Image-to-Markup Generation with Coarse-to-Fine Attention

Presenter: Ceyer Wakilpoor

Yuntian Deng[1] Anssi Kanervisto[2] Alexander M. Rush

[1]Harvard University

[3]University of Eastern Finland

ICML, 2017

Outline

Outline

Introduction

- Optical Character Recognition for mathematical expressions
- Challenge: creating markup from compiled image
- Have to pick up markup that translates to how characters are presented, not just what characters
- Goal is to make a model that does not require domain knowledge, use data-driven approach
- Work is based on previous attention-based encoder-decoder model used in machine translation and in image captioning
- Added multi-row recurrent model before attention layer, which proved to increase performance

Outline

Problem Statement

- Converting rendered source image to markup that can render the image
- The source, $x \in \mathcal{X}$ is a grayscale image of height, H, and width, W ($\mathbb{R}^{HxW}$)
- The target, $y \in \mathcal{Y}$ consists of a sequence of tokens $y_1, y_2, ..., y_C$
  - C is the length of the output, and each y is a token from the markup language with vocabulary $\sum$
- Effectively trying to learn how to invert the compile function of the markup using supervised examples
- Goal is for $compile(y) \approx x$
- Generate hypothesis $\hat{y}$, and $\hat{x}$ is the predicted compiled image
  - Evaluation is done between $\hat{x}$ and x, as in evaluating to render an image similar to the original input

Outline

Model

- Convolutional Neural Network (CNN) extracts image features
- Each *row* is encoded using a Recurrent Neural Network (RNN)
- When paper mentions RNN, it means a Long-Short Term Memory Network (LSTM)
- These encoded features are used by an RNN decoder with a visual attention layer, which implements a conditional language model over $\sum$

Outline

Convolutional Network

- Visual features of the image are extracted with a multi-layer convolutional neural network, with interleaved max-pooling layers
  - Based on model used for OCR by Shi et al.
- Unlike some other OCR models, there is no fully-connected layer at the end of the convolutional layers
  - Want to preserve spatial relationship of extracted features
- CNN takes in input $\mathbb{R}^{HxW}$, and produces feature grid, $\mathbf{V}$ of size $CxH'xW'$ where c denotes the number of channels, and H' and W' are the reduced dimensions from pooling

Outline

Row Encoder

- Unlike with image captioning, OCR there is significant sequential information (i.e. reading left-to-right)
- Encode each row separately with an RNN
  - Most markup languages default left-to-right, which an RNN will naturally pick up
  - Encoding each row will allow the RNN to use surrounding horizontal information to improve the hidden representation
- Generic RNN: $h_t = RNN(h_{t-1}, v_t; \theta)$
- RNN takes in $\mathbf{V}$ and outputs $\tilde{\mathbf{V}}$:
  - Run RNN over all rows $h \in \{1, ..., H'\}$ and columns $w \in \{1, ..., W'\}$
  - $\tilde{\mathbf{V}} = RNN(\tilde{\mathbf{V}}_{h,w-1}, \mathbf{V}_{h,w})$

Outline

Decoder

- Decoder is trained as conditional language model
- Modeling probability of output token conditional on previous ones: $p(y_{t+1}|y_1, ..., y_t, \tilde{\mathbf{V}} = softmax(\mathbf{W}^{out}\mathbf{o}_t)$
- $\mathbf{W}^{out}$ is a learned linear transformation and $\mathbf{o}_t = tanh(\mathbf{W}^c[\mathbf{h}_t; \mathbf{c}_t])$
- $\mathbf{h}_t = RNN(\mathbf{h}_{t-1}, [y_{t-1}; \mathbf{o}_{t-1}])$

Outline

Attention

- General form of context vector used to assist decoder at time-step t:

$$\mathbf{c}_t = \phi(\{\tilde{\mathbf{V}}_{h,w}\}, \alpha_t)$$

- General form of $\mathbf{e}$ and the weight vector, α:

$$\mathbf{e}_t = a(\mathbf{h}_t, \{\tilde{\mathbf{V}}_{h,w}\})$$

$$\alpha_t = softmax(\mathbf{e}_t)$$

- From empirical success choose a: $e_{it} = \beta^T tanh(W_h \mathbf{h}_{i-1} + W_v \tilde{\mathbf{v}}_t)$ and $\mathbf{c}_i = \sum_i \alpha_{it} \mathbf{v}_t$
- $\mathbf{c}_t$ and $\mathbf{h}_t$ are simply concatenated and used to predict the token, y_t

