Handwriting to LaTeX Markup via Neural Network

Zachary Hervieux-Moore
ORF 525
Final Project
zhm@princeton.edu

Abstract

Neural networks are well known for their image classification abilities due to the strong ability of convolutional neural networks (CNN) to extract features from an image. More recently, recurrent neural networks (RNN) are being used to achieve cutting edge results in language translation thanks to modern architectures such as sequence-to-sequence models. In this paper, we present a combination of these two networks that converts handwritten mathematical equations to LaTeX markup. While this is not a novel problem, the optical character recognition (OCR) field has been doing this for years, it is a new approach. It also has benefits over conventional OCR techniques as no assumptions are needed about the structure of the underlying markup language. Thus, this technique is easily able to handle new mathematical symbols without the need to change the algorithm. Empirical results show that the system is sub par of with domain-specific LaTeX OCR systems with a 25% character accuracy with a limited training set.

1 Introduction

The problem of turning handwritten equations into mathematical expressions has a rich history in the early 2000's. Much of the work done used techniques used in OCR such as principal component analysis [1]. However, most of these techniques require additional work to be done to handle the grammar of mathematical equations and other calligraphic problems such as demarcating superscripts, fractions, integral bounds, matrices, etc. In terms of performance, these systems are able to achieve roughly an 89.6% accuracy rate of perfectly classifying an equation [2].

More recently, a team from Harvard created a neural network to decompile LaTeX documents [3]. It is particularly noteworthy as the same network architecture is able to be trained on either generating LaTeX from images of a compiled source or generating HTML from images of the rendered webpage. Their model makes use of many recent developments in neural networks. It combines a CNN with a sequence-to-sequence RNN; heuristically, the CNN is used to extract features and the RNN is used to infer syntactical meaning. Figure 1 contains the schematic diagram of their model which heavily influenced the design of our neural network.

Their model begins with a 6 layer CNN. As alluded to earlier, CNN's have a great ability to extract information from images. This is exemplified by Yan LeCun *et al.* in [4] where they used it to achieve cutting edge performance in optical character recognition. Again, it shines in classifying images in [5] and [6]. It is also used in video tagging [7].

Although CNN's are good at classifying, it is not very good at extracting semantics from data. This is why the RNN is used. By tokenizing the LaTeX alphabet, it is possible to used the output of the CNN to drive a "translation" machine. Translation in the sense that it is taking pictures and outputting a new format with the same semantic meaning. RNN's are used heavily in translation tasks. In [8], Cho *et al.* introduced a new scheme called sequence-to-sequence (seq2seq) that allows for arbitrary length of inputs and outputs. This is a desirable trait for our network to have as mathematical

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

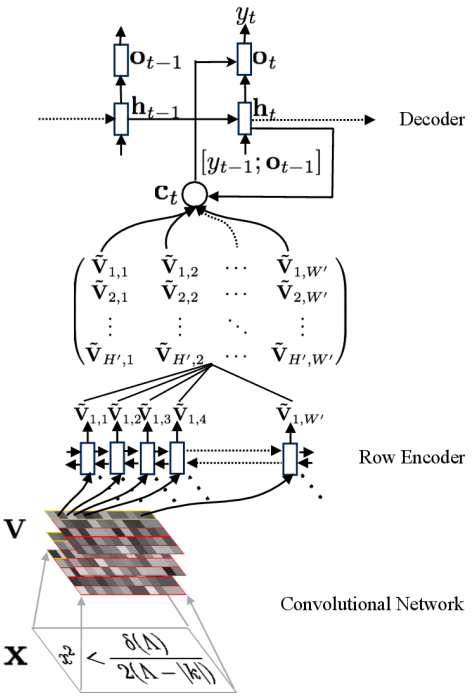

Figure 1: Neural network used by [3] to decompile LaTeX.

equations have unspecified lengths. Finally, the last thing that [3] uses in their LaTeX decompiler is an attention-based encoder-decoder which was introduced by Bahdanau, Cho, and Bengio in [9]. This encoder-decoder allows the output to be a weighted combination of all of the inputs, not just the immediately preceding one. Again, this is useful in mathematical equations as the output could be a function of many neighbouring inputs. An example of this is the integral. Once an integral sign appears, the likelihood of dx appearing is very high. However, it appears much after the integral sign, thus, an attention-based mechanism allows for the neural network to keep emphasis on the integral sign until the dx has appeared.

