Each word problem was divided into two parts - problem text or supporting sentences and question text which is shown in 2. NLTK (Bird and Loper, 2004) was used to tokenize the sentences in a word problem. The last sentence in the list of tokenized sentences was considered as the question sentences and rest as supporting sentences. The equations were labeled in postfix notation e.g. *num1 num3 +*

### 3.3 Setting

We used publicly available glove pre-trained embeddings [3] that were trained on Common Crawl containing 42 billion tokens, with a vocabulary size of 1.9 million. The development set was fixed to 5% of the training data. The embedding weights for these were uniformly initialized. Dropout (Srivastava et al., 2014) rate of 0.2 was used while learning the embeddings A, B and C. The embedding and the LSTM output dimensions were set to be 64. The concatenated vector representation for each word is a vector of size 364. Maximum number of words appearing in the supporting sentences and question sentences were 56 and 34 respectively. No dropout rates were specified for the LSTMs. The recurrent weights were initialized as an random orthogonal matrix. The input weights were assigned from a Glorot (Glorot and Bengio, 2010) uniform distribution with bias being initialized to zeros. Following this procedure, we were able to reduce the output vocabulary size. The encoder and decoder hidden and cell state was also fixed to be of dimension 64. Keras[4] (Chollet and others, 2015) deep learning li-

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

Table 2: Preprocessing Steps For Word Problems

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

Table 3: Frequency Analysis of Operations in Training Data

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

[3] http://nlp.stanford.edu/data/glove.42B.300d.zip

[4] https://keras.io/


32nd Pacific Asia Conference on Language, Information and Computation
Hong Kong, 1-3 December 2018

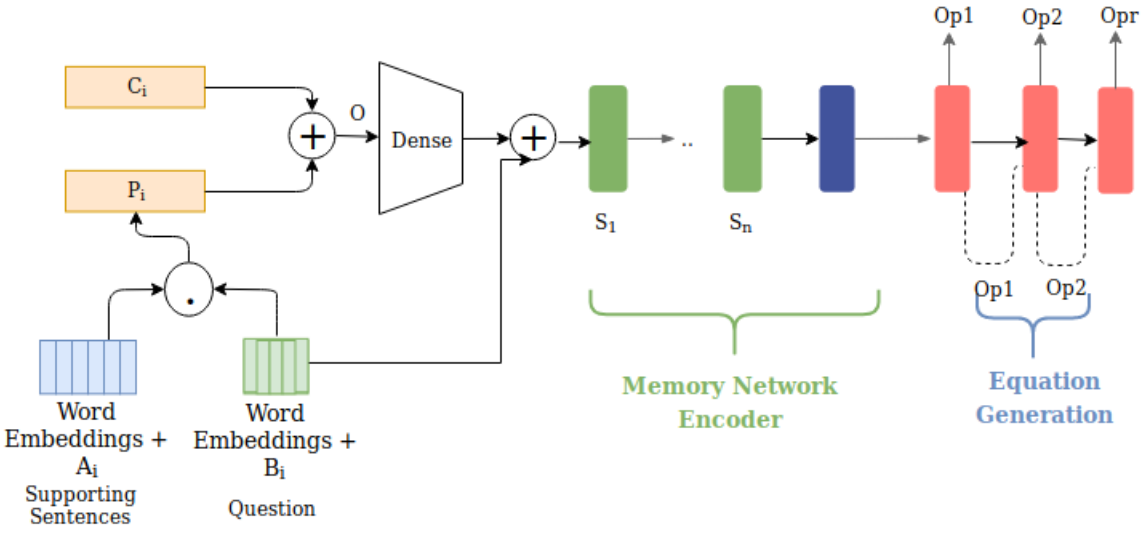



Figure 1: Architecture Diagram Of EquGener for Single Layer

brary was used to build the network. Adam (Kingma and Ba, 2014) optimizer was used for optimization of the parameters. The system was trained for 50 epochs with validation set. In case of multiple hops, same embeddings A and C were used in different layers.

$$A_1 = A_2 = .. = A_K \quad (18)$$

$$C_1 = C_2 = .. = C_K \quad (19)$$

$$u_{k+1} = u_k + o_k \quad (20)$$

We experimented for only two hops.

