Hunter TEXNet | CSCI 350 Project Proposal

Alex Taradachuck Ralph "Blake" Venté

March 2020

Abstract

The OpenAI request for research spawned research in using Visual Neural Models for translating mathematical expressions into markup code. Since the debut, progress has been largely incremental. Singh 2018, Wang 2019, and Bender 2019 all attribute properties of their respect models to the biases in the corpus and its pre-processing. We present a new dataset and a pipeline for harvesting examples from source files hoping to minimize these biases.

1 Previous Work

1.1 Debut

The prospect of accurately transcribing mathematical expression into a markup representation is enticing because it opens the doors for bringing new life to old mathematical texts or those for which the source code is unavailable.

A Harvard project called *What you get is what you see* (WYGIWYS) by Deng, Kanervisto, and Rush 2016 documents strategies for machine translation of mathematical notation using an attention-based encoder-decoder neural model. Notably, the researchers were interested in the influence of markup alone on the efficacy of a model — without providing explicit information about underlying grammars (Deng, Kanervisto, and Rush 2016, p. 1).

Introduced in 2016 by Deng, Kanervisto, and Rush 2016, the `im2latex100k` dataset has led research forward by removing the overhead of heavy pre-processing. Using this dataset, other solutions to IM2LATEX have seen substantial progress. Recently, it has come under question as a possible *bottleneck* for progress. We discuss this in greater detail in 2.2.

1.2 Architectural Designs

Since its inception, solutions to IM2LATEX have used neural models. In fact, OpenAI's Request for Research recommended the use of a neural model and even specified the attention machanism that makes these models so powerful, but there have been a wide variety in the particular architectures of the neural models.

Deng, Kanervisto, and Rush 2016 Genthial and Sauvestre 2016 Wang and Liu 2019 Bender et al. 2019

2 Corpus Harvesting

2.1 Challenges

We harvested examples from Cornell's arXiv archive. As the arXiv strictly forbids scraping, we found a patron who properly obtained copies and uploaded them to `archive.org`. We wrote a script to extract the source code files, and notably, we use the `pandoc` document parser.

On this note, LaTeX and its subset, TeX pose an unique set of challenges compared to other steps of the pre-processing pipeline. LaTeX is a Turing-Complete language, equivalent in power to general purpose programming languages. From the theoretical standpoint, it may contain loops and recursion, classifying it as a recursively enumerable language. As such, no LaTeX parser is guaranteed to terminate, and regular expressions alone lack the expressive power to extract all cases of a particular pattern. Practically, we mostly needed to contend with macro expansion. This means that the original scripts of Deng, Kanervisto, and Rush 2016 required substantial changes before being suitable for our purposes.

2.2 Innovations

To start, we follow closely in the footsteps of Singh 2018. After preprocessing steps on `im2latex100k`, Singh arrived upon a dataset of 93,784 compiling examples, which he regarded as "too small for good generalization." As such, he mined additional examples from the KDD Cup 2003 dataset, resulting in about additional 50,000 examples (Singh 2018, p. 8).

Practically, `pandoc` handled the files from the corpus gracefully, but due to unlinked files and mismatched encodings, we set `timeout 5` to terminate the program after 5 seconds of stalling. `pandoc` also facilitates extracting expressions. Without it, we were presented with the challenging prospect of filtering out text not containing mathematical expressions, but `pandoc` generates an abstract syntax tree and encases the mathematical expressions in the more "specific" expression `\((.*?)\)` compared to the expression `$(.*?)$` which resulted

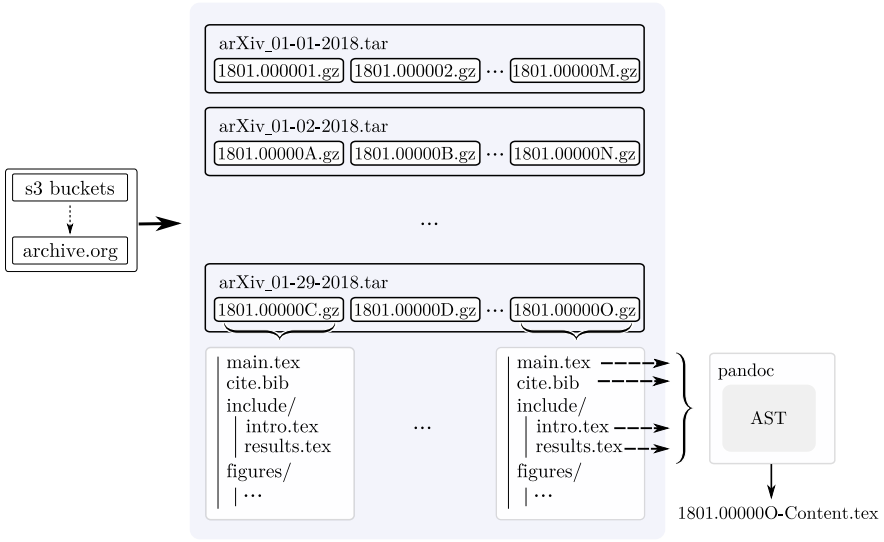

Figure 1: Examples flow from archive.org and are stored on our disks, where we extract the `tar` files first by day, then by submission, into the document folders. Then `pandoc` pulls in all of the dependencies of this particular document and writes out one stand-alone document with macros expanded. Then we match the regular expression `\((.*?)\)` that `pandoc` uses to wrap mathematical expressions. As a general tool `pandoc` can convert between other markup language in the same way, making it useful for standardizing data from disparate sources.

in false matches that contained only text. One such false match that was eliminated is provided in the appendix. As a fall-back, we rely on Singh's script to test for the presence of at least one LaTeX command (from the command vocabulary, Σ) before writing out the final formula code. This has the added benefit of removing examples that are strictly numbers and those that are otherwise linear strings of alphanumeric characters.