Model

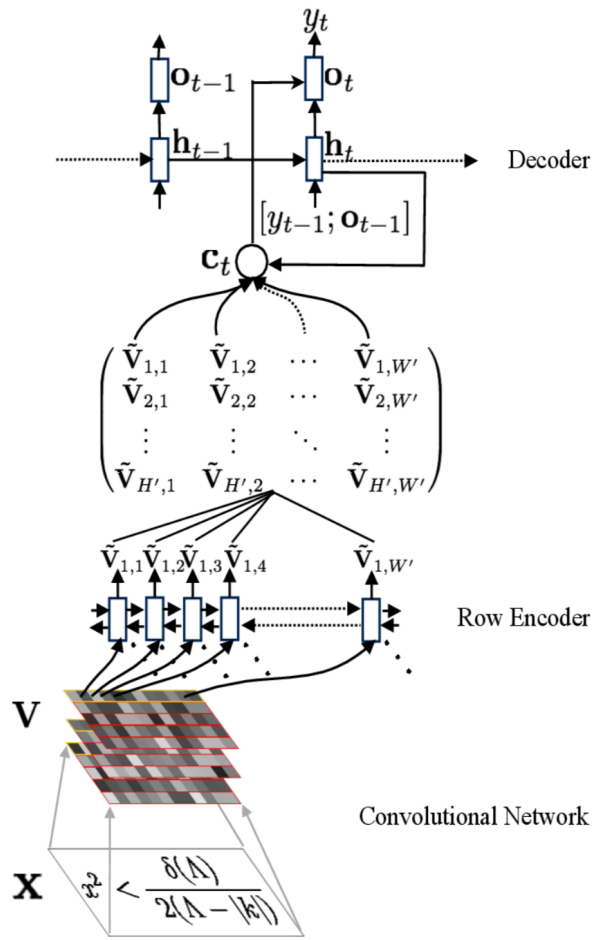