Problem

The problem at hand is given a black and white image, $x \in \mathcal{X}$, of written formulae and a LaTeX source $y \in \mathcal{Y}$, we wish to find a function f such that the 1/0 risk is minimized. In notation,

$$\arg\min_{f \in \mathcal{F}} \sum_{i=1}^{n} 1_{y_i \neq f(x_i)}$$

More specifically, the functional class $\mathcal{F}$ is the neural network we create and all of its potential weights, $\mathcal{X}$ are images with a certain width and height, and $\mathcal{Y}$ are sequences of tokens that comprise a LaTeX equation.

Model

As said previously, the model we used was greatly influenced by [3] and we define the precise network we used. The first step is to create a batch of images. We used a batch size of 2 as we did not have a lot of data. We also batched the images so that they had similar sizes to the other images in the optimization step to aid with training. Next, it was passed through 6 convolutional layers and 4 pooling layers. Table 1 shows the exact parameterization of the layers. Finally, we used ReLU for the activation units.

Table 1: CNN layers description. c: number of filters, k: kernel size, s: stride, p: padding, po: pooling size, bn: batch normalization.

Conv	Pool
c:512, k:(3,3), s(1,1), p(0,0), bn	-
c:512, k:(3,3), s(1,1), p(1,1), bn	po:(1,2), s:(1,2), p:(0,0)
c:256, k:(3,3), s(1,1), p(1,1)	po:(2,1), s:(2,1), p:(0,0)
c:256, k:(3,3), s(1,1), p(1,1), bn	-
c:128, k:(3,3), s(1,1), p(1,1)	po:(2,2), s:(2,2), p:(0,0)
c:64, k:(3,3), s(1,1), p(1,1)	po:(2,2), s:(2,2), p:(2,2)

After the CNN, they are fed into the row encoder. As mentioned before, the row encoder follows an attention-based mechanism. In this specific implementation, the rows of the CNN layer are fed into the encoder. The reasoning being that equations are read mostly laterally. This is repeated for every row of the CNN output and the encoder produces an equally sized output. To achieve the attention, the encoder is fed into the decoder in such a way that the decoder can influence which parts of the encoder are important. Essentially, it is a backward feedback mechanism that allows the hidden state of the decoder to specify which neighbouring features are the most important.

Finally, we get to the decoder. The decoder has an initial input. It is a special token called `START`. This is coupled with the output of the encoder to produce the next predicted token. This new token is then used as input to the decoder. This is iteratively applied until the decoder produces another special token called `END`. At this point, the prediction process is done. Again, to be able to batch unequally sized formulas, we introduced a third special token called `PADDING`. This works similarly to `END` in the sense that the prediction process should stop when the decoder outputs `PADDING`. Implementation wise, we again followed the exact model in [3] and used a bidirection RNN for the encoder with a hidden state of size 256. The decoder was of size 512 with token embeddings of size 80. Finally, the individual cells of the RNN's were comprised of Long-Short Term Memory cells.

Data

We used a the `IM2LATEX-100K` dataset that provides 103,556 different LaTeX equations coupled with their compiled pdf images. However, no dataset exists for handwritten equations. Thus, we set out to use the `IM2LATEX-100K` as the labels for our new dataset and to generate the inputs. To speed up the process of collecting the data, we created an iPad app that displayed equations from the `IM2LATEX-100K`. We could then input the equation via the iPad and then download the images at the press of a button. Figure 2 shows the layout of the app. The top half displays the equation, the bottom half for writing (empty in this screenshot), and a couple of buttons for minor editing capabilities. While much faster than alternative methods, only 500 equations were collected.

After the data was collected, preprocessing needed to be done. The main issue that needs to be dealt with is to reduce the size of images. Thus, the first thing we did was to find the smallest bounding box and add a 8 pixel padding so that the CNN doesn't suffer from features being located on the edges. Figure 3 shows an example of a preprocessed input.

Another issue is that images should be batches together relative to the similar sizes to reduce the amount of padding that needs to be done. Thus, a custom batching step was created to sort the images according to size and batch the images most similar in size. On the equation side, there are many different normalization techniques that can be used. We opted to follow the steps in [3] and normalize all the equations. For example, ensuring that subscript is before superscript is one such correction. Finally, we opted to use a word based tokenization of LaTeX equations as opposed to individual characters. That is, ‘
sigma’ is considered to be one token even though it takes 6 characters. This was justified because the LaTeX symbols are what is rendered into the image. Finally, we went over the `IM2LATEX-100K` dataset and collected all the symbols and tokenized them. We then removed any token that was used only once.

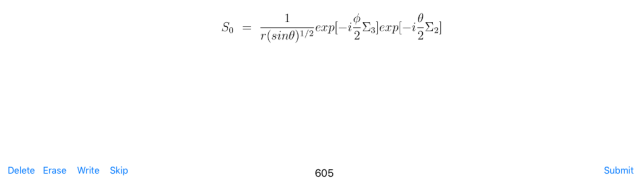

Figure 2: Screen shot of the app created to gather data.

