Evaluate and Switch between LaTeX and Natural Mathematical Expressions with Encoder-Decoder

Zeyun Wu
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, CA 92093
zew038@ucsd.edu

Abstract

This paper introduces a pilot study of evaluating and translation between LaTeX and natural mathematical expressions. We implemented algorithms that generate LaTeX and natural expressions by defining rules and permutating elements. We utilized encoder-decoder networks with attention mechanism to translate LaTeX expression to natural expression with high performance. Some critiques and future direction were discussed and the study was still preliminary.

1   Introduction

LaTeX (L^AT_EX) is a typesetting software system that allows users to gain scalable, systematic, and consistent presentation of contents. It is commonly used by people from all fields in various circumstances, including writing this paper. It is especially useful in formatting mathematical expressions. For instance, this is an example of well-organized presentation of LaTeX equation computing gradient during backpropagation when training neural network.

$$\nabla_{\mathbf{w}^l} L[\eta(x;\theta), y] = \frac{\partial L[\eta(x;\theta), y]}{\partial z^l} \frac{\partial z^l}{\partial \mathbf{w}^l}$$

While LaTeX is a great application for composings written documentation, during daily communication, we constantly use natural language expressions to describe our thoughts. It would be naturally interesting and useful to explore the following two major tasks: translation between LaTeX and natural expression, and evaluate whether a given pair of LaTeX and natural expressions is equivalent to each other. To fulfill these two tasks, we utilized encoder-decoder networks with attention, which is one of state-of-the-art algorithms in natural language translation.

Currently, there had been a lot of study about converting images of mathematical expression into LaTeX. However, there is no study about convertion between LaTeX and natural expressions yet. Moreover, there is no available public dataset for similar tasks. Therefore, we also developed methods to generate LaTeX and natural expression pairs.

2   Dataset

Currently, there is no public data that can be used in the tasks we would like to explore. Thus, we implemented methods to generate pairs of equivalent LaTeX and natural expressions (in English) from pre-defined rules and elements.

By default, we treated all LaTeX expressions as under math mode. We first defined limited number of possible operations including plus, minus, power, fraction, integral, etc, as well as elements (Table 1).

Table 1: Elements used in generating expression pairs

Types	Elements in Corresponding Type
Digit (for LaTeX)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Number (for natural)	zero, one, two, three, four, five, six, seven, eight, nine
Variable	x, y, z

Table 2: Examples of rules for generating base and complex expression pairs. Numbers are placeholder representing elements to be filled. There can be several different natural expressions for the same LaTeX expression, which are separated by ';'.

Types	LaTeX rule	Natural rule
Base	0 \ge 1	0 is greater or equal to 1
Base	\frac{0}{1}	0 over 1; 0 divided by 1
Base	0\times 1	0 times 1; 1 multiplied by 0; product of 0 and 1
Complex	\int_{0}^{1} h(2) \,d3	integral of h (2) from 0 to 1 with respect to 3
Complex	\lim_{0 \to 1} h(2)	limit of h (2) as 0 goes to 1

29 We then defined three types of rules to write expression pairs. By permutating elements in defined
30 rules, we generated over 3000 pairs of equivalent expressions.

2.1 Define Expression Rules

32 There are three types of rules: base, complex, and nested. Some of the examples were listed in
33 Table 2.

34 Base rules were the most basic and simple rules. They consisted of basic operations: plus, minus,
35 multiplicatiom, fraction, power, and comparations. They had two or less parameters, and tended to
36 have the most simple grammar in both LaTeX and natural expressions. Note there can be several
37 different but equivalent natural expressions for one single LaTeX expression. For example, for
38 multiplication, people would say one times two, or two multiplied by one, or product of one and two.

39 Complex rules generally were relatively complicated. They often had three or more parameters, and
40 were with complex grammer. There were less number of complex rules than base rules.

41 Nested expressions were composed by nesting multiple base expressions or complex expressions.
42 For base and complex rule, they were filled by elements, while nested expressions were filled by base
43 or complex expressions. For example, we can have '\frac{x+1}{x} \ne 1' and 'x plus one over x is
44 not equal to one' as a nested pair. Nested data was not used during training. Rather, they were used
45 only during testing stage.

2.2 Data Generation

47 With defined rules, we generated expression pairs by permutating appropriate elements in correspond-
48 ing placeholder positions. Generally all possible cartesian products of elements would be used to
49 generate expression pairs. However, for positions of some rules, only one type of elemenets were
50 legit. For example, in interal expression, the boundaries can only be filled with elements in Digit type
51 (in LaTeX) or Number type (in natural), and the vairable can only be filled with elements in Variable
52 type (see Table 1 and 2).

53 In total, we generated 3391 data pairs which would be used during training, along with several nested
54 pairs. We shuffle all the expression pairs and split 80% of the pairs into training set, and rest into
55 testing set. Again, nested expressions were not used during training.

3 Experiment

3.1 Tokenization

58 Before we can feed data into network, we would need to define token for both LaTeX and natural
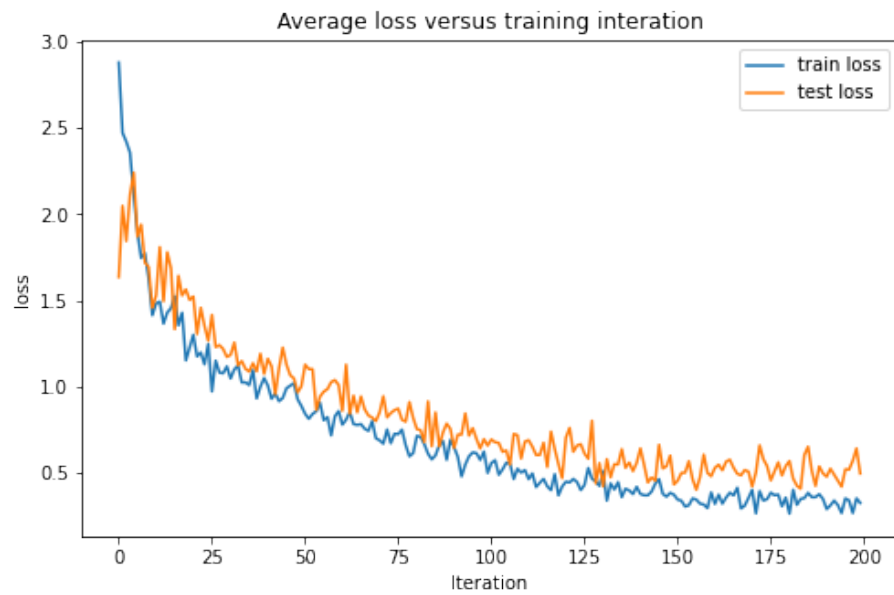59 languages.

3.2 Encoder-Decoder with Attention

4 Result

Figure 1: Training and testing loss curves versus training iteration

5 Discussion

References

[1] https://www.kaggle.com/austinreese/craigslist-carstrucks-data