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

## 4 Results and Observation

### 4.1 Comparison With Other Systems

We compared our system with other systems on MA1 and IXL datasets. Table 4 shows the accuracy of the systems in solving word problems in terms of percentage of problems solved. In terms of overall performance our system beats all the state-of-the art systems. *EquGener* beats ARIS by a big margin. ARIS uses rigid templates for various verbs and rely on different external resources. *EquGener* implicitly handles all kinds of varieties of real world scenarios in a continuous space and can be more effective than discrete fixed templates.

| System | MA1 | IXL | Avg |
|---|---|---|---|
| ARIS(2014) | 83.6 | 75.0 | 79.3 |
| KAZB(2014) | 89.6 | 51.1 | 70.35 |
| ALGES(2016) | - | - | 77.0 |
| (Roy and Roth, 2016b) | - | - | 78.0 |
| (Mitra and Baral, 2016) | 96.27 | 82.14 | 89.20 |
| Att. encoder-decoder | 92.23 | 71.61 | 81.92 |
| *EquGener* 1 hop | 91.26 | **85.19** | 88.22 |
| *EquGener* 2 hops | 94.18 | **85.19** | **89.68** |

Table 4: Comparison Of proposed system with other systems. Numerical values represent % of problems solved

*EquGener* outperforms KAZB significantly. KAZB uses a joint log-linear model distribution over a full system of predefined equations and alignments between the text and equation templates. As the alignment space is exponential while aligning with the slots, beam search is employed to find approximate solution. KAZB (2014) uses surface level features for the words, does not employ any semantic representation of them. So KAZB performs poorly on IXL where there is information gap and irrelevant quanitities. *EquGener* makes use of the dense semantic representation and can identify irrelevant quantities easily. *EquGener* improves upon ALGES (2015) by 12% in terms of overall accuracy. ALGES solves word problems by

generating equation trees and scoring them. The tree that best represents the computation inside the problem is scored highest. It employs a quantity schema and uses grammatical features to find to form quantity sets. Integer Linear Programming is applied on the quantity sets to find type-consistent equation trees. Dependence on external parsing tools, lack of semantic knowledge attributes to the errors of the system. *EquGener* is independent of any external tool usage and performs better in isolation. Similarly, Roy and Roth (2016b) solve the word problems through equation tree generation. Two classifiers are used - one to find relevant quantities and other to find the operator as the Least Common Ancestor between them. Certain constraints are imposed on tree generation, the tree with highest score is chosen as the equation tree. Incorrect equations are generated due to erroneous quantity extraction. *EquGener* does not depend on any extraneous information like grammatical and dependency features and is able to identify words which denote the quantities and operation. Mitra and Baral (2016) is the current state-of-the-art system. *EquGener* beats its overall accuracy by 0.48%. Mitra and Baral (2016) classify each addition or subtraction problem into 3 concepts and each concept is associated with a formula. Different features are defined for each formula. Multiple formulas can be applied to a word problem, a log linear model scores each formula based on its features. The biggest limitation of this system is its reliance on external tools like wordnet, dependency parser and conceptnet. The system can only solve addition and subtraction problems. *EquGener* does not need any computation of extra features to solve word problems. It can solve problems with any arithmetic operation. Our baseline system performs very poorly on IXL on problems with irrelevant quantities. The attention encoder-decoder decodes the equation based on the whole sequence, the system has to find the attention weights on all the words including question words. *EquGener* on the other hand generates the equation based on the question.

### 4.2 Operand Prediction

The operand prediction accuracies improve with number of hops. *EquGener* requires several hops to identify the exact operands correctly. The major error in operand prediction in our system resulted from the predictions that were out-of-order. The accuracy of the relevant quantity or operand prediction (Roy and Roth, 2016b) was 89.1%. *EquGener* improved upon this accuracy by 3%.

