Turning Equations into Latex: Pipeline vs. End-to-End

Adam Jensen (oojensen) & Henrik Marklund (marklund)

Acknowledgments

We are very grateful to: Guillaume Genthail for his time and for providing the im-to-latex-100k dataset; Ramtin Keramati for support and feedback; and Joseph Chang for valuable suggestions. We are also grateful to our grader for putting up with our half inch margins.

Introduction

Image-to-LaTeX conversion is an important but challenging task. Learning a function from rendered LaTeX back to the original markup would enable students to more quickly complete their homework. The function from rendered LaTeX to the markup is challenging to learn because the vocabulary of valid LaTeX tokens is large (about 500 symbols) and the available data sets are not huge (in the low 100,000s of labelled examples). A clear winner in this domain has not yet emerged. As a consequence, academics around the world struggle.

Related Work

Structured text OCR or markup decompilation can be thought of as a subset of the image captioning problem. Image captioning has received a lot of attention in recent years. The go-to deep learning solution that has emerged from work done by Karpathy et al (2015) and many others for end-to-end image captioning is a convolutional encoder with an LSTM decoder. To the best of our knowledge, the current state-of-the art on LaTeX decompilation is held by Deng et al (2017) using a similar encoder-decoder architecture. They achieved exact match results of over 77%. We were not able to exceed these results, but we did achieve similar accuracy to Genthial et al who worked on this problem in CS231N. Previous CS229 students Chang et al. have also explored the problem using a more old-school pipeline system (2016). Our main contribution is 1) replicating the results Genthial & Sauvestre (2016) and 2) comparing our implementation of an end-to-end deep learning approach to our implementation of a pipeline-based approach.

Convolutional neural networks were pioneered by LeCunn (1998) and LSTMs were proposed by Hochreiter (1997). Our work also uses the attention mechanism proposed by Xu et al (2015). We provide more information on how these models work in their respective sections.

Methods

The 5 Stage Pipeline Approach

Our first method is an adaption of the multi-stage approach proposed by Chang et. al (2016). We divide the model into five stages: segment into individual symbols, classify individual symbols, resegment based on classification (e.g two minus signs on top of each other is an equal sign), perform “structural analysis” (e.g. answer: how do symbols relate to each other?), and then parse the relationships to Latex. In this paper we focus on the first two stages.

Data Set

For the pipeline approach, we used the InftyCDB-3 and datasets of Mathematical Symbols.[1] The dataset includes high resolution black and white images of 275 different math symbols. Each training example is a high resolution (sizes vary widely) black and white image of 1 rendered symbol, paired with a code for that symbol (not LaTeX). For InftyCDB3 there are 70637 total examples and we split them 90:10:10 (56509 / 7063 / 7065) into Train, Dev and Test sets. We also used 30 examples from InftyMDB-1 to manually evaluate our pipeline model.

Data preprocessing

For our fully connected neural net we need the images to be the same size and also small so as to speed up training. Consequently, we do three preprocessing steps: 1) Pad them all to the largest size 120 x 188; 2) downsample all images to a size of 72 x 113 ; 3) center crop the image to a size of 30 x 35. We further normalize and invert the images. We perform the exact same three steps on the segmented individual symbols with the goal of matching the images we trained on as closely as possible.

[1] http://www.inftyproject.org/en/database.html

Data augmentation

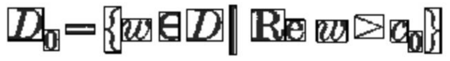
When first trying out the OCR model on the segmented symbols in the equations it performed poorly. It became clear that there is a mismatch between the images generated by the segmentation algorithm and the images of individual symbols that we are training our OCR model on. We augmented the data by zooming in, and shifting the images in horizontal and vertical directions. We then retrained our model on a larger set of now 156k images (instead of 56k). This made all the difference. By manual inspection we can now conclude that the algorithm classifies symbols in the InftyMDB-1 equations well.

Segmentation

We first use Recursive Projection Profile Cutting (RPPC), as suggested by Chang et. al (2016), to segment the images into individual symbols. Using RPPC means we project all points down to a horizontal or vertical line and cut where there still are white spots, recursively.

Since we cut vertically and horizontally this method fails on some symbols (eg. square roots and italics, or disjoint symbols like ≥ and = and i). If a character has two disjoint parts this character will be split into two. We follow Chang and deal with this using hand-engineered rules. Example of a segmentation:

$$D_0 = \{w \in D \mid \mathrm{Re}\, w > c_0\} \longrightarrow$$



Results on classifying individual symbols (OCR)

Our best model is a fully connected layer connected a softmax layer. We train on the InftyCDB-3 dataset. Our loss is a categorical cross-entropy loss. These OCR results are marginally better than Chang's best model (.9888%). We attribute our success to making the function easier to learn by down-sampling. Chang et al. did not downsample.
The formula for Z in the fully connected model's forward propagation is:

$$Z^{[1]} = \begin{bmatrix} | & | & | \\ z^{[1](1)} & z^{[1](2)} & z^{[1](3)} \\ | & | & | \end{bmatrix} = W^{[1]}X + b^{[1]}$$

...and the equation for the softmax output is: $\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$ for $j = 1, \ldots, K$.

