BERT's Handwriting
Testing BERT Word Prediction Effect on Accuracy in Handwritten Text Recognition

Anna C. Widder[*]
Department of Mechanical Engineering
Stanford University
`awidder@stanford.edu`

Abstract

In this paper I demonstrate that the addition of BERT to handwritten text recognition does not improve accuracy in transcription. My model is trained on the well-developed IAM dataset and can accept new input from a user in the form of sentences. The model first recognizes the words in a given handwritten sentence and then feeds the predicted text to BERT with low-certainty words masked out. The result is no significant change in character or word error rate for texts decoded with BERT vs. those without.

1 Introduction

The first handwriting-recognition tablet debuted in 1987 with the Linus Write-Top, which marked the entrance of handwritten text into the digital world of the average worker. Previously images of handwriting were primarily used for archival purposes and were not practical in daily use. As tablet technology has accelerated, the inherent incompatibility between handwritten text and the digital world is a problem with growing demand for a solution. Digital notetaking, scans of whiteboards and brainstorming sessions, online collaboration—all of this could be digitized and therefore searchable, compressible, and much more. Herein lies the interest of this problem: handwritten text recognition programs exist, but they often mistake words and don't really "think" about what the sentence as a whole most likely says.
The input to my algorithm is an grayscale image (800x64) of a sentence of handwritten text, black words on a white background. I then use an handwritten text recognizer (HTR) to predict the text of the sentence with some certainty of each word. The HTR consists of a convolutional neural network (CNN) to extract important features, followed by a two layer BLSTM (RNN) to encode the sequencing and time-step of the characters, and lastly a CTC layer that decodes the text with an associated relative uncertainty on each word. The most uncertain words are masked out and the text is fed as input to a BERT trained algorithm, which outputs the final predicted text. In short, we input a grayscale image of handwritten text to an HTR+BERT algorithm that outputs a predicted transcription.

2 Related work

There are already many successful methods for HTR. When HTR first began in the 1990s, character-based recognizers were based on Hidden Markov Models (HMMs) [22, 1, 14]. However, for a given

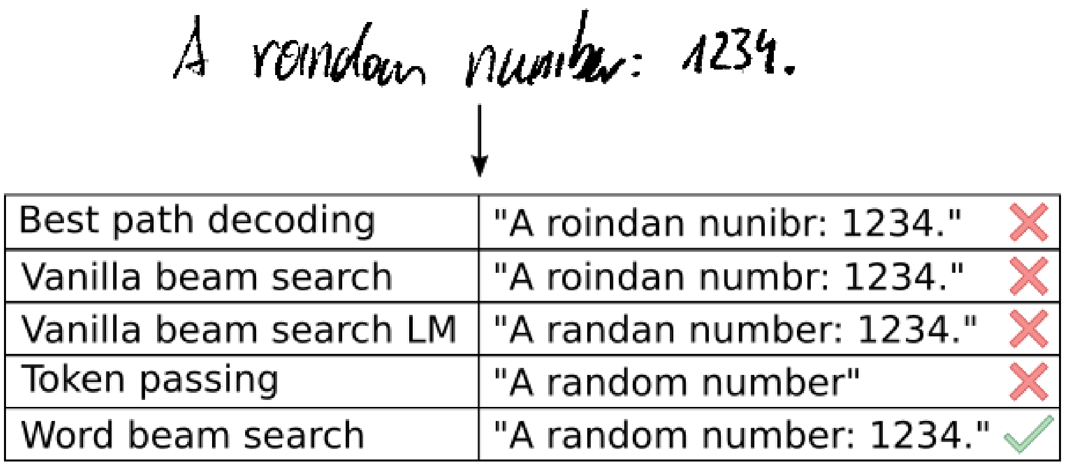
[*]TBD

character in a word HMMs inherently do not consider the previous or following characters. This ultimately limited the attainable accuracy and extension of such models. [7]

As the field progressed, HMMs were abandoned for a neural networks. CNNs proved effective at character and even word recognition, since the networks detect higher and higher level features and have proved useful for raw image classification. [10]. However, CNNs are not designed for sequence data. Consequently, Shi, Bai and Yao recently proposed a "CRNN" network: a convolutional, recurrent neural network [21]. This network uses a CNN to learn relevant representations directly from image data without preprocessing, then uses an RNN to produce a sequence of labels (in this case words) before finally translating the RNN output into text. This model combines the benefits of RNNs as discussed in Graves' work with the feature recognition of CNNs to tackle HTR from multiple angles. [9, 7, 10]