### 4.3 Operator Prediction

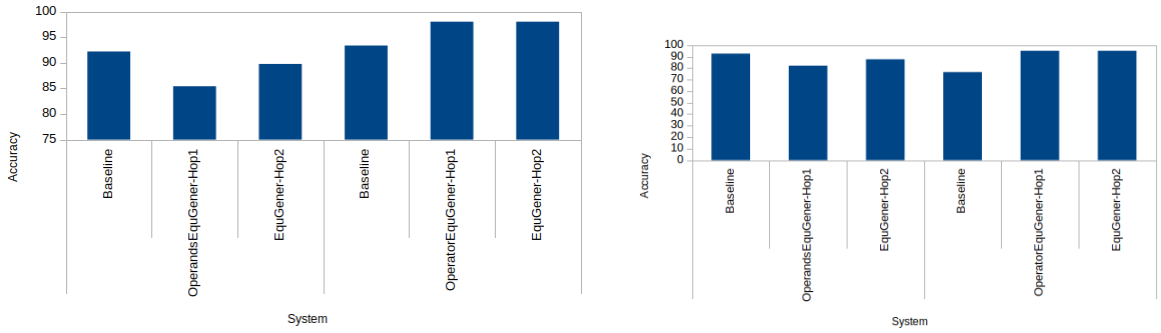
*EquGener* outperforms the baseline system in operator prediction by significant margin. This strengthens our hypothesis that memory networks are better at capturing the intention of verbs that have direct correlation to arithmetic operations. The accuracy in verb categorization by ARIS (Hosseini et al., 2014) was 81.2%. The accuracy reported for LCA operation prediction was 88.7% with all features (Roy and Roth, 2016b). *EquGener* outperforms these two systems in operator prediction by a big margin with operator prediction accuracy 97% on an average. As the training set contains problems containing all operations, the baseline model predicted multiplication and division operations though the IXL dataset only contained AddSub problems. This prediction anomaly was not observed in case of *EquGener*.

## 5 Error Analysis

Some of the erroneous output produced by *EquGener* are shown in the Table 5. In the first example system is able to identify operators and operation accurately. However system could not identify the direction of transfer for verb 'borrow' which resulted in an erroneous prediction of order of equation components. We observe that this is a problem due to data sparsity and can be overcome by making the dataset more dense. In the second example system predicted an operand *num5* which is present nowhere in problem description. Though we expect the numerical values mentioned in the problem description not to be attached to any context, repeated occurrence may violate this assumption in certain cases. An architectural improvement to handle the numerical values can improve the results.

## 6 Discussion

The figure 2 below shows the relative attention of words in supporting sentences conditioned on question words. The words in the question are shown




(a) Operand and Operator Predictions for MA1

(b) Operand and Operator Predictions for IXL

| TestSet | Question | Predicted | Actual |
|---|---|---|---|
| MA1 | Joan picked *num3* apples from the orchard . Melanie borrowed *num1* apples from her . How many apples does Joan have now ? | *num1 num3 -* | *num3 num1 -* |
| IXL | Tom went to *num1* hockey games this year , but missed *num4* . He went to *num3* games last year . How many hockey games did Tom go to in all ? | *num1 num5 +* | *num1 num3 +* |

