Hunter TEXNet | CSCI 350 Final Project

Alex Taradachuk   Ralph “Blake” Venté

March 2020

Abstract

The OpenAI request for research spawned investigation into 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 of 170,000 rendered examples and 500,000 normalized formulas along with a pipeline for harvesting examples from source files hoping to minimize these biases. We train a Convolutional Neural Network with Long-Short Term Memory Cells and an attention mechanism to a testing accuracy of 0.87 bigram BLEU score.

Contents

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.

deng2016you by **deng2016you** 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 [**deng2016you**]. The previous state of the art required exhaustively documenting the grammar to allow a conventional character recognition model to translate the expression.

Introduced in 2016 by **deng2016you**, the `im2latex100k` dataset has facilitated research by reducing the overhead of heavy pre-processing. Using this dataset, other solutions to IM2LATEX have seen substantial progress. Recently, it has come under scrutiny as a possible *bottleneck* for progress. We discuss this in greater detail in Section 2.2.

1.2 `im2latex` Problem Specification

Before delving into how architectures were designed to solve the problem, let us first define it. Formally, this problem is a sequence-to-sequence image captioning task. A neural model learns an abstract encoding and its respective decoding into the original sequence [**deng2016you**]. At inference time, when presented a single image, the trained model predicts one token at a time, aiming to reconstruct the original sequence. All the while, the model receives as input the output of previous steps. This is accomplished using Long-Short Term Memory, which we outline in 1.3.

1.3 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 sequence-to-sequence attention machanism that makes these models suitable for the problem, but there have been a wide variety in the particular architectures of the neural models.

deng2016you genthial2016image wang2019translating bender2019learning

2 Corpus Harvesting

2.1 Challenges

We harvested examples from Cornell's arXiv journal. As the arXiv strictly forbids scraping, we discovered that a generous archivist properly obtained copies (paying for transfer from Amazon s3 buckets) 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 whereby commands used as contractions needed expanding into their definition. This means that the original scripts of **deng2016you** required substantial changes before being suitable for our purposes.

2.2 Innovations

To start, we follow closely in the footsteps of **singh2018teaching**. 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 [**singh2018teaching**].

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 in false matches that contained only text. One such false match that was eliminated is provided in Figure 4. 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 **deng2016you** the processed formulas were discarded implicitly because they caused such errors, but with the macros expanding, documents containing macros reclaimed their representation in the final set, eliminating one possible source of bias in the dataset.

From **deng2016you**, 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.

bender2019learning note 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 **deng2016you**, 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 syntactically valid examples. We decided to randomly subsample about 40% of that as our final candidate set. After deduplicating, we were left with 170,00 original examples. Our formula files can be found at `https://github.com/rvente/TeXNet.ai/Dataset.` A graphical view of our data pipeline can be seen in Figure 1.

3 Network Architecture

Like much of our own, our network is based on **singh2018teaching**'s research. The general architecture has several essential features that we enumerate with brief explanations then proceed with detailed remarks on their relevance to our architecture.

Convolutional Neural Network (CNN) Layers in our neural model learn convolutions on pixel space that extracts meaningful information into a compressed internal embedding, that is, a vector which has a dense encoding of the contents of the input image.

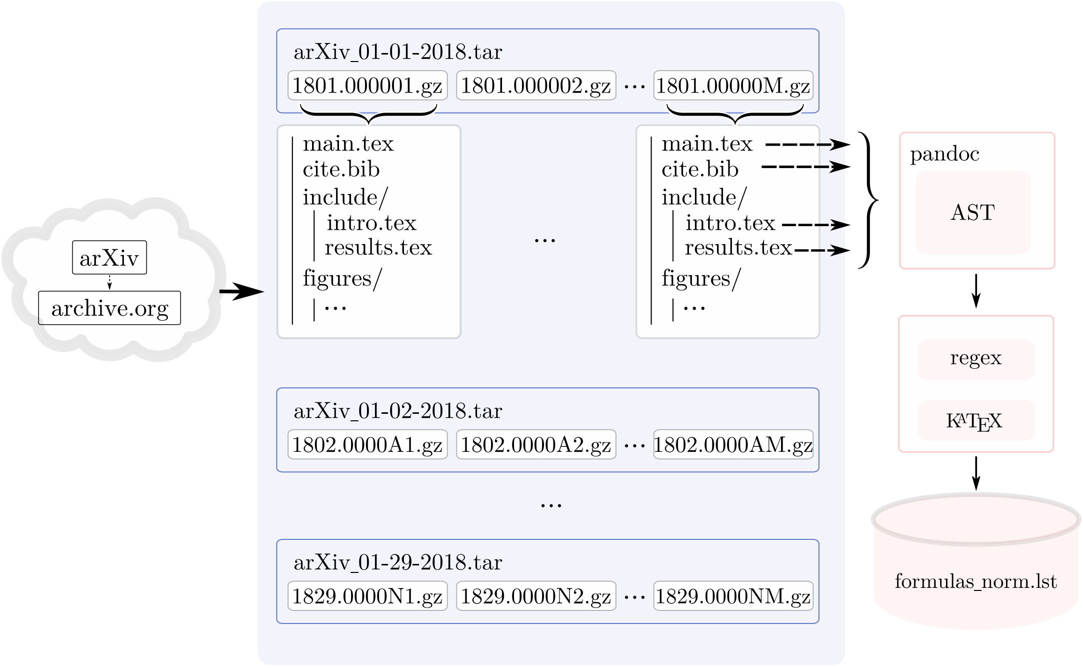


Figure 1: Examples flow from the Internet Archive and are stored on our disks, where we extract the `tar` files first by day, then by submission, into the document folders. Then `pandoc` pulls in all of the dependencies of this particular document and writes out one stand-alone document with macros expanded. 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. Special thanks to Brian Newbold `http://bnewbold.net/`, an internet archivist, working at the Internet Archive who provided the raw form of the data we pre-processed.

Recurrent Neural Network (RNN) This allows for reasoning over time because information from prior time-steps can factor into current time steps. It accomplishes this by incorporating as input the output of previous layers.

Long-Short Term Memory (LSTM) Looseley speaking, the LSTM cells in the neural network "learn what to forget and when to forget it." This aspect of the architecture allows a particular subset of the previous layer's output to be factored in dynamically based on what the LSTM cells have learned is important.

Attention machanism Attention mechanisms allow for the network to mask way, or "ignore" portions of the input image. The network can now use information about past predictions to "keep track of where it's looking," allowing for focused steps in the prediction process.

3.1 Convolutional Neural Network

3.2 Recurrent Neural Networks

At time t, the an RNN'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. The issue with an RNN alone is that memory of previous inputs quickly decay when moving from time-step to time-step.

3.3 Long-Short Term Memory Configuration

To solve these issues, researchers use need a network to selectively add and remove inputs into the current state [**zhang2019dive**]. Dating back to 1997, this LSTM architecture solves this using gates to control the flow of data during training and inference [**zhang2019dive**]. It takes 3 inputs, c_{t-1} memory at time $t-1$,

4 Training and Evaluation

We trained the neural network on a Virtual Machine (VM) with an NVIDIA Tesla V100. Later, that platform became unavailable, so we migrated training to a VM with an NVIDIA Tesla P1 instead. Unfortunately, this change means we cannot make conclusions about the seconds to convergence between datasets of different size.

For evaluation, we use Bilingual Evaluation Understudy (Bleu), the emerging standard among neural network solutions to the im2latex problem. Bleu is an automated metric for assessing machine translation accuracy **papineni2002bleu**. It is computed by comparing a target translation (ground truth) to the predicted sequence.

We report 3 metrics: first we compute global Bleu score across all examples and then again with 2 splits. The first split evaluates performance by length in tokens of the target sequence and the next split evaluates performance based on the contents of the expression.

We selected 3 "categories," chosen based on the qualities of the expression. For Group A, we select expressions containing matrices, arrays, and peice-wise functions (cases). For Group B, we select those containing fractions. We leave the rest of the data in Group C. We note that the sets not disjoint.