The RNN layers of current HTR models are specially designed for textual data. The layers employ *Bi-Directional Long Short-Term Memory* models (BLSTMs), which pass the input data forwards and backwards through two hidden LSTM layers connected to the same output. The LSTM layers are comprised of recurrently connected subnets, called memory blocks, which store information over time to combat vanishing gradients in the RNN. [12] The forwards and backwards pass of BLSTMs are useful for sequential data as they allow information about the past and future to be used in the network. [10, 8, 11] The BLSTM layers of HTRs are trained with Connectionist Temporal Classification (CTCs). This method, proposed by Graves et al., allows RNNs such as the BLSTM to be trained on pairs of data (images) and target labelings (text). CTC loss with an RNN removes the need for presegmented training data and post-processed outputs that is normally required by CNNs. [9, 20]

The final labeling (text) output of the model is computed by the decoding algorithm of the RNN/BLSTM output. The current state-of-the-art decoding algorithms are Vanilla Beam Search (VBM) proposed by Hwang and Sung and uses character-level beam search with a character-level language model (LM), token passing proposed by Graves which constrains its output to a dictionary of words and uses a word-level LM, and most recently a Word Beam Search (WBS), which confines VBM to a dictionary and allows non-alphanumeric characters. [13, 8, 20]


A random number: 1234.

Best path decoding	"A roindan nunibr: 1234."	✗
Vanilla beam search	"A roindan numbr: 1234."	✗
Vanilla beam search LM	"A randan number: 1234."	✗
Token passing	"A random number"	✗
Word beam search	"A random number: 1234."	✓

Example decodings of the difference CTC decoding methods. Image from [19].

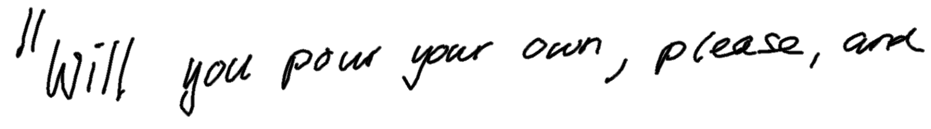
Shifting away from HTR, language modeling has its own rich and complex models. In general, language models are pretrained on enormous amounts of textual data before being applied to a variety of down-stream tasks. Current methods for creating the pretrained model are a feature based approach, a fine tuning method, and the new BERT model. Briefly, the fine tuned method finely tunes all of the model's parameters to the end-goal task, as in the Generative Pre-trained Transformer (OpenAI GPT), while feature based approach uses task-specific models that incorporate pre-trained representations as additional features of the data, as in ELMo. [18, 16] The language model BERT (Bidirectional Encoder Representations from Transformers) is designed to "pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers." [3] Essentially, BERT uses a masked language model which randomly masks out some tokens (ie words) of the input and attempts to correctly identify them from a given vocabulary.

3 Dataset and Features

I am using the IAM Handwriting Database, which contains handwriting samples from 657 different people. [15] Specifically, I am using 13,353 isolated and labeled text lines, which is in the form of grayscale .png images of varying size and a text file containing the label information.

The line images are read in with OpenCV-Python, a computer vision library. A random distortion is applied to augment the data set, the contrast is enhanced, and then the image is resized to 64 x 800. Additionally, there are known corrupted images in the IAM set and such images are replaced with a completely black background.

The data set is split into 90% training, 5% validation, and 5% test, and of the training set only 9500 random examples of each batch are chosen per epoch. The mini-batch size is 50. In the BLSTM, 100 time steps are used.



"Will you pour your own, please, and

Sample image from the IAM database, reading "*"Will you pour your own, please, and.*"

4 Methods

For the HTR section of my project, my algorithm is a CNN followed by a BLSTM RNN and CTC loss with WBS.

The CNN works by alternating convolution layers with pooling layers to extract the important features of the text. Convolution layers essentially filter the input (in this case images) to produce a new version of the input with the filter applied. The filter can be thought of as something that highlights the salient components of the input. The pooling layers discretize the input and reduce its dimensionality, which allows for assumptions to be made about features contained in the discretized portions of the input. Because I am using the CRNN method, I do not use the traditional fully connected layer in my CNN.