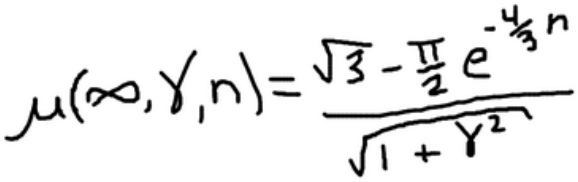

Figure 3: Preprocessed input image.

Training

Training was done on an Amazon AWS EC2 instance with a NVIDIA Tesla K80 to achieve maximum performance. Even with this GPU, training takes a day and is most likely due to the fact of the small dataset. Implementation wise, 80% of the data was used as training samples and 10% of the data was used for validation. After each batched gradient descent, we used the Adadelta method, the validation set was used to assess the current neural net. If it's accuracy increased, the current step sized was not changed, if it did not increase twice in a row, the step size was reduced. The last 10% of the data was used for testing. The algorithm was not trained on this data at any point in the optimization process. We only use this last 10% to quantify any claims.

Results

After almost a full day of training, we get that the word accuracy of our model is 25%. This is below the 75% accuracy that the model we based ours on got on the `IM2LATEX-100K` or the 95% achieved by cutting edge OCR techniques. While this is not the result we wanted, we speculate that the entire gap from 25% to 75% would be reconciled if we had a proper dataset with tens of thousands of data points. Despite the lack of performance, the loss function did systematically get smaller as seen in Figure 4.

One interesting thing to note of the predictions is that they all seem to learn that if one has a padded equation, then the padding always goes at the end. There is never a padding that occurs in the middle. Furthermore, the predictions also know that an equal sign goes somewhere in the middle of the equation and often puts a couple of equal signs in the middle to hedge its bets on where exactly it is. That is, you see many predictions of the form

$$\texttt{START} \rightarrow \ldots \rightarrow = \rightarrow = \rightarrow = \rightarrow \ldots \rightarrow \texttt{END}$$

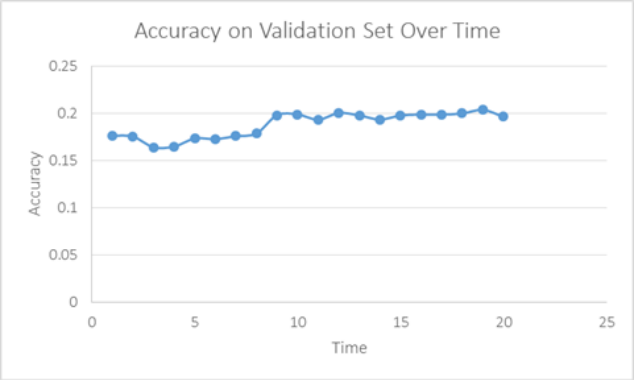

Figure 4: Accuracy over time.

Conclusion

While this method did not achieve state of the art performance, it does provide some positives. For one, it shows the versatility of the model we used. The model is able to generalize the problem of going from images to some type of syntactic language. Secondly, as the loss kept diminishing throughout the entire training period, it hints that more time would have allows the neural net to converge to something better than the result achieved. However, ultimately, better results and a larger dataset go hand in hand. Finally, based on the results of the predictions, using 0/1 loss may have been counterproductive of the goal and resulted in the behaviour seen. That is, with 0/1 loss, the hard assignment forces the neural net to try and get the most number of tokens correct as opposed to confusing the token with a similar looking token, say x and X. Thus, it encourages the neural net to weight the most common symbols such as $=$, $\{$, and $\}$ more heavily. It would be a great extension of this project to see the effects of different loss functions on the prediction outcomes.

References

[1] M. Kumar, "Pca-based offline handwritten character recognition system," *The Smart Computing Review*, vol. 3, no. 5, 2013.

[2] M. Suzuki, F. Tamari, R. Fukuda, S. Uchida, and T. Kanahori, "Infty," *Proceedings of the 2003 ACM symposium on Document engineering*, 2003.

[3] Y. Deng, A. Kanervisto, and A. M. Rush, "What you get is what you see: A visual markup decompiler," 2016.

[4] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, p. 2278–2324, 1998.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[6] C. Szegedy, A. Toshev, and D. Erhan, "Deep neural networks for object detection," in *Advances in Neural Information Processing Systems*, pp. 2553–2561, 2013.

[7] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 1725–1732, 2014.

[8] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," 2014.

[9] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2014.