From **papineni2002bleu**, Bleu score has a penalty for translations that are too long and those which are too short. The former case is handled implicitly, and the latter requires an explicit brevity penalty, BP computed as

$$\mathrm{BP} = \begin{cases} 1 & \text{if } c > r \\ e^{1-r/c} & \text{otherwise} \end{cases} . \tag{1}$$

with r as the length of of the ground-truth sentence and c as the length of the candidate sentence. Then, we have

$$\textsc{Bleu} = \mathrm{BP} \exp\left(\sum_{i=1}^{n} w_i \log p_i \right) , \tag{2}$$

where p_n is geometric mean of n-gram precisions from 1 to N and w_n is a parameter weight, such that $\sum_i w_i = 1$. **papineni2002bleu** used $w_n = 1/N$. **papineni2002bleu** show that Bleu score is predictive of human judgements, resulting in its widespread use in translation tasks.

We use the same reformulation of this metric as [**singh2018teaching**] and [**deng2016you**], the bigram Bleu (Bleu2) score computed with $w_1 = w_2 = .5$ and $w_i = 0$ for $i > 2$. Also like Singh, the scores we report actually compare non-normalized target sequences to the normalized predicted sequences, resulting in a pessimistic predictive score because the network was not trained on this sequence. Despite this, the Bleu2 scores are quite satisfactory.

4.1 Limitations and Future work

Through the testing of our current model, we have observed notable performance decrease with increasing sequence length. This is most likely due to the low proportion of training data sequences over 140 symbols however it could also be a result of a loss in resolution as sequence length increases. A next step forward could include partitioning the data if sequence length is long enough. Additionally, the preprocessing scripts utilized by Singh [**singh2018teaching**] leave a white outline around the images generated from LaTeX which the network is hardwired to expect. As a result, in order for the model to work on pulled examples from the internet, we had to preprocess the images in the following manner using imagemagick:

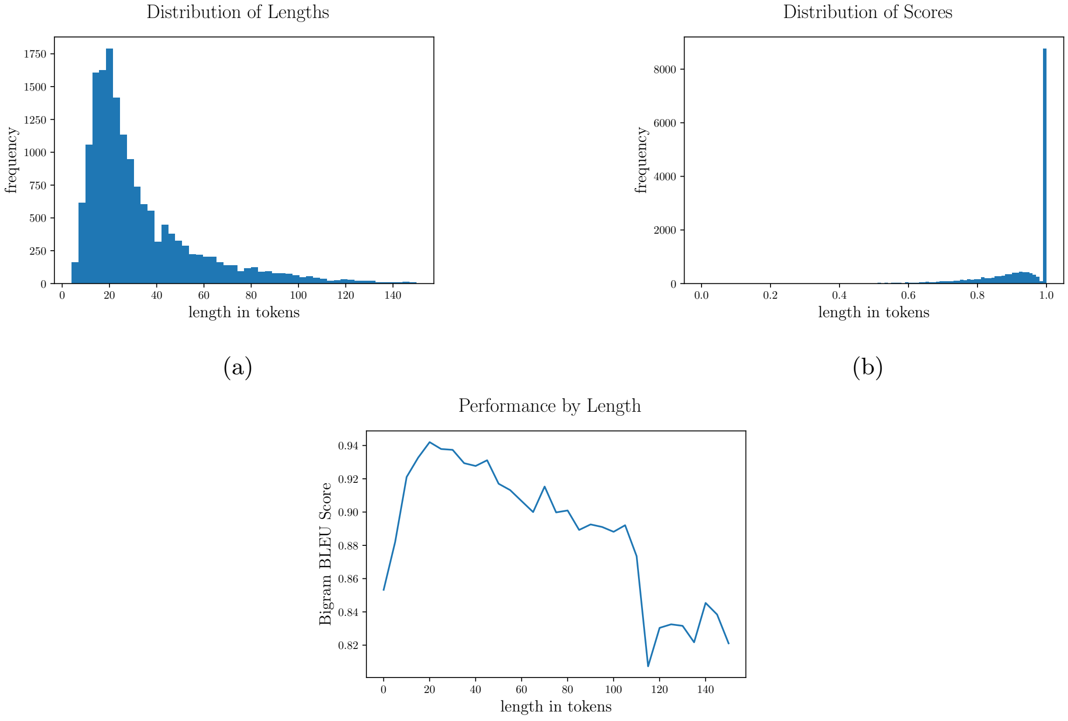
1. Resize image to fit under the 128 pixel token height limit
   - convert *filename* -resize x90 *filename*
2. Make the white background transparent with a fuzz of 5% to account for the feather expected by the model
   - convert *filename* -fuzz 5% -transparent white *filename*