The BLSTM RNN works by feeding the output of the CNN both forwards and backwards through the network to extract the sequence dependency of the text. The memory blocks of the BLSTM is able to store more information for longer than a standard RNN, which makes it ideal for HTR.

My CTC loss uses WBS as its decoding mechanism. Beam search in general works by generating the translation word by word from left-to-right while keeping a fixed number (beam width) of active candidates at each time step. By increasing the beam width, the translation performance can increase at the expense of significantly reducing the decoder speed. [4]. WBS is a relatively new method that uses beam search restricted to a dictionary while allowing for non-alphanumeric characters between words and using a LM. WBS can use a variety of different methods with different degress of complexity to score the beams [20]:

- Words": only use dictionary, no scoring: O(1)
- "NGrams": use dictionary and score beams with LM: O(log(W))
- "NGramsForecast": forecast (possible) next words and apply LM to these words: O(W*log(W))
- "NGramsForecastAndSample": restrict number of (possible) next words to at most 20 words: O(W)

The trainable hyperparameters for HTR with WBS are the beam width, the scoring mode, and the LM smoothing. I also used early stopping to prevent long runtimes when no progress was being made.

WBS outputs the beam scores for each possible character for every character position in each word it decodes. This output is of shape (length of possible text) x (batch size) x (number of possible characters). This beam score is how I derive my certainty of the decoded word. I used two methods to determine the certainty of a word: average top beam score of the word and average chosen character beam score of the word. The average top beam score of a word takes the top beam score for every character in a word (regardless of what character that beam score corresponds to) and divides by the length of the word. The average chosen character beam score takes the beam score of the chosen character for every character in a word and divides by the length of the word.

The trainable hyperparameters for this certainty section is which method to use and what threshold to choose for that method.

Finally, based on the certainty scores for each word in a sentence, I individually mask each word in the sentence with a score below the threshold and pass the masked text to BERT for prediction. This prediction of masked words is the same operation as the training method for BERTs encodings, the details of which are beyond this report.

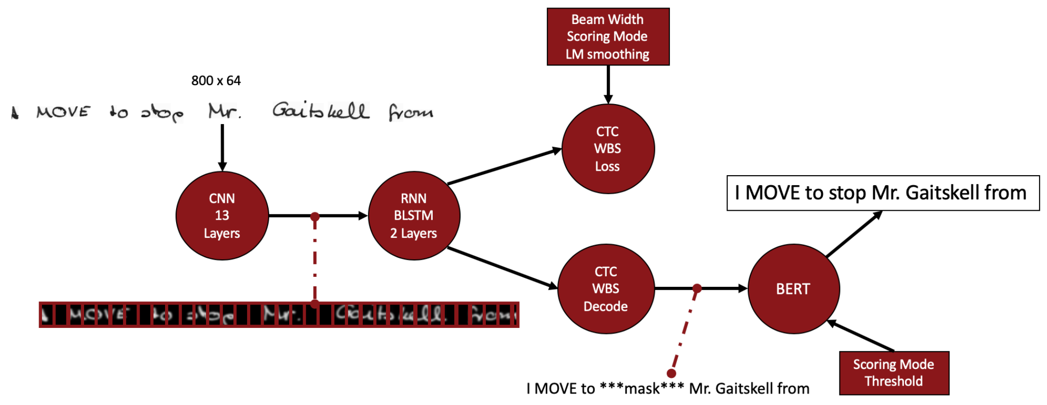


Diagram of model.

5 Experiments/Results/Discussion

For my code HTR section, I draw heavily on the previous code bases of Sushant Gautam and Harald Scheidl [5, 19]. MY HTR begins with a 13 layer CNN with CONV and POOl layers and leaky ReLU activation some additional batch normalization. The layer breakdown is: a 5x5 CONV with 2x2 POOL, a 5x5 CONV with a 1x2 POOL, a 3x3 CONV with a 2x2 POOL with simple batch normalization, a 3x3 CONV, a 3x3 CONV with 2x2 POOL, a 3x3 CONV with 1x2 POOL and simple batch normalization, and lastly a 3x3 CONV with 1x2 POOL for a final output shape of 100x1x512. Next is a 2 layer BLSTM RNN with 512 hidden cells trained on a CTC loss. Finally I decode the RNN output using WBS.