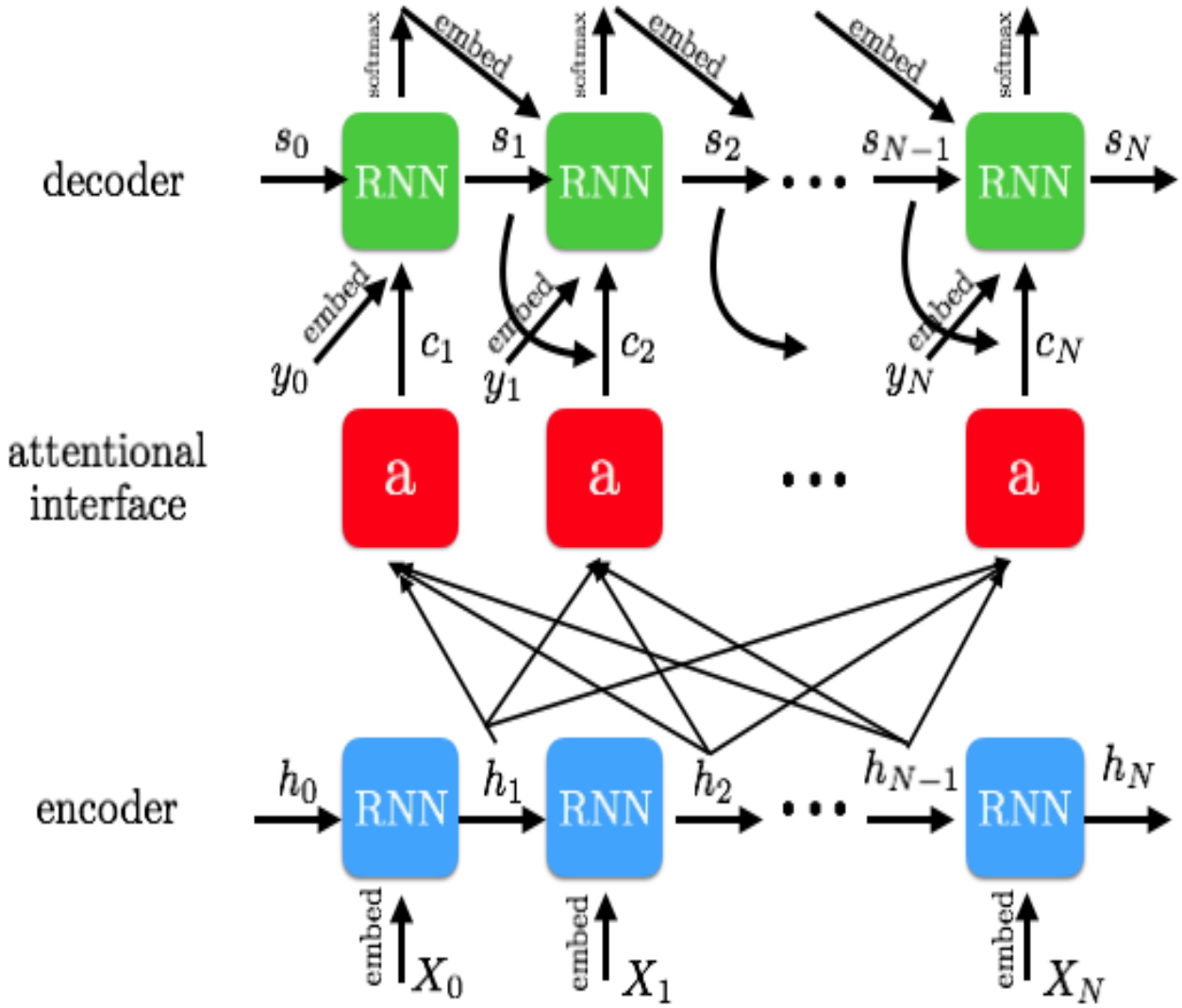
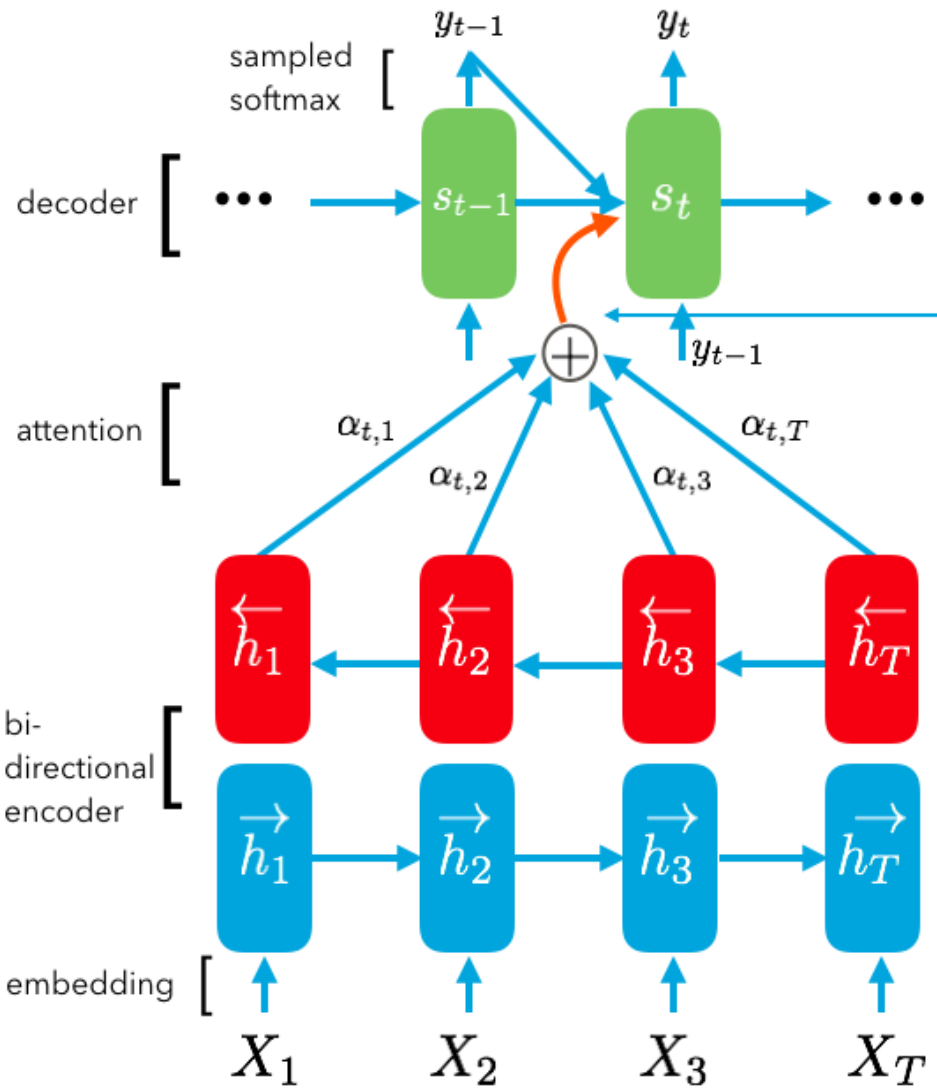


