Solving Math Word Problems

Ryan Wong
Department of Electrical Engineering
Stanford University
Palo Alto, CA 94305
rawong@stanford.edu

Abstract

This project focused on the development of a deep natural language processing (NLP) model that can solve a broad range of math problems posed in text by predicting an appropriate set of equations, from which the correct answer can be derived. An attention-based sequence to sequence (seq2seq) model was the best performing model, exhibiting test accuracy of 12.8% (correct numerical answers) and BLEU of 48.10 (equations predicted) on the Dolphin18k dataset. Despite its relatively low accuracy, the model learnt mathematical syntax reasonably well, with 80.5% of predicted equations being syntactically correct. Character-level representation of numbers (vs. word level) and dataset augmentation were key contributing factors to improving performance.

1 Introduction

Math word problems form a natural abstraction to a range of quantitative reasoning skills that are often required of an autonomous agent to behave intelligently, such as understanding financial news, calculating change in everyday transactions, and analyzing productivity in an organization. Solving such problems remains a key challenge for artificial intelligence (AI) today, as they involve recognizing how goals, entities and quantities in the real-world map into a mathematical formalization.

2 Related Work

Given dataset limitations, the majority of precedent research have applied statistical learning approaches with manual to semi-manually-picked features [1, 2, 3] and rule-based semantic parsers [4]. However, despite some progress, Huang et al. [5] demonstrated that the performance of such techniques is still quite low on larger datasets with a greater diversity of problems.

Google DeepMind [6] recently trained an attention-based sequence to sequence (seq2seq) model that was able to achieve 36% accuracy on four-way multiple choice word problems and also provide an accompanying answer rationale.

Tencent AI [7] recently demonstrated promising results using a hybrid sequence to sequence recurrent neural network (RNN) trained on a dataset of ~20,000 Chinese word problems that achieved ~65% accuracy, outperforming other state of the art statistical learning methods, such as ZDC in [3]. However, this was restricted to single unknown, linear problems.

Inspired by the promising results of Google and Tencent AI, this project aimed at developing a seq2seq model that can solve a broader range of math word problems (not just single unknown, linear problems) using the relatively new Dolphin18k dataset [5].

3 Problem formulation

A math word problem P is a sequence of words (question text) that describe a problem to be solved using mathematical reasoning (refer to Table 1 below). Given P, the model's sole objective is to generate a sequence of mathematical tokens (numbers and operators) that represent an appropriate set of equations E (the "Problem"), from which the correct numerical answer/s A can be derived using a third-party solver package. i.e. the model's goal is to accurately map P to E. This project adopted the popular Python library, Sympy [8] as the third-party solver package.

Table 1: Example maths word problem

Problem (*P*)	Find 2 numbers whose sum and product are 11.
Equations (*E*)	x + y = 11 x * y = 11
Answer (*A*)	x=1.113; y=9.887

4 Dataset and preprocessing

4.1 Dolphin18k dataset

The Dolphin18k dataset [5], prepared by Microsoft Research, contains 18,460 math problems extracted from Yahoo Answers!. The dataset contains a broad range of single-to-multi-unknown linear and non-linear word problems of elementary to high school difficulty.

Using Dolphin18k, I selected 13,273 examples that had gold standard equations and were able to be parsed by Sympy. The dataset was then split into three parts: training set (10,591 examples), development set (1,486 examples) and a test set (1,196 examples). The median question text length was 31 words, with the majority of problems containing one or two-unknowns (~85%). Figures 1 and 2 below illustrate the dataset statistics.

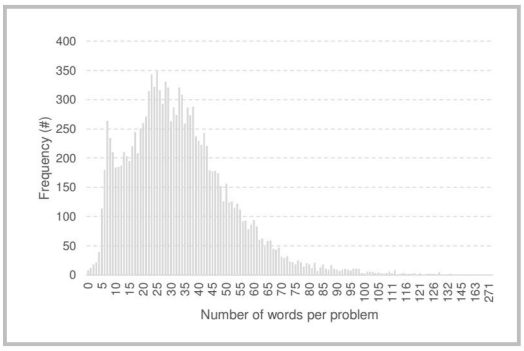

Figure 1: Histogram of question text length

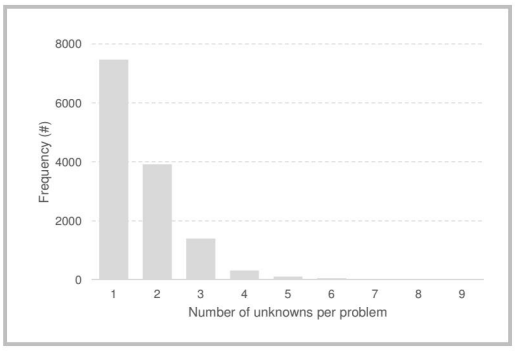

Figure 2: Histogram of no. of unknowns

4.2 Pre-processing

Question text and gold standard equations were first pre-processed prior to training, validation and testing.