For the HTR I used a mini-batch size of 50, which provided more data to be processed per batch and gave better results. For the trainable hyperparameters for HTR with WBS, I used a beam width of 20, the NGramsForecastAndSample scoring method, and an LM smoothing of 0.1. I chose these exact values empirically, but with some thought. I chose to use NGramsForecastAndSample because I wanted to take advantage of my data being lines of text and not individual words (NGramsForecast), but not ridiculously increase my runtime (NGramsForecastAndSample).

For the BERT masking and prediction, I used a wonderful library called FitBERT which takes a string with a single masked word and a list of options and uses BERT to predict the most likely fill in. The model of BERT used is the "large uncased" version.

My metrics for success were character error rate (CER), or the number of correct characters divided by the total number of characters, and the word error rate (WER), or the number of correct words divided by the total number of words.

Since my project has two components, I trained my models separately. I struggled a lot to achieve character and word error rates for the HTR portion that would even remotely allow for BERT to succeed. Despite training and training, and tuning hyperparameters all over the place, my CER hovered at 30% and my WER at 80%. Finally I hit on the parameters described above and was able to move on to the certainty and BERT tuning. My HTR ultimately had a CER of 5.2512% and WER of 53.0769% on the test set.

```
Batch: 1 / 13
Ground truth -> Recognized
[ERR:50] "Courier knows what 's going on . If he doesn't mind ," -> "toe"
[ERR:41] "why should you ? " " I see . " John took the" -> "toe"
[ERR:43] "refilled glass and looked over the rim at his" -> "toe"
[ERR:48] "companion . " You want me to talk , is that it ? "" -> "toe"
```

Sample output of an early run on my HTR model. Outputs such as this made training the HTR difficult and slow.

Using the average top beam score method of certainty for BERT masking, my results were qualitatively the same as if I had not done any BERT masking at all. I found empirically that the best threshold was a score of 3, which resulted in a CER of 5.2832% but a WER of 52.7692%.

The average character beam score method did not fair as well. I found that qualitatively the model did worse at translating the texts than without a BERT mask. Essentially, I masked out words that the HTR would have gotten right and then BERT got them wrong. Empirically the best threshold was around -9, which resulted in a CER of 5.5491% and a WER of 54.0000%. This method may not have done as well because the individual characters of each word may not have had the best beam score since the beams were judged on a word, rather than character level.

Overall the addition of a BERT layer did not dramatically improve the performance of HTR but it did increase the runtime significantly. It is odd that my model performed better on the test set than the training validation set, but I believe this is due only to irregulariteis in the data and would be solved with an expanded dataset. I believe my model struggled with the dataset, which was short lines of text and not full sentences or paragraphs that BERT was trained on. I augmented the data in an attempt to increase my regularization but it did not help.

```
[OK] "" It would make me more interesting ," -> "" It would make me more interesting ,"
[OK] "I suppose ? " she enquired archly ." -> "I suppose ? " she enquired archly ."
[ERR:1] "" No , it wouldn't ! " he almost snapped ," -> "" No , it wouldn't ! " he almost sapped ,"
[ERR:2] "surprising her . " You sounded like some-" -> "surprising her " You sounded like some-"
[OK] "body else for a moment there . It doesn't" -> "body else for a moment there . It doesn't"
[ERR:2] "become you to behave like a - a tart , Di . "" -> "become you to behave like a - a tart Di . ""
[OK] "" Nigel - ! " she gasped . " I didn't mean to" -> "" Nigel - ! " she gasped . " I didn't mean to"
[ERR:1] "behave like anything of the kind . I" -> "behave like anything of the kind . ."
[OK] "was only joking . " " Well , don't . It 's" -> "was only joking . " " Well , don't . It 's"
[OK] "miserable enough my having to take my" -> "miserable enough my having to take my"
```

Sample output from the successful model run with average top scoring.

My code is at `https://github.com/kiprin629/cs230_bh.git`

6 Conclusion/Future Work

Overall my project investigated whether adding a BERT masking layer to HTR would improve character and word error rates. Based on an HTR netwrok with a CNN to RNN to CTC loss with word beam search, adding a layer of BERT masking does not improve CER and very very minimally improves WER at the cost of significantly increased runtime.

Given more time in the future, I think this project would find more success if it was extended to HTR for paragraphs. This could be done by adding line segmentation and using full paragraphs of text as an input. Since BERT was trained on full sentences, this is likely to be the change that would most dramatically improve the accuracy. Additional metrics for masking words could be investigated, for example some measure of the certainty of the word that is compute on a word level rather than an average of the character-level beam scores. The image processing for the input data could be further developed to smooth out the background and handle real-time images. The BERT masking may be more successful if the code were extended to mask off more than one word at a time. I think also that the dataset could be updated to include a variety of handwriting types, since this dataset is older and many of the samples are in cursive.

