Turning equations into \LaTeX**

### Pipeline vs. End-to-End

**Motivation**

There is currently no good solution for converting handwritten notes into \LaTeX. As a consequence, STEM students around the world struggle. In this project we convert images of equations in \LaTeX\fig{latex}

1. A multi-stage pipeline approach where two main steps are:
   - Segmenting the image into individual symbols
   - Classifying individual symbols
   - Resegement based on classification (e.g. two minus signs on top of each other equal sign)
   - Structural analysis (how do symbols relate to each other?)
   - Parse to \LaTeX

   Between 80k and 1.5 million parameters depending on the specific implementation.

**Data and Features**

1. InftyCDB-3 = Mathematical Symbols ~ 70k
   - High resolution images of math symbols
   - Ground truth math symbols, 275 tokens
2. We downsample, normalize and center crop.
3. Split into Train / Val / Test

   **484 different tokens**
   - Token lengths up to 140.
   - We limit to 70, which gets us 50k data.

   **Im-to-latex-100k** – Equations – 100k

   **Gray scale**

   **InftyMDB-1** – Equations – 5k

   We downsample, normalize and center crop.

   Downsampled images of math equations sampled from arXiv \cite{3}

   **Consequence, STEM students around the world struggle.**

   **In the future**

   **Large quantities** for the end-to-end model. The need for data augmentation will likely persist in the pipeline model when switching to handwriting.

   **Training the end-to-end model was very difficult** partly because it was so slow. We did better with the hand-engineered rules that the pipeline model uses. We did better with the end-to-end model.

   **Discussion**

   Training the end-to-end model was very difficult, partly because it was so slow and partly because attention models are in general difficult to train. The multi-stage pipeline approach has the advantage of not having to rely on the many hand-engineered rules that the pipeline model uses. We did better with the pipeline approach which offers many advantages: 1) it is faster to train and iterate; 2) easier switch to handwriting since there exists enough data on individual symbols (the same is true of full equations which is needed in large quantities for the end-to-end model). The need for data augmentation will likely persist in the pipeline model when switching to handwriting.

   **Putting it all together with structural analysis**

   We used a simple structural analysis which read the characters from left to right, top to bottom. It introduced a superscript if satisfying a few criteria the most important one being if the center is up to the right of the previous character (whilst checking the characters in the figure below). Therefore a resegmentation, based on hand engineered rules, after the individual symbols have been classified is needed.

   **5 Stage Pipeline**

   **Model overview**

   1. Segment into individual symbols
   2. Classify individual symbols
   3. Resegment based on classification (e.g. two minus signs on top of each other \rightarrow equal sign)
   4. Structural analysis (how do symbols relate to each other?)
   5. Parse to \LaTeX

   **Cross-Entropy Loss (used for both approaches, for each token prediction)**

   \[
   H(p_i, q_i) = - \sum_i p_i \log q_i
   \]

   **Data Augmentation**

   When first trying out the OCR on the segmented images in the equations it performed poorly. There is a mismatch between the images generated by the segmentation algorithm and the images of individual symbols that we are training on. We augmenting the data by zooming in, and shifting the images in horizontal and vertical directions. We then retrained on a larger set of 140k images. This made all the difference. The algorithm classifies symbols in the InftyMDB-1 equations well.

   **Putting it all together with structural analysis**

   We used a simple structural analysis which read the characters from left to right, top to bottom. It introduced a superscript if satisfying a few criteria the most important one being if the center is up to the right of the previous character (whilst checking to see if the previous character was a subscript in which case add additional criteria is needed). Analogous rule for subscript. The structural analysis is the bottleneck of the algorithm as of right now as it can not handle equations where a symbol have both a superscript and subscript. For all other equations the algorithm seems to perform well (since it classifies almost all of the tokens correct when we manually check performance on a subset of 30 examples).

   \[
   \{ \alpha, \beta \} = \{ \gamma, \delta \} \\
   \text{LeftBrace} \alpha \text{LeftBrace} \beta \text{RightBrace} \gamma \text{RightBrace} \\
   \text{LeftBracket} a \infty \text{LeftBracket} \infty \text{RightBracket} \subset \text{ subsets } S
   \]

   **Results**

   **Example predictions**

   **Input image:**

   **Predicted Latex (correct):**

   \[
   F (\mu | \nu) = (| \partial | \mu | \nu) A \in (| \nu) - (| \nu) \partial (| \nu) \in (| \nu)
   \]

   **Input image:**

   **Predicted Latex (mostly wrong):**

   \[
   \text{Gamma} (s+1) = (s+1) ! (s+1) !
   \]

Adan Jensen (ojensen@stanford.edu)
Henrik Marklund (marklund@stanford.edu)