Table 5: Predicted and Actual Equations from Different Test Sets

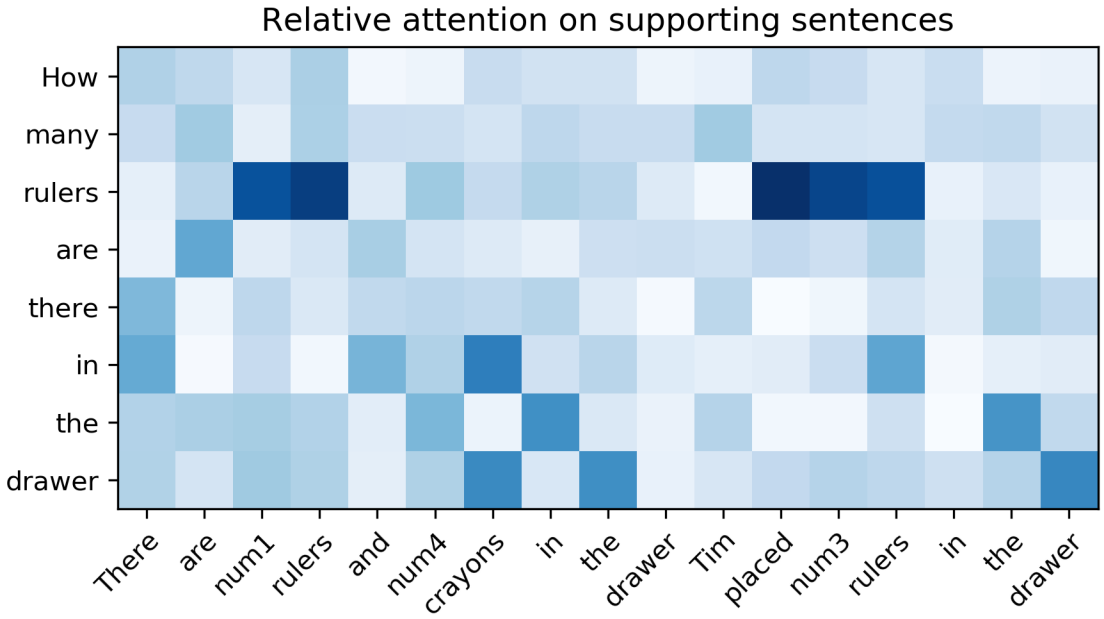



Figure 2

in Y-axis and the words in the X-axis constitute the supporting sentences. *EquGener* is able to figure out *num4* as an irrelevant quantity as it is associated with the entity crayons whereas the question is

asked about the rulers. The verb "place" appears in the context of the rulers, so it also receives higher weights.

## 7 Previous Work

Mukherjee and Garain (2008) explained different techniques used for word problem solving in their survey paper. Bobrow (1964) represented word problems in terms of a relational model. Bakman (2007) touched upon understanding of word problems involving extraneous information. Multiple approaches like template alignment, verb categorization, CFG rules can be used to solve word problems. Liang et al. (2018) suggested that there exists mainly two kinds of approaches for solving MWPs - one involving understanding and the other without understanding. Kushman et al. (2014) system was a joint log linear distribution over the full set of equations and alignments between the variables and text. The number of equations was dependent on the number of training equation templates. The number slots were filled by the numbers present in the text while the unknown or variable slot were filled by the nouns in the problem text. Huang et al. (2017) also extracted relevant templates and did fine grained inferencing to solve word problems. Illinois Math Solver (Roy and Roth, 2016a) used two modules to solve any arithmetic word problem. The first module was a CFG based Semantic Parser and other module solved the problem by decomposing it into a series of classification problems (Roy and Roth, 2016b) with formation of an expression tree through constrained inference. Hosseini et al. (2014) 's system used verb categorization for identifying relevant variables, their values and mapping them into a set of linear equations which can be easily solved. The system identified 7 kinds of verbs used in the problems which was predicted by support vector machines. Mitra and Baral (2016) created an arithmetic word problem solver that learned how to use formulas to solve simple addition and subtraction problems. Templates corresponding to particular formulas were manually modeled with pre-defined slots. All possible applications of different formulas were passed through a log-linear model to pick the best solution with highest score. The features to the model were dependency labels by running Stanford dependency parser, POS tags, some linguistic cues, Wordnet (Miller, 1995) features. Recently there have been renewed interest in solving word problem through deep learning techniques. Huang et al. (2018) used intermediate meaning representation to generate equations. There has been attempts (Ling et al., 2017) to generate answer rationales for arriving at the final answer. Wang et al. (2017) used a hybrid model of RNN and similarity-based information retrieval methods to outperform existing solvers. They used a 5 layer deep network - one word embedding layer, two-layer GRU (Chung et al., 2014) on the encoder side and two-layer LSTM (Hochreiter and Schmidhuber, 1997) as the decoder. A modified version of the sequence-to-sequence model was used to validate each generated output symbol with the help of some hand-written rules and modified the softmax function on the decoder side. There have been a concerted effort to create large scale and diversified datasets. Huang et al. (2016) analyzed the existing datasets, and created a large-scale dataset from community question answering web pages. Most of the neural network frameworks suffer from lack of performance in presence of a limited training dataset. (Weston et al., 2014) introduced long term memory components in neural networks for better reasoning called "memory networks". The memories can be retrieved and written multiple times and can be used for prediction in multiple tasks. (Sukhbaatar et al., 2015) came up with an end-to-end memory network which required less supervision in giving answers to a question asked from a story like setting. In this paper, we trained the system on a training set containing word problems with single operations and showed that memory network based architecture was able to learn to generate such equations.

## 8 Conclusion and Future Work

We proposed an end-to-end memory network based encoder decoder which was able to generate single equations for simple word problems. The system registered improvement in performance in presence of multiple hops. We can also extend our work for word problems with multiple operations which requires the decoder to generate multiple equations. We can also explore techniques to handle OOVs suggested in (Weston et al., 2014) taking into account the feature representation of the context where OOVs appear.

## References


Yefim Bakman. 2007. Robust understanding of word problems with extraneous information. *arXiv preprint math/0701393*.

Steven Bird and Edward Loper. 2004. Nltk: the natural language toolkit. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*, page 31. Association for Computational Linguistics.

Daniel G Bobrow. 1964. 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. ACM.

François Chollet et al. 2015. Keras. `https://github.com/fchollet/keras`.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Denise Dellarosa. 1986. A computer simulation of childrens arithmetic word-problem solving. *Behavior Research Methods, Instruments, & Computers*, 18(2):147–154.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256.

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

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

Danqing Huang, Shuming Shi, Chin-Yew Lin, Jian Yin, and Wei-Ying Ma. 2016. How well do computers solve math word problems? large-scale dataset construction and evaluation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 887–896.

Danqing Huang, Shuming Shi, Chin-Yew Lin, and Jian Yin. 2017. Learning fine-grained expressions to solve math word problems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 805–814.

Danqing Huang, Jin-Ge Yao, Chin-Yew Lin, Qingyu Zhou, and Jian Yin. 2018. Using intermediate representations to solve math word problems. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 419–428.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

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

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. Mawps: A math word problem repository. In *Proceedings of NAACL-HLT*, pages 1152–1157.

Nate Kushman, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay. 2014. Learning to automatically solve algebra word problems. Association for Computational Linguistics.

Alex M Lamb, Anirudh Goyal ALIAS PARTH GOYAL, Ying Zhang, Saizheng Zhang, Aaron C Courville, and Yoshua Bengio. 2016. Professor forcing: A new algorithm for training recurrent networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4601–4609. Curran Associates, Inc.

Chao-Chun Liang, Yu-Shiang Wong, Yi-Chung Lin, and Keh-Yih Su. 2018. A meaning-based statistical english math word problem solver. *arXiv preprint arXiv:1803.06064*.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. *arXiv preprint arXiv:1705.04146*.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

George A Miller. 1995. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41.

Arindam Mitra and Chitta Baral. 2016. Learning to use formulas to solve simple arithmetic problems. In *ACL (1)*.

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

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

Subhro Roy and Dan Roth. 2016a. Illinois math solver: Math reasoning on the web. In *HLT-NAACL Demos*, pages 52–56.

Subhro Roy and Dan Roth. 2016b. Solving general arithmetic word problems. *arXiv preprint arXiv:1608.01413*.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. 2015. End-to-end memory networks. In *Advances in neural information processing systems*, pages 2440–2448.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

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

Jason Weston, Sumit Chopra, and Antoine Bordes. 2014. Memory networks. *CoRR*, abs/1410.3916.