References

[1] Yoshua Bengio et al. "LeRec: A NN/HMM hybrid for on-line handwriting recognition". In: *Neural computation* 7.6 (1995), pp. 1289–1303.

[2] D. C. Ciresan et al. "Convolutional Neural Network Committees for Handwritten Character Classification". In: *2011 International Conference on Document Analysis and Recognition*. Sept. 2011, pp. 1135–1139. DOI: `10.1109/ICDAR.2011.229`.

[3] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[4] Markus Freitag and Yaser Al-Onaizan. "Beam Search Strategies for Neural Machine Translation". In: *Proceedings of the First Workshop on Neural Machine Translation* (2017). DOI: `10.18653/v1/w17-3207`. URL: `http://dx.doi.org/10.18653/v1/W17-3207`.

[5] Sushant Gautam. *Handwritten Line Text Recognition using Deep Learning with Tensorflow*. `https://github.com/sushant097/Handwritten-Line-Text-Recognition-using-Deep-Learning-with-Tensorflow.git`. 2019.

[6] Felix A Gers, Nicol N Schraudolph, and Jürgen Schmidhuber. "Learning precise timing with LSTM recurrent networks". In: *Journal of machine learning research* 3.Aug (2002), pp. 115–143.

[7] Alex Graves and Jürgen Schmidhuber. "Offline handwriting recognition with multidimensional recurrent neural networks". In: *Advances in neural information processing systems*. 2009, pp. 545–552.

[8] Alex Graves et al. "A novel connectionist system for unconstrained handwriting recognition". In: *IEEE transactions on pattern analysis and machine intelligence* 31.5 (2008), pp. 855–868.

[9] Alex Graves et al. "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks". In: *Proceedings of the 23rd international conference on Machine learning*. ACM. 2006, pp. 369–376.

[10] Alex Graves et al. "Unconstrained on-line handwriting recognition with recurrent neural networks". In: *Advances in neural information processing systems*. 2008, pp. 577–584.

[11] Adnan Ul-Hasan and Thomas M Breuel. "Can we build language-independent OCR using LSTM networks?" In: *Proceedings of the 4th International Workshop on Multilingual OCR*. ACM. 2013, p. 9.

[12] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[13] Kyuyeon Hwang and Wonyong Sung. "Character-level incremental speech recognition with recurrent neural networks". In: *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2016, pp. 5335–5339.

[14] Stefan Knerr et al. "Hidden Markov model based word recognition and its application to legal amount reading on French checks". In: *Computer Vision and Image Understanding* 70.3 (1998), pp. 404–419.

[15] U-V Marti and Horst Bunke. "The IAM-database: an English sentence database for offline handwriting recognition". In: *International Journal on Document Analysis and Recognition* 5.1 (2002), pp. 39–46.

[16] Matthew E Peters et al. *System and methods for performing nlp related tasks using contextualized word representations*. US Patent App. 16/223,739. June 2019.

[17] Melisa Qordoba and Sam Qordoba. *FitBERT*. `https://github.com/Qordobacode/fitbert.git`. 2019.

[18] Alec Radford et al. "Improving language understanding by generative pre-training". In: *URL https://s3-us-west-2. amazonaws. com/openai-assets/researchcovers/languageunsupervised/language understanding paper. pdf* (2018).

[19] Harald Scheidl. *Handwritten Text Recognition with TensorFlow*. `https : / / github . com / githubharald/SimpleHTR.git`. 2019.

[20] Harald Scheidl, Stefan Fiel, and Robert Sablatnig. "Word Beam Search: A Connectionist Temporal Classification Decoding Algorithm". In: *2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. IEEE. 2018, pp. 253–258.

[21] Baoguang Shi, Xiang Bai, and Cong Yao. "An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition". In: *IEEE transactions on pattern analysis and machine intelligence* 39.11 (2016), pp. 2298–2304.

[22] Matthias Zimmermann and Horst Bunke. "Automatic segmentation of the IAM off-line database for handwritten English text". In: *Object recognition supported by user interaction for service robots*. Vol. 4. IEEE. 2002, pp. 35–39.