Figure 2: Network structure of WYGIWYS. Given an input image, a CNN is applied to extract visual features, then for each row in the final feature map we employ an RNN encoder. The encoded features are then used by an RNN decoder with a visual attention mechanism to produce final outputs. For clarity we only show the RNN encoding at the first row and the decoding at time step t.

Attention


decoder
attentional interface
encoder
RNN
a
softmax
embed
s_0
s_1
s_2
s_{N-1}
s_N
c_1
c_2
c_N
y_0
y_1
y_N
h_0
h_1
h_2
h_{N-1}
h_N
X_0
X_1
X_N

Encoder Decoder with Attention



$$p(y_i|y_1,...,y_{i-1},X) = g(y_{i-1},s_i,c_i)$$

$$s_i = f(s_{i-1},y_{i-1},c_i)$$

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

$$\alpha_{i,j} = \frac{exp(e_{ij})}{\sum_{k=1}^{T_x} exp(e_{ik})}$$

$$e_{ij} = a(s_{i-1},h_j)$$

where a is a feed forward neural network.

$$h_j = [\rightarrow h_j; \leftarrow h_j]_{concat}$$

Example

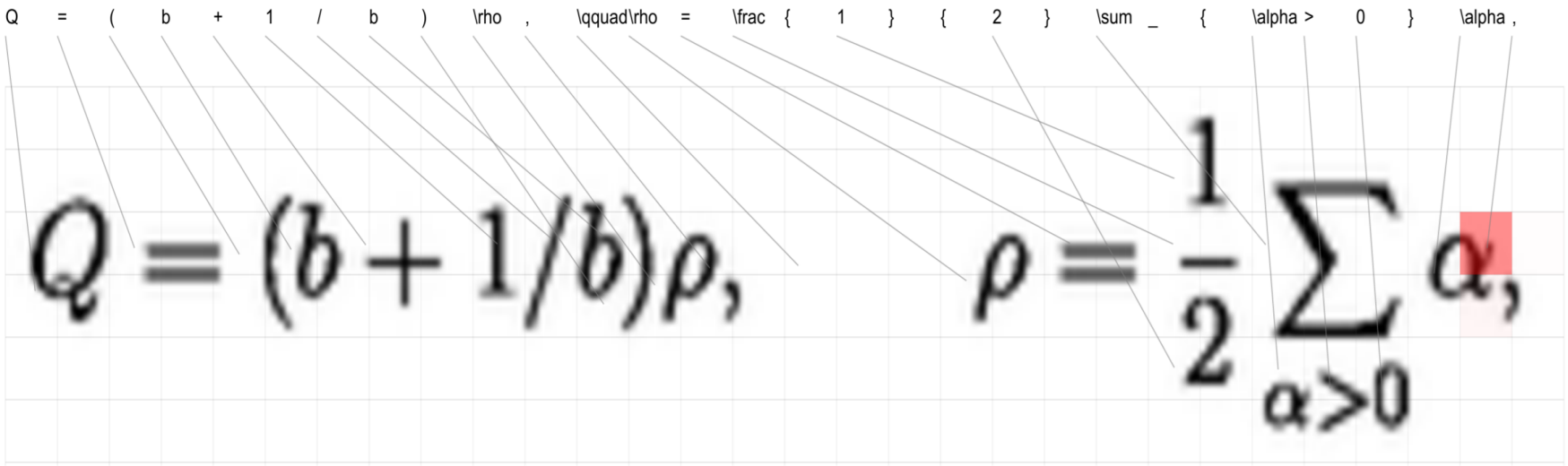


Figure 1: Example of the model generating mathematical markup. The model generates one LaTeX symbol y at a time based on the input image $\mathbf{x}$. The gray lines highlight $H' \times V'$ grid features after the CNN $\mathbf{V}$ and RNN Encoder $\tilde{\mathbf{V}}$. The dotted lines indicate the center of mass of α for each word (only non-structural words are shown). Red cells indicate the relative attention for the last token. See `http://lstm.seas.harvard.edu/latex/` for a complete interactive version of this visualization over the test set.

Model Architecture

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)

Table 2: CNN specification. 'Conv': convolution layer, 'Pool: max-pooling layer. 'c': number of filters, 'k': kernel size, 's': stride size, 'p': padding size, 'po': , 'bn': with batch normalization. The sizes are in order (height, width).

Outline

Experiment Details

- Beam search used for testing since decoder models conditional language probability of the generated tokens
- Primary experiment was with IM2LATEX-100k, which is a dataset of mathematical expressions written in latex
- Latex vocabulary was tokenized to relatively specific tokens, modifier characters such as ^ or symbols such as \sigma

Transform	Original	Normalized
SubSup	`H^I_1`	`H_I^1`
ReqBrack	`H^I`	`H^{I}`
Desugar	`H'`	`H^{\prime}`
ExpOperators	`\sin`	`\operatorname{sin}`
InfixPrefix	`\over`	`\frac{}{}`