All numerical quantities in question text and equations were converted to numbers and rounded to three decimal places. This was performed to standardize numerical representation, thereby reducing overall vocabulary size and encouraging the model to learn shared embedding representations. For the character-level seq2seq models (refer to 5.2 and 6.3.2), number representations were at the character level (vs. word level) and terminated with a special number end token (<N_END>).

All equations were converted to Polish notation to reduce operator ambiguity and make Sympy parsing of predicted equations easier. In addition, answer variables (primary unknowns) were relabeled as u_0, u_1, …, u_i. All other auxiliary unknowns were relabeled as x_0, x_1, …, x_j. Special stop tokens were inserted at the end of each equation as a delimiter (<EQN_END>).

5 Models

Two model architectures were explored: a simple similarity model as a baseline and seq2seq. Multiple variants of the latter were investigated to optimize performance.

5.1 Similarity model

I implemented a similarity model akin to the one in [5] as a baseline. The model computes the solution to a math word problem ("query") using the set of equations ("equation template") of the most similar question in the training set. More specifically:

1. Based on a bag of words (BoW) featurization of question text, it computes the pair-wise distance between the query and every example in the training set;
2. retrieves the equation template of the question (in the training set) that has the smallest distance (i.e. most similar to the query);
3. extracts the significant numbers from the query question text; and then
4. substitutes these significant numbers into the equation template identified in step 2.

5.2 Seq2seq model

Figure 3 below shows the base architecture of the two-layer attention-based seq2seq model used in this project. The sequence of question text words are first converted to word embeddings prior to input into the encoder network. Based on the encoder's hidden states and the attention mechanism, the decoder network then predicts the appropriate sequence of mathematical symbols representing a set of equations. LSTM units were used given their robustness to vanishing / exploding gradient issues and the improved scaled Luong method was used as the attention mechanism.

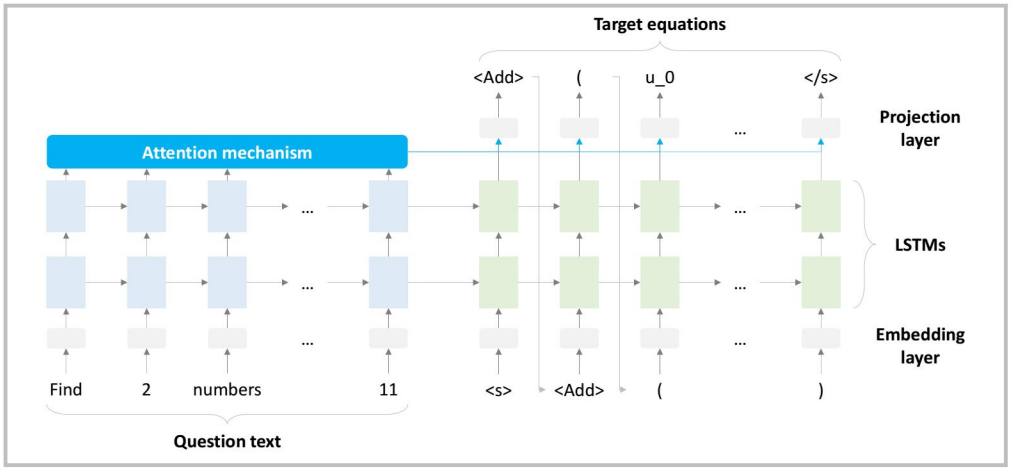

Figure 3: Seq2seq base architecture

The Tensorflow Neural Machine Translation tutorial codebase [9] was used as a foundation for development and training of the seq2seq models.