Model	Train acc	Val acc	Train loss	Val loss
FC + Softmax	***0.996***	***0.990***	***0.019***	***0.079***
Conv1	*0.993*	*0.986*	*0.046*	*0.112*
Conv2	*0.996*	*0.988*	*0.022*	*0.089*

Table 1. Results for OCR.

Results of Using Rule-Based Structural Analysis to generate LaTeX

In order to translate our segmented symbols into LaTeX, we implemented a rule-based structural analysis system. We read the characters from left to right, top to bottom, and generate latex for each symbol. The structural analysis is currently the bottleneck of our multi-stage algorithm. It does not handle equations where a symbol have both a superscript and subscript. For other equations the algorithm performs quite well. Of the correctly segmented symbols, the OCR model classifies 93% correctly when we manually check performance on a subset of 30 example equations (approx. 400 individual symbols). Sample input and output are shown in Figure XXX.

$$D_0 = \{w \in D \mid \mathrm{Re}\, w > c_0\} \longrightarrow$$

```
D_{ 0 } = \LeftBrace w \in D \LeftBracket R e w > c_{ 0 } \RightBrace
```

Figure 1. Example prediction from pipeline system

The End-to-End Approach

Now we turn to a fully end-to-end approach where we input the image of the equation and let the neural network output the latex. Here the network needs to learn to perform segmentation, structural analysis and latex grammar by itself without any hand engineering. This is a challenging task which requires a lot more data than the multi-stage pipeline approach. However, the payoff of getting it to work is great results, and a much more general model that could also be used to decompile other kinds of markup.

Dataset

Deng et al (2016) at Harvard crawled arXiv for mathematical equations and gathered a 100k equations. This is known as the im2latex-100k dataset. Each training example is: pixels of a rendered latex equation, matched with ground-truth markup LaTeX. The images range from 40 x 160 pixels up to much larger sizes. We filtered out all equations with more than 70 tokens due to memory constraints of our GPU. After filtering, we had 54,766 training images, 6,032 dev and 6,335 test. The vocabulary is very different from the other dataset, and there are 485 symbols in the vocabulary.

Input image: $a(\xi) = \frac{\xi^{-1}-\xi}{q^{-1}\xi^{-1}-q\xi}, \quad b(\xi) = \frac{q^{-1}-q}{q^{-1}\xi^{-1}-q\xi}.$

Ground truth: *a (\xi) = \frac { \xi ^ { - 1 } - \xi } { q ^ { - 1 } \xi ^ { - 1 } - q \xi } \, , \quad b (\xi) = \frac { q ^ { - 1 } - q } { q ^ { - 1 } \xi ^ { - 1 } - q \xi } \,*

Model

We start out with the neural translation model described in Luong et al (2017) and switch the LSTM encoder to a convolutional OCR network that takes images as input[2]. The LSTM takes the encoding as input and outputs 1 token per timestep using a softmax layer.

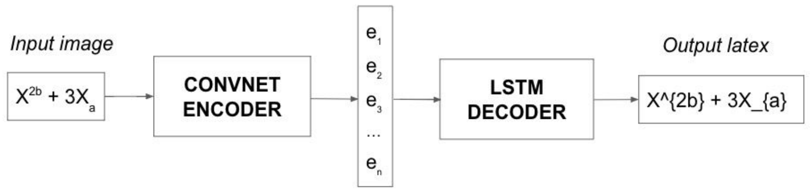


Figure 2. Encoder-decoder model.

Our cost function is cross entropy loss summed over all the tokens in a decompiled markup sequence. For one token, the loss is:

$$H(y, \hat{y}) = \sum_i y_i \log \frac{1}{\hat{y}_i} = -\sum_i y_i \log \hat{y}_i$$

Encoder: Convolutional Neural Net

Our encoder is a convolutional neural net that outputs an encoding of the image of a smaller size h*w*512 (see Genthial & Sauvestre, 2016). This encoding is unrolled to a series of vectors ($e_1 \ldots e_n$) corresponding to different parts of the image.

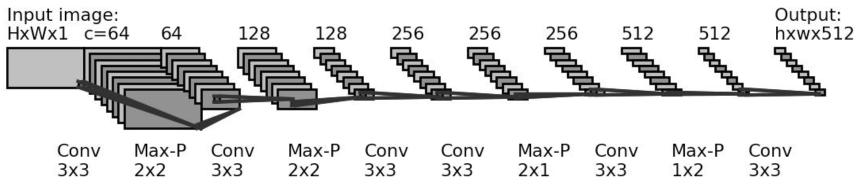


Figure 3.One of three encoder architectures we tried. Convnet1.