MatrixEnv	`\matrix`	`\begin{array}...`
Drop	`\label{}`	–

Table 1: Preprocessing transformations applied to LaTeX abstract syntax tree in normalizing mode. These transformations are mostly safe, although there are some corner cases where they lead to small differences in output.

Experiment Details

- Created duplicate model without encoder as control to test against image captioning models, the control was called CNNEnc
- Evaluation by comparing input image and rendered image of output latex
- Initial learning rate of .1 and halve it once validation perplexity doesn't decrease
- Low validation perplexity means good generalization when comparing training to validation set
- 12 epochs and beam search using beam size of 5

Outline

Results

- 97.5% exact match accuracy of decoding HTML images
- Reimplement Image-to Caption work on Latex and achieved accuracy of over 75% for exact matches

Model	Preprocessing	BLEU (tok)	BLEU (norm)	Edit Distance	Exact Match	Exact Match (-ws)
INFTY	-	51.20	66.65	53.82	15.60	26.66
CNNENC	norm	52.53	75.01	61.17	53.53	55.72
WYGIWYS	tok	73.71	73.97	84.26	74.46	77.04
WYGIWYS	norm	58.41	87.73	87.60	77.46	79.88

Table 3: Main experimental results on the IM2LATEX-100K dataset. Reports the BLEU score compared to the tokenized formulas (BLEU (tok)), and the BLEU score compared to the normalized formulas (BLEU (norm)), column-wise image edit distance, exact match, and exact match without whitespace columns.

Results

$$Z = \sum_{spins} \sum_{cubes} \prod W(a|e,f,g|b,c,d|h),$$

$$\{\Psi \circ \mu, f\} = \left(\overline{X}_i f\right)\left(Y^i \Psi\right) \circ \mu\,,$$

$$U_n(\theta,\phi) = \begin{pmatrix} \cos(\theta/2) & -e^{-in\phi}\sin(\theta/2) \\ \sin(\theta/2)e^{in\phi} & \cos(\theta/2) \end{pmatrix}$$

$$\sin\frac{\pi\alpha' s}{2} + \sin\frac{\pi\alpha' t}{2} + \sin\frac{\pi\alpha' u}{2} = -\frac{\pi^3}{16}\alpha'^3 stu + o(\alpha'^5),$$

Figure 5: Typical errors in the LaTeX dataset. We show the operations needed to get ground truth from the rendered predictions. Red denotes add operations and blue denotes delete operations.

Implementation

- Mostly written in Torch, Python for preprocessing, and utilized lua libraries
- Bucketed inputs into similar size images

Citations


https://theneuralperspective.com/2016/11/20/recurrent-neural-network-rnn-part-4-attentional-interfaces