The loss function was set to the cross-entropy loss over the decoder outputs with respect to

the gold standard equations. Models were trained using the Adam optimizer with an initial learning rate of 0.001 and batch size of 64 over 50-70 epochs. Adam was selected as its adaptive learning rate minimizes the sensitivity to initial learning rate selection. Embeddings were initialized using pretrained GloVe 100 dimensional word vectors [10] and also optimized during the training process.

Inference was performed by feeding the model the sequence of question text tokens and performing either greedy or beam search on the decoder to generate the predicted equations.

A wide range of seq2seq variants were investigated, including:

- **Bidirectional encoder** to improve representations of question text by allowing the encoder to have forward and backward information at each step.
- **Decoder vocabulary truncation** to just the mathematical symbols required to generate the equations (as opposed to the entire vocabulary).
- **Character level model** (vs. word level) whereby numbers are represented at the character level rather than word level. This was to achieve a compact vocabulary that can theoretically express any number. Word representation maintained at the word level.
- **Dataset augmentation** of the training set with permuted versions of each example. Permutation involved changing a single number in the question text (and respective position in equations) with a uniformly random number between -100 and 100. This was explored to encourage the model to more effectively learn number mappings from question text to equations and improve generalization to unseen numbers.
- **Beam search** for the decoder output (vs. the default greedy approach).

5.3 Evaluation metrics

The models were evaluated based on the following metrics:

- **Accuracy**: percentage of problems (out of 100%) that the model can generate the correct final numerical solution. A solution was deemed to be correct if all quantities were within +/- 10% of the gold standard answer/s.
- **BLEU**: the corpus-level BLEU score (out of 100) of the predicted equations relative to the gold standard equations for n-grams of up to length 4.

Given the “all-or-nothing” nature of the accuracy metric, BLEU was chosen as an additional metric to give a sense of how well the model is able to generate meaningful systems of equations (despite potentially resulting in the incorrect numerical solution). In addition, BLEU was used to save the best performing model during training given its ease of computation, whilst accuracy was used for offline hyper-parameter and architecture tuning.

6 Results and discussion

6.1 Summary of results

Table 2 below summarizes the models’ performance on the training and test sets. The best performing model was the character level seq2seq model with data augmentation and beam search (model 10 highlighted bold in the table), exhibiting a test accuracy of 12.8% and test BLEU of 48.10 for its predicted equations, almost tripling the performance of the baseline.

Figure 4 below illustrates the incremental improvements made to the vanilla seq2seq to achieve the best performing model. The transition to a character-level model resulted in the largest incremental improvement in test accuracy of +8.2%.

Overall, accuracy is relatively low reflecting the complexity of the Problem (infinite solution space) and the “all-or-nothing” nature of the accuracy metric. However, it is interesting to note the moderately high BLEU score – the model learnt mathematical syntax reasonably well despite not getting the correct numerical answer.

Table 2: Summary of all models' validation and test performance.

	BLEU		Accuracy (%)	
	Dev	**Test**	**Dev**	**Test**
Baseline				
1. Similarity model	15.10	16.21	4.24	4.35
Seq2seq models				
2. Uni encoder 2 x 128 units (vanilla seq2seq)	35.97	36.40	1.14	1.17
3. Bi. encoder 2 x 128 units	38.02	40.26	1.48	2.17
4. Bi. encoder 2 x 256 units	40.81	40.61	3.90	3.18
5. Bi. encoder 2 x 512 units	39.15	39.96	2.56	2.42
6. Char. model 2 x 256 units	46.76	48.54	6.32	7.78
7. Char. model 2 x 512 units	48.56	48.50	9.35	10.62
8. Char. model 2 x 1,024 units	48.51	48.22	7.67	8.95
9. Char. model 2 x 512 units + data aug.	48.69	49.29	10.22	11.54
10. Char. model 2 x 512 units + data aug. + beam	**47.30**	**48.10**	**11.70**	**12.79**