Decoder: LSTM with and without attention.

We try two different decoders. One, without an attention mechanism, uses the average of the encoding vectors to generate the initial hidden state of the LSTM as described by Genthial & Sauvestre.

[2] We also create one model starting from the simpler translation model described in Chollet. (2017). This model does not use an attention mechanism.

Our attention setup is the same as described in Luong et al (2017) with the initial LSTM state being zero vectors (see figure 4). In the Attention step an importance score is calculated for each encoded vector e based on the current hidden state h_t. We use scaled Luong scoring as described in Luong et al (2017). These scores are used to generate a weighted average of the encoded image. This weighted average $\bar{e}_t$ is called the context vector and is concatenated with h_t in order to generate the attention vector a_t. The attention vector is fed back into the LSTM as well as used to generate the next token via softmax layer. This attention wrapper lets the decoder attend to different parts of the image at each time step.

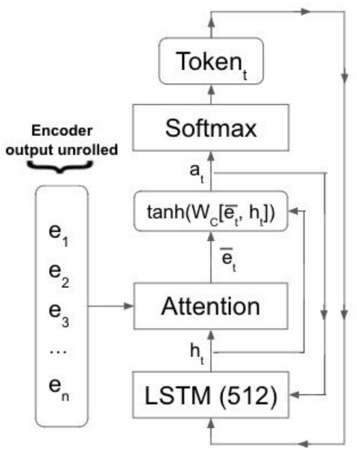


Figure 4. Decoder with attention.

Training

We apply forced teaching: we feed in the correct token to the LSTM at each time step. We implemented a learning schedule very similar to Genthial's. It utilizes a warm-up period of three epochs, then after 6 epochs, the rate starts decaying exponentially for 6 epochs. Our winning model trained for 12 epochs on a GPU which took 18 hours. We used Adam mini-batch optimization, a mini-batch size of 16, and clipped gradients at a max norm of 3.

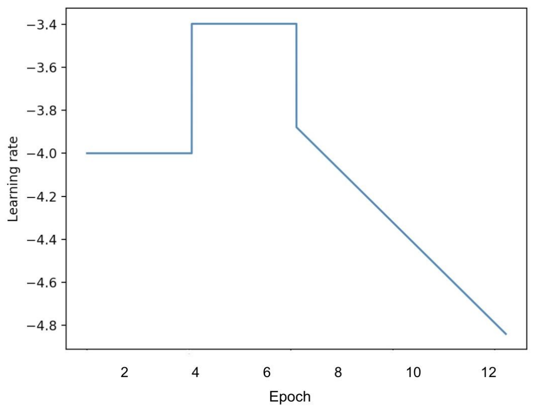


Figure 5. Learning rate schedule in log10 scale.

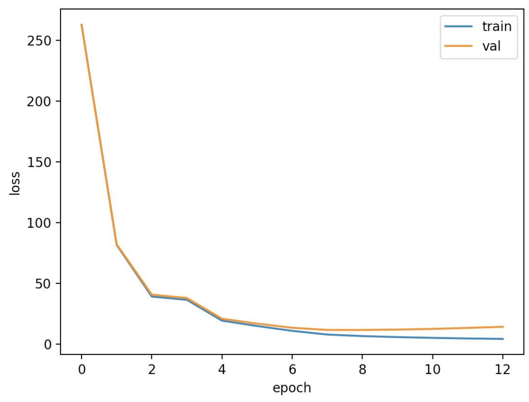


Figure 6. Train and validation loss over the epochs for our winning model.

Results

In table 2, we show the results on the validation set of a s subset of experiments performed. The attention model is our best model. Evaluating this model on the test set gives: EM: 17.2, Token accuracy: 48.4, Edit distance. 2.9.

Hyperparameters			**Results**[3]		
Encoder	**Decoder**	**ds**	**EM**	**Tok. acc**	**Edit dist.**
Convnet1	*No atten.*	0.6	0%	21.06	6.5
Convnet1	*No atten.*	0.9	0%	20.19	7.5
Convnet2	*No atten.*	0.6	0%	18.7 %	6.8
Convnet2	***Attention***	***None***	**17.0 %**	**47.7**	**3.3**
Benchmark: Genthial (2016)			22%	N/A	N/A

ds: downsample factor.
EM: exact match.
Tok. acc: Token by token accuracy.
Edit dist: Levenshtein distance per 10 tokens: for 10 tokens in a predicted sequence, how many edits need to be made to reach ground truth on average.
Convnet2: Same as Convnet1 with the first two Max Pool layers removed.

Table 2. Results for End-to-End model.