Most importantly, `pandoc` expands all the macros in the source documents. Without this step, papers in the dataset that relied heavily on macros to shorten the length of the written expressions contributed very few formulas to the resulting collection because they caused definition-not-found errors and were eliminatedin later steps. In Deng, Kanervisto, and Rush 2016 the processed formulas were discarded implicitly because they caused such errors, but with the macros expanding, documents reclaimed their representation in the final set, eliminating one possible source of bias in the dataset.

From Deng, Kanervisto, and Rush 2016, we also reiterate that several LaTeX input sequences map to the same output image, so we use the same normalization with Khan Academy's KaTeX as they did in their scripts. Like `pandoc` KaTeX parses the input sequences into an abstract syntax tree and rebuilds the expressions without disrupting semantics.

Bender et al. 2019, p. 5 notes that their IM2LATEX model struggled very short sequences, and performed better on those that were slightly longer. They attribute this to an imbalance in the dataset: very short formulas are underepresented. In fact, this may be traced back to the scripts written by Deng, Kanervisto, and Rush 2016, which discarded examples shorter than 40 bytes in length. We halved the minimum length of an example.

2.3 Product

In all, we mined 1 month of `tar` files, extracted 15,200 `tex` files, expanded macros 7,200 unique files and generated data 500,000 rendering examples. We decided to use about 40% of that for our final testing. [1] A graphical view of our data pipeline can be seen in Figure 1.

3 Network Architecture

Our network has its basis in Singh's research. The architecture has several essential features that we enumerate with breif explanations and proceed with detailed remarks on their implementation in our architecture.

Convolutional Neural Network Layers in our neural model learn convolutions on image space that extracts meaningful information into a compressed internal encoding vector.

Recurrent Neural Network At time t, the network's layer n takes as input the output of layer n at time $t-1$. To train such a network using backpropagation, it is "unrolled" so as to emulate a feed-forward network, allowing the updates to propagate.

Long-Short Term Memory Looseley speaking, the LSTM cells in the neural network "learn when to forget" about a particular subset of the previous layer's output. This allows for reasoning over time.

4 Evaluation

For evaluation, we use Bilingual Evaluation Understudy (BLEU), the emerging standard among neural network solutions to the IM2LATEX problem. BLEU is

[1] Data can be found at `https://github.com/rvente/TeXNet.ai/Dataset.`

an automated metric for assessing machine translation accuracy Papineni et al. 2002, p. 1.

References

[Ben+19] Sidney Bender et al. "Learning Fine-Grained Image Representations for Mathematical Expression Recognition". In: *2019 International Conference on Document Analysis and Recognition Workshops (ICDARW)*. Vol. 1. IEEE. 2019, pp. 56–61.

[DKR16] Yuntian Deng, Anssi Kanervisto, and Alexander M Rush. "What you get is what you see: A visual markup decompiler". In: *arXiv preprint arXiv:1609.04938* 10 (2016), pp. 32–37.

[GS16] Guillaume Genthial and Romain Sauvestre. "Image to Latex". In: (2016).

[Pap+02] Kishore Papineni et al. "BLEU: a method for automatic evaluation of machine translation". In: *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics. 2002, pp. 311–318.

[Sin18] Sumeet S Singh. "Teaching Machines to Code: Neural Markup Generation with Visual Attention". In: *arXiv preprint arXiv:1802.05415* (2018).

[WL19] Zelun Wang and Jyh-Charn Liu. "Translating Mathematical Formula Images to LaTeX Sequences Using Deep Neural Networks with Sequence-level Training". In: *arXiv preprint arXiv:1908.11415* (2019).

A Lessons Learned in Data Cleaning

```
1132      The net flow vector $(1,1,0,\hdots,0)$ has previously
              been considered for the complete graph by Corteel,
              Kim, and M\'esz\'aros \cite{CKM}.  They used the
              Lidskii formula~\eqref{eq:lidskiivol} and constant
              term identities to derive the following product
              formula for the volume of $\mathcal{F}_{K_{n
              +1}}(1,1,0,\ldot     s,0)$.
1133      It would be of interest to rederive this result using
              the refined Kostant constants.
1134     \begin{theorem}[{\cite[Theorem 1.1]{CKM}}] \label{thm:
             volkn11}
1135     Let $n\geq 1$. For the complete graph $K_{n+1}$,

   1    s,0)$. It would be of interest to rederive this result
            using the refined Kostant constants. \begin{theorem}[{\
            cite[Theorem 1.1]{CKM}}] \label{thm:volkn11} Let $
```

Figure 2: Math encased in dollar-signs has been deprecated in LaTeX, but remains in wide use as an artifact from TeX. It was deprecated to facilitate parsing: unlike `\(` which indicates that it opens to the right, the dollar-sign has no such indication. The reason the previous scripts were matching text was because the hanging `$` of the first line matched with the very first dollar sign of the 4th line. Viewed another way, the regex was using the same token twice, which initially caused a false match marked as (1).