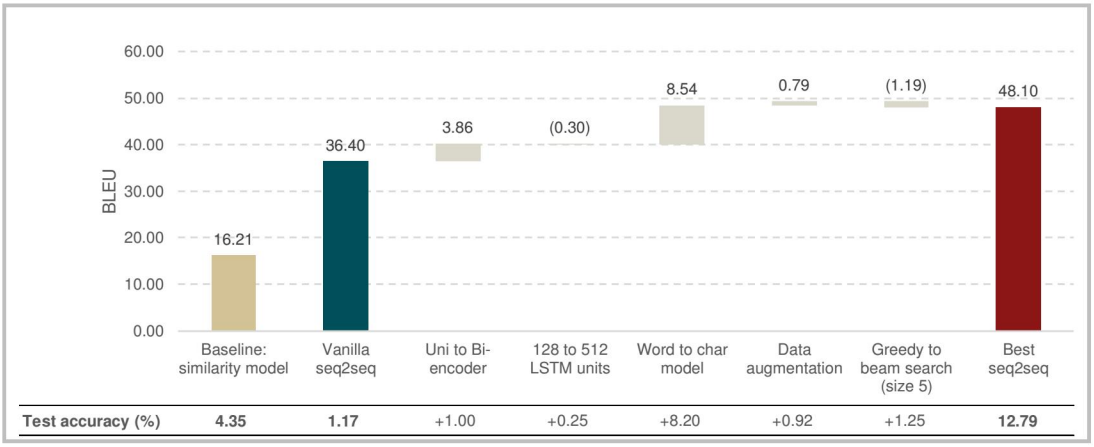

Figure 4: Bridge of test set performance from vanilla seq2seq to best performing model.

6.2 Baseline: similarity model

The baseline similarity model only achieved 4.35% test accuracy and test BLEU of 0.162. The relatively poor performance is due to a number of reasons:

- **Rule-based**: model relies on brittle, rule-based parsing of question text and equations.
- **BoW representation**: features cannot capture any sense of ordering, which is important in maths problems. e.g. "one divided by two" ≠ "two divided by one"
- **Disjoint optimization**: model not optimized end-to-end in a data driven manner. e.g. the question text parser and significant number extractor were built independently and not optimized jointly.
- **Fixed set of equation templates**: model restricted to the set of equation templates in the training set. Hence, it does not generalize well to unseen problems.

6.3 Seq2seq model

The seq2seq model materially improved test accuracy to 12.8% by overcoming a number of these baseline limitations – optimization was performed in an end-to-end manner without relying on feature engineering or a fixed set of equation templates. Sections 6.3.1 to 6.3.5 detail the key seq2seq improvements and observations.

6.3.1 Bidirectional encoder and decoder vocabulary truncation

A bidirectional encoder (instead of unidirectional) with truncation of decoder vocabulary (to just mathematical symbols) improved test accuracy by ~1.0% and test BLEU by ~3.86.

Bidirectionality improved performance as it enabled the model to learn better representations for question text based on past as well as future context. This intuitively makes sense as we (as humans) typically must read the entire question text, taking into account both words before and after numerical quantities to determine an appropriate mapping into equations.

Truncation of the decoder vocabulary to just the mathematical symbols improved performance by compressing the vocabulary from 9,061 to 1,376. Thus reducing the output space and eliminating redundant parameter weights (i.e. weights for non-mathematical symbols, such as entity names, that would never be generated by the decoder).

6.3.2 Character-level model

However, when analyzing the word-level model, ~17% of predicted equations were incorrect simply due to the presence of one or more unknown tokens due to the limited decoder vocabulary (refer to Table 3 below). Encoding numbers at the word-level is problematic given the continuous (and hence infinite) space of possible numbers.

Table 3: Test set performance of word-level and character-level models

	Word model (Model 5)	**Char. model (Model 7)**	**Char. model with aug. (Model 9)**	**Char. model with aug. and beam (Model 10)**
Syntactically correct eqns. (%)	91.8%	70.2%	70.0%	80.5%
Equations with unknowns (%)	17.2%	-	-	-
BLEU	39.96	48.50	49.29	48.10
Accuracy (%)	2.42	10.62	11.54	12.79

Hence, a character-level model was investigated, whereby numbers are encoded at the character level (rather than word level) – enabling the model to express any number using a compact, discrete vocabulary. Representation of words were maintained at the word-level. This reduced the decoder vocabulary size from 1,376 to just 41 symbols and resulted in a material improvement in test accuracy of 8.2% and test BLEU of 8.54.

However, the character-level model proved to be a “double-edged sword”. While eliminating the unknown token issue, the percentage of syntactically correct equations dropped from ~92% to ~70%. This reduction is likely due to the greater freedom afforded by the character-level model and its weaker notion of a complete number as a whole (since they are now represented as a series of characters).