[3] Results are reported differently from those in the poster that goes with this paper. Also, we reran some experiments as well as introduced Attention which bumped up the results..

Example Input Image: $\chi_h^{NS}(q) = \sum S_h^{h'} \chi_{h'}^{NS}(\tilde{q})$

Pred: `\chi _ { h } ^ { N S } ( q ) = \sum _ { N S } { S } _ { h } ^ { h ^ { \prime } } \chi _ { h } ^ { N S } ( q )`

Truth: `\chi _ { h } ^ { N S } ( q ) = \sum S _ { h } ^ { h ^ { \prime } } \chi _ { h ^ { \prime } } ^ { N S } ( \tilde { q } )`

Qualitatively comparing the two approaches

In table 3, we present a qualitative comparison of the two approaches. The multi-stage model we implemented is representative of a classical machine-learning approach, with many steps and hand-tuned rules. The end-to-end approach is an example of the trendier, more computationally expensive approach recently enabled by the power of GPUs.

	Multi-stage pipeline	**End-to-End**
Model complexity	**Low** Between 80k - 1.5M parameters.	High Between 6.5 - 11M parameters.
Training	**Easy** Training is a fast due low model complexity which makes iterating on the model easier.	Very Difficult Training is slow and difficult due to a large number of parameters as well as the inherent difficulty of training models with an attention mechanism.
Learning grammar and structure	Difficult Depends on many hand engineered rules and sophisticated structural analysis.	**Easy** The end-to-end model (even without attention) quickly picked up Latex grammar and high level structures like integrals and fractions. The same model could be used to decompile HTML or any other visual markup language.
Applicability to handwriting	**Moderate** Getting a pipeline-based approach to handwriting is likely to be much easier since there exists enough data on individual symbols.[4]	Difficult (due to lack of data) Large datasets mapping handwritten equations to ground truth LaTeX do not exist.[5] Synthetic data generation is one option tried in Deng et al (2017).

Table 3. A comparison between the two approaches.

As you can see from the table, neither approach wins in every category. However, with enough compute, and the right data set we expect the deep-learning system to have superior asymptotic performance.

Future work

In future work getting the multi-stage pipeline model to work on the im-to-latex-100k would let us quantitatively compare the two approaches instead of only comparing them qualitatively. We were not able to provide a quantitative comparison because the vocabularies in the two datasets are very different. So, in order to provide a fair comparison for the two models, we would need to create a symbol-level dataset with the im-to-latex100k characters. For the pipeline model, the next step is to implement a sophisticated structural analysis algorithm like the one proposed in Eto & Suzuki (2001).

The main next step for the end-to-end referenced here is to implement beam search. Genthail et al. saw a significant performance improvement from implementing beam search (from 22 to 32% exact string match). Our learning curves show a slight divergence between our test and val set beginning at epoch 6 so we may be overfitting marginally. In the future it would be good to perform experiments with regularization. We would also like to do more hyperparameter search. We are also curious about trying a purely convolutional image captioning approach as in Aneja et al (2017).

[4] The CROHME dataset as well as the Detexify are two examples of large datasets of individual symbols. See http://www.isical.ac.in/~crohme/ and https://github.com/kirel/detexify-data

[5] CROHME dataset have only about 12k handwritten equations which is not even close to enough.

References

Aneja, J., Deshpande, A., & Schwing, A. (2017). Convolutional Image Captioning. *arXiv preprint arXiv:1711.09151*.

Chang, J., Gupta, S., & Zhang, A. (2016). Painfree LaTeX with Optical Character Recognition and Machine Learning.

Chollet, F. (2017, September 29). A ten-minute introduction to sequence-to-sequence learning in Keras [Web log post]. Retrieved December 14, 2017, from https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html

Deng, Y., Kanervisto, A., Ling, J., & Rush, A. M. (2017, July). Image-to-Markup Generation with Coarse-to-Fine Attention. In International Conference on Machine Learning (pp. 980-989).

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

Genthial, G., & Sauvestre, R. (2016) Image to Latex.

A. Karpathy and L. Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 3128–3137, June 2015.
Genthial, G., & Sauvestre, R. (2016) Image to Latex.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied t
o document recognition.Proceedings of the IEEE, november 1998

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

Eto, Y., & Suzuki, M. (2001). Mathematical formula recognition using virtual link network. In *Document Analysis and Recognition, 2001. Proceedings. Sixth International Conference on* (pp. 762-767). IEEE.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., ... & Bengio, Y. (2015, June). Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning* (pp. 2048-2057).

Contributions

Both team members worked a lot on this project. Neither could have accomplished it without the other. They collaborated on implementations of the end-to-end approach in both Keras and Tensorflow. Henrik Marklund did all the work relating to the pipeline approach. Adam Jensen did a lot of work on a different project that they ended up abandoning.