The use of our own data, scraped from real world examples, lended our model to be more accurate when it came to inferencing downloaded images. However, there was a disparity in training time between Singh's model and ours, despite having similair BLEU scores, which could've impacted results. As a proof of concept however, building im2latex models with deeper layers and increasing data size is promising given our results and the general trend NPL models are currently taking.

A Qualitative Analysis

A.1 Reducing Plain Text Matches

Figure 4 is a sample from a particular document whose source code induced a false match. Math code 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 the expression opens to the right, the dollar-sign gives no such indication. The reason the previous scripts were matching these plain-text strings was because matching all strings that start and end in dollar-signs doesn't preclude text from matching. In this case, the last `$` of the first line matched with the first `$` of the 4th line.

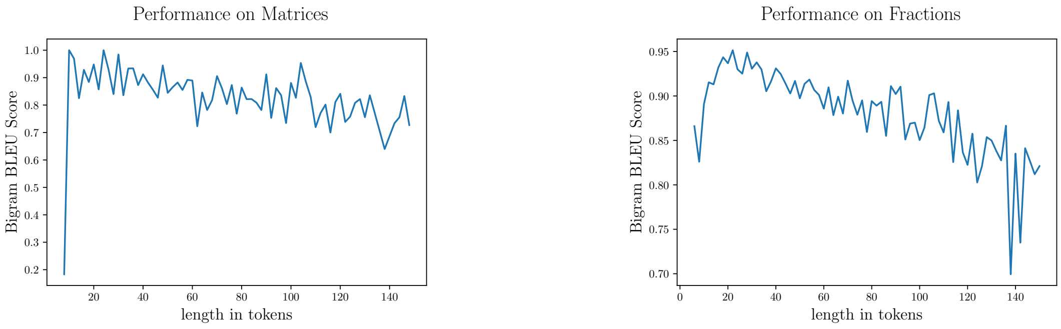


(a)

(b)

(c)

Figure 2: This assessment is on the validation set $N = 1,652$. (a) Plotted with 50 bins. The mean length is with a mean of 32.56 and standard deviation of 23.82. (b) Plotted with 100 bins. Bleu Scores are strongly concentrated on 1.0, with $\mu = 92.5$ and $\sigma = 23.81$. It seems that the discontinuous indentation before 1.0 is an artifact of how Bleu score penalizes non-perfect matches. At 32 tokens. There, a single incorrect token causes bin. (c) As expected, performance is inversely proportional to length, however on the longest tokens, the model still perfomed at a respectable .82 Bleu score. The steep drop-off after 100 tokens is caused in part by there being fewer examples contributing to the average on that bin, so a few uncharacteristically bad predictions have a large influence.



(a)

(b)

Figure 3: (a) We generalize cases (peicewise functions), arrays, matrices, column vectors here and evaluate performance on that set finding $\mu = 0.85, \sigma = 0.14$. Unsurprisingly, matrices were more challenging for the model than general math, but matrices are underepresented at $N = 291$ (b) We match `frac` and `over` for fractions finding $\mu = 0.91$ and $\sigma = 0.11$ Samples $N = 2897$.

The source code

```
 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 $\mathcal{F}_{K_{n
    ↪ +1}}(1,1,0,\ldot s,0)$.
 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 $n\geq 1$. For the complete graph $K_{n+1}$,
```

matched the regular expression `$(*.?)$` with the string

```
$. 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 4: From `arXiv:1801.07684` **benedetti2018combinatorial**