6.3.2 Dataset augmentation

The training set was increased from 10,591 to 45,371 by performing dataset augmentation (as described in section 5.2) with a target of four permutations per example. Training on the augmented dataset improved test accuracy by 0.92% and test BLEU by 0.79. This technique was inspired by the analogue of data augmentation in computer vision, where a variety of transformations are often performed to the original images, such as rotation, scaling and cropping, to improve model robustness. The observed performance uplift suggests that the simple dataset augmentation employed here also increased model generalization.

6.3.3 Optimization of hyper-parameters

Beam search (with a beam size of 5) improved validation set accuracy from 10.22% (greedy) to 11.70% (refer to Figure 5). It also materially improved the percentage of syntactically correct equations from 70.0% to 80.5% (refer to Table 3 above). A beam size of 5 was selected given the immaterial improvement observed for larger beam sizes. Beam search improved performance by adopting a longer-term view to generating output (rather than the myopic greedy view of taking the most likely prediction at each step).

The number of LSTM units was selected to be 512 (refer to Figure 5 below). Reduced validation set performance was observed for more units, suggesting overfitting.

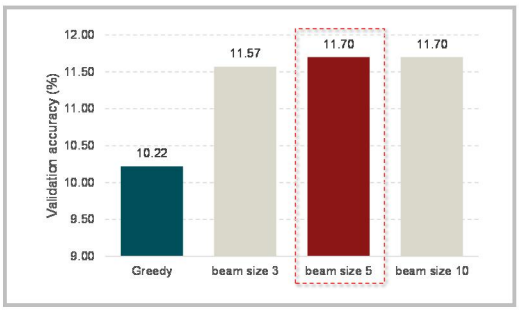

Figure 5: Val. accuracy vs. beam size

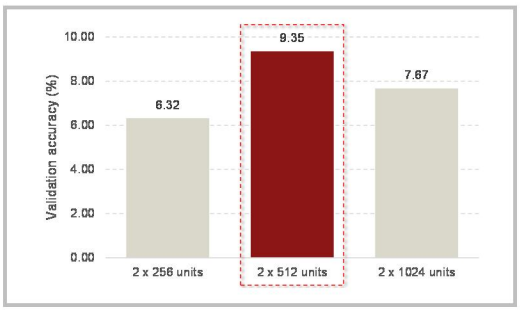

Figure 6: Val. accuracy vs. number of units

6.3.4 Attention mechanism

The attention mechanism proved to be particularly helpful in mapping numerical quantities in question text to equations by improving the model's access to all locations within the encoder sequence, thus mitigating the information bottleneck issue.

Figure 7 below shows attention heat maps for two example problems during inference using the character-level seq2seq model. The question text (input) is along the left vertical axis, while the predicted equations (output) are along the bottom horizontal axis. As you can see, the regions of higher attention (denoted by the brighter colors) during generation of numbers in the equations correspond to their respective location in the question text. e.g. in the left chart, during the prediction of "180" in the equations, the model was attending the most to "$180" in the question text.

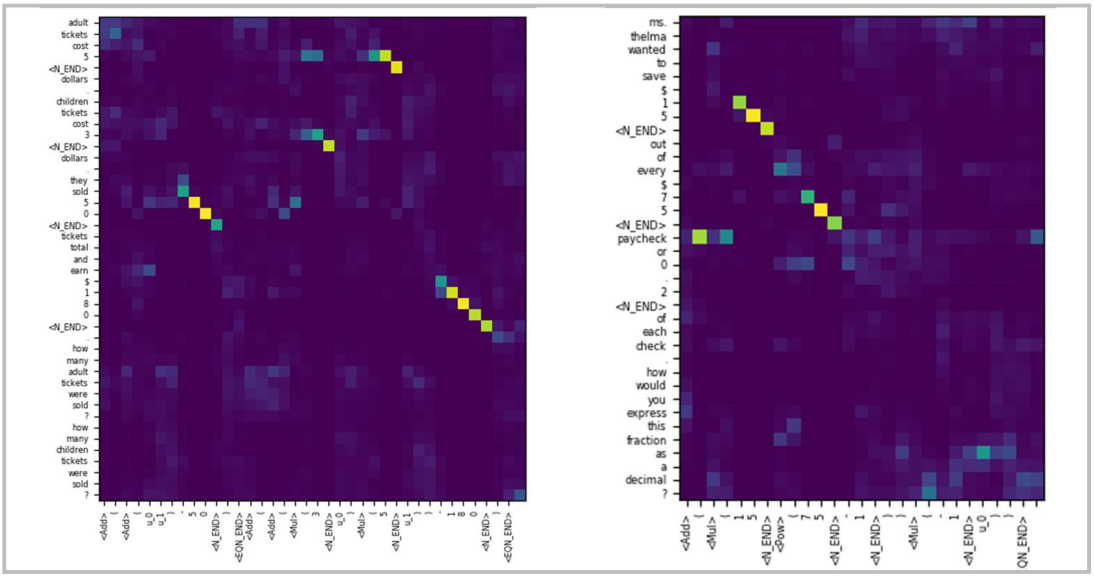

Figure 7: Attention heat maps for two test examples.

6.3.5 Shared embedding space

The model learnt a semantic representation of the joint encoder-decoder embedding space. Figure 8 below shows a visualization of the decoder embeddings learnt by the character-level model (encoder embeddings excluded for clarity) using principal components analysis (PCA). As you can see, the model learnt to map mathematical symbols into semantic groups – operators (denoted in green) in the top right hand corner, digits (denoted in orange) in the bottom right and unknowns (denoted in blue and red) predominantly in the bottom left.

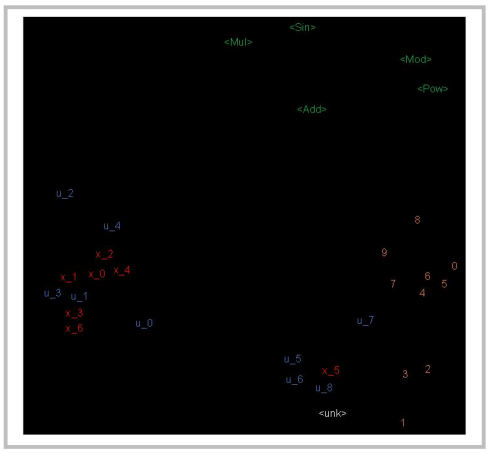

Figure 8: Decoder embeddings visualization using PCA

7 Conclusion

A deep NLP model that can solve math word problems posed in text has been developed. An attention-based seq2seq model achieved the best performance with test accuracy of 12.8% and test BLEU of 48.10, materially outperforming a baseline similarity model (4.35% test accuracy). Despite the relatively low accuracy, the model exhibited a strong sense of mathematical syntax with ~80% of predicted equations syntactically correct. Character level representation of numbers and dataset augmentation were key factors to improving performance.

Potential areas for further work include exploration of techniques to improve mapping of numerical quantities in question text to equations, such as different attention mechanisms, neural network based significant number extractor, or Pointer networks.

References

[1] Hosseini, Mohammad Javad, et al. "Learning to solve arithmetic word problems with verb categorization." Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). 2014.

[2] Kushman, Nate, et al. "Learning to automatically solve algebra word problems." Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Vol. 1. 2014.

[3] Zhou, Lipu, Shuaixiang Dai, and Liwei Chen. "Learn to solve algebra word problems using quadratic programming." Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing. 2015.

[4] Shi, Shuming, et al. "Automatically solving number word problems by semantic parsing and reasoning." Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing. 2015.

[5] Huang, Danqing, et al. "How well do computers solve math word problems? Large-scale Dataset construction and evaluation." Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Vol. 1. 2016.

[6] Ling, Wang, et al. "Program induction by rationale generation: Learning to solve and explain algebraic word problems." arXiv preprint arXiv:1705.04146 (2017).

[7] Wang, Yan, Xiaojiang Liu, and Shuming Shi. "Deep Neural Solver for Math Word Problems." Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing. 2017.

[8] Meurer A, Smith CP, Paprocki M, Čertík O, Kirpichev SB, Rocklin M, Kumar A, Ivanov S, Moore JK, Singh S, Rathnayake T, Vig S, Granger BE, Muller RP, Bonazzi F, Gupta H, Vats S, Johansson F, Pedregosa F, Curry MJ, Terrel AR, Roučka Š, Saboo A, Fernando I, Kulal S, Cimrman R, Scopatz A. SymPy: symbolic computing in Python, PeerJ Computer Science 3:e103 (2017)

[9] Minh-Thang Luong, Eugene Brevdo & Rui Zhao (2017). Neural Machine Translation (seq2seq) Tutorial. https://github.com/tensorflow/nmt.

[10] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation.