Gregg Shorthand recognition with deep neuron networks

Brian Chu, Mason Sawtell, Yi Dai

March 2021

1 Introduction

1.1 Motivation

Transcribing written letters and numbers is often a litmus test for those who are learning about neural networks; datasets like the MNIST digits have made it easy to source a large number of points to train on, and classifying characters is a relatively simple task. However, we wanted to see if we could use a neural network approach on other systems of writing as well, which have different traits that make the task of identifying what is written more challenging. One particular candidate, Gregg shorthand, stood out to us because it utilizes connected glyphs, like cursive, but doesn't remotely look like standard written English, which means that it's more challenging to correlate between the written representation and the target word.

1.2 Gregg Shorthand

Gregg shorthand is a writing system invented in the early 20th century, intended on speeding up transcription of the spoken word. Instead of abbreviating each letter, like other systems of fast writing, Gregg has a representation for each sound being said. This way, transcription is more efficient since different ways of saying the same sound can be compressed into one representation. For ease of writing, each representation is based on the arcs and radii of an elliptical figure; adjacent sounds are indicated by one shape ending where the next shape begins. In some ways, this makes Gregg similar to cursive handwriting, since each word (or group of words) is a continuous curve.

Correlating between handwritten examples and the actual word is somewhat more challenging than if we were to look at individual letters, because we can't just analyze the discrete components of a word and string them together. Instead we have to look at the word as a whole. As such, this poses an interesting problem for a neural network to solve, given that it must also utilize feature detection to notice the components of the shorthand and transcribe that into the syllables of a word.

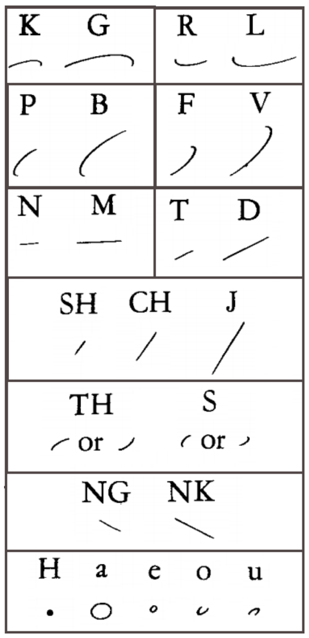

Figure 1: Basic shapes in Gregg Shorthand

2 Data

2.1 Collection

Gregg shorthand is relatively rare and there are few datasets that are widely available for public consumption. However, we were able to source a reasonably large corpus of data, the Gregg-1916 dataset, which is comprised of digital, black-and-white scans of words from the 1916 Gregg Shorthand Dictionary. In total, there are 15,711 images, each with a corresponding label indicating which word is represented.

2.2 Preliminary analysis and augmentation

”apprehensive”

”zincification”

Every image is in black and white, with no intermediate shades of gray, so we have nice binary representations that are easier to process since there’s only one bit of data per pixel. However, because of how different words are different shapes, every image has a different width and height. Variable length encoding is a hassle to deal with, but every shape has to be a particular size, as shown by the shapes for ”sh”, ”ch”, and ”j” sounds in figure 1, so we can’t just resize the images, otherwise the scaling and angles would all be wrong. In addition, because the shapes are not invariant under rotation we can’t rotate them by too much either. So, to make the images consistent, we mostly apply padding to the images to make them a consistent size, with conservative augmentation (scaling by at most %96 size and rotating by up to 2°).

3 Methods

3.1 Architecture Design

As a typical image recognition problem, CNN is very useful in extracting patterns from Gregg Shorthand. But in the case of recognizing entire words, a simple CNN would not work. Here we use the neuron network architecture of Zhai’s work [1] as reference. In summary, a CNN-based feature extractor generates a feature vector from the input. The resulting vector is then fed into a sequence generator that creates hypotheses of labels. Since the hypotheses are not completely correct words, a word retrieval module predicts what word was being generated using string similarity criteria to determine the best prediction.

The feature extractor is a series of 10 convolutional layers, with batch normalization and max-pooling layers in between, followed by two fully connected layers. This component uses a binary classification approach, where we check to see if specific characters in the training alphabet appear in the label. The main difference between the approach used in the paper and our project was the alphabet of possible characters. While the paper trains the feature extractor on the English alphabet from ”a” to ”z”, our approach was to train on the CMU Pronouncing Dictionary [2], which maps English words to standardized syllabic units. We first map the original English label to the list of its syllables, then use that list as the new label upon which to train. Our approach was suggested as a follow-up to the work done in the original paper, since Gregg shorthand is based on the pronunciations of words, so with the availability of various pronunciation dictionaries, it makes it simpler to recognize the individual syllables and use that to retrieve the word.

The rest of the model works in the same way as Zhai’s approach [1]. With the sequence generator, we utilize a bidirectional decoding method where we generate a hypothesis starting at the beginning of the feature vector and moving forwards, and a hypothesis starting at the end of the feature vector and moving backwards. This is done because the information in the feature vector gets noisy as the sequence gets longer, so predictions

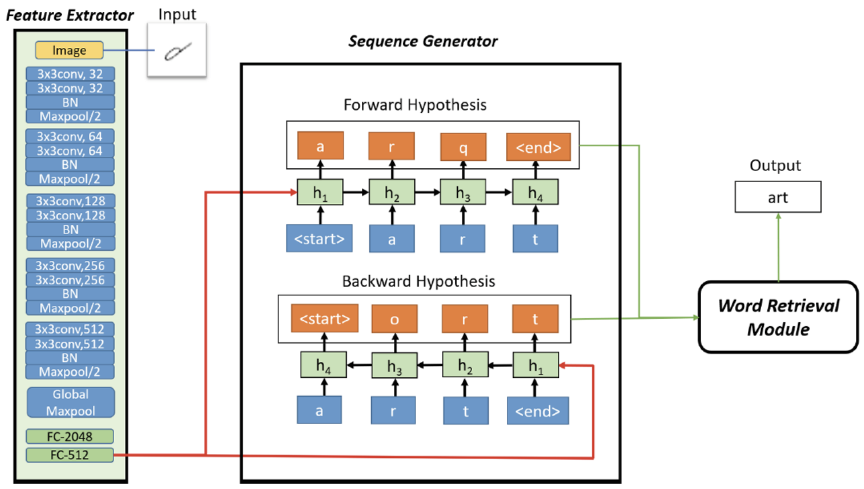

Figure 2: Architecture of the neuron network. Instead of the characters as shown in the figure, we use standardized syllabic units as basic units.

start off strong and worsen as we move along the vector. By moving in both directions, we can get strong predictions for both the beginning and the end of the target word.

Finally, in the word retrieval module, we take both the forward and backward hypotheses and use various metrics to determine which word in the dictionary fits the best with the hypotheses. In particular, we used the Levenshtein distance, BLEU, and bidirectional BLEU. Levenshtein distance counts editorial operations (inserting/removing/replacing characters) and is a basic measure of how different two strings are from each other. BLEU is a standard algorithm to evaluate similarity and is effectively a geometric average of the weighted proportions of n-grams (n-long sequences of characters) that appear in both the target and the hypothesis. Bidirectional BLEU is a novel algorithm developed by the authors of the original paper, and modifies the BLEU algorithm to use both the forward and backward hypotheses, weighting the forward hypothesis more for the n-grams closer to the beginning of the word, and the backward hypothesis more for the n-grams closer to the end. Each of these metrics is weighted and summed together to form a final similarity score between the hypotheses and every word in the dictionary, then the word with the largest score is returned as the final prediction.

3.2 Training

We split our dataset of 15,711 images into %90 training, %5 test, and %5 validation sets, and ran it over 20 epochs, with batch size equal to 32 data points. We took our model, converted the driver code to run in a notebook format, and ran it on Google Colab so as to utilize the GPU processing power on the cloud machines as opposed to our own, less powerful computers.

4 Results

The training took 10 hours, and achieves optimal validation loss at the 9th epoch as shown in Figure 3, 4. We observe some overfit at the end of the training, so the model at epoch 9 is kept for evaluation.

We evaluated the models with different word retrieval criteria in terms of accuracy, editorial distance, and BLEU distances. The distance scores are taken average among all testing data points. As shown in Table 1, the criteria taking equal weights on editorial distance and modified Levenshtein distance with forward and backward hypothesis performs uniformly better.

Table 1: Average metric results for some retrieval criteria on all hypothetical words. In the metrics, ”ed” stands for edit distance, ”acc” stands for accuracy; In the criteria, ”b” stands for sentence BLEU, ”ed” stands for editorial distance, and ”ml” stands for modified Levenshtein distance, with ”b” and ”f” at the end of each criteria referring to forward hypothesis and backward hypothesis, respectively. All results are truncated to 3 digits.

Retrieval criteria	acc	ed	BLEU-1 to BLEU-4
All equal	0.167	0.516	0.585, 0.451, 0.390, 0.346
bf	0.048	0.386	0.485, 0.339, 0.275, 0.231
bb	0.043	0.382	0.492, 0.338, 0.272, 0.227
edf	0.081	0.438	0.526, 0.377, 0.309, 0.263
edb	0.087	0.431	0.526, 0.374, 0.308, 0.263
mlf	0.078	0.437	0.516, 0.372, 0.307, 0.262
mlb	0.071	0.403	0.496, 0.354, 0.292, 0.249
0.5bf+0.5bb	0.201	0.539	0.599, 0.476, 0.415, 0.372
0.5edf+0.5edb	0.097	0.437	0.530, 0.387, 0.324, 0.280
0.5mlf+0.5mlb	0.179	0.520	0.601, 0.462, 0.399, 0.354
0.25(edf+edb+mlf+mlb)	**0.203**	**0.542**	**0.610**, **0.480**, **0.418**, **0.374**
0.35(edf+mlf)+0.15(edb+mlb)	0.176	0.512	0.585, 0.447, 0.384, 0.341
0.15(edf+mlf)+0.35(edb+mlb)	0.164	0.501	0.581, 0.444, 0.380, 0.335

Table 2 exemplifies the outputs of forward hypothesis and backward hypothesis of Gregg Shorthand of ”unnoticed”. We can see that the prediction is only reliable at the start with forward hypothesis, and in the end with backward hypothesis. When both predictions are taken into consider for retrieving the word, we can retrieve accurate prediction of the entire word. This example adds credit to our result that balanced weights in the word retrieval module produces better results.

”unnoticed”

Actual	AH N N OW T IH S T
Forward	AH N AH T T EH T
Backward	N N N IH S T
Combined	AH N N OW T IH S T

Table 2: Sample output of word ”unnoticed” in syllabic units

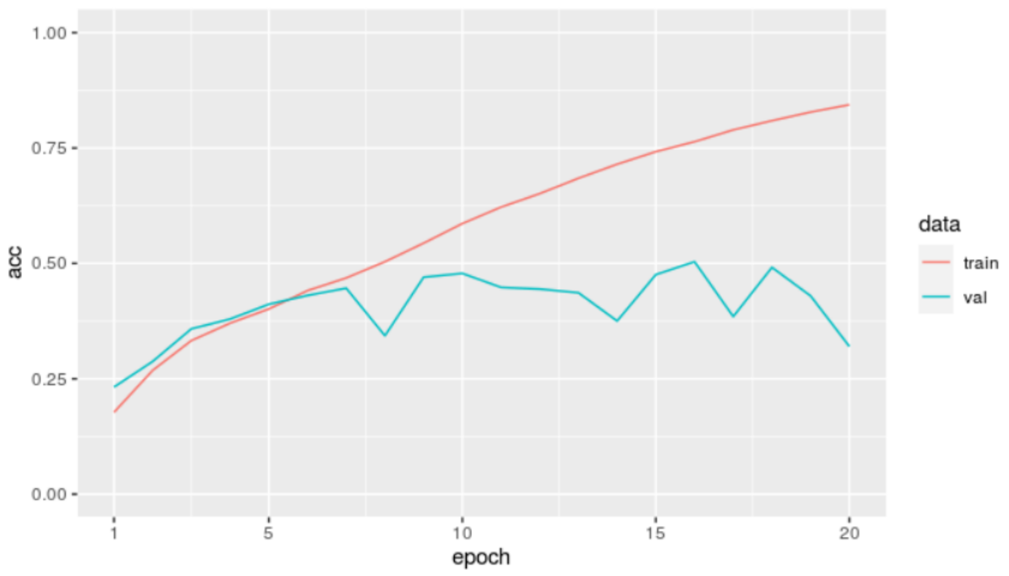

Figure 3: Accuracy of our model as a function of the number of epochs

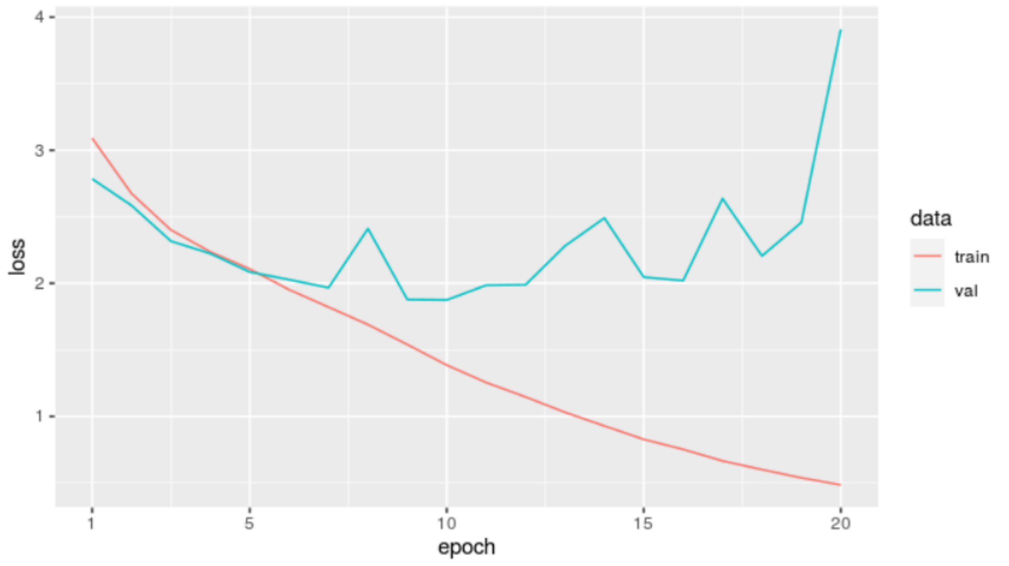

Figure 4: Loss as function of epochs

5 Conclusion and Discussion

If we were to only base our overall results on the accuracy of the model itself, it may seem as though the model does not perform all that well, especially with the less than 50% accuracy on the validation data that we see in the graphs. However, it is important to note that there are some flaws to Gregg shorthand that translate to inaccuracies in the model; because it is based on pronunciation, certain words might actually be very close in pronunciation but be different in spelling. We see many examples of this when we list out the hypothesized words versus the actual targets. For example, the words ”versed” and ”verse”, which are off by one letter, or the words ”fertile” and ”virtual”, which could be pronounced ”fer-tull” and ”ver-chull” and so have similar representations in Gregg. As such, since our model is predicated on how words are pronounced, similar words in Gregg can end up with the same hypothesized pronunciation and result in predicting the same word, even though they may be spelled differently. So, looking at some of the examples where the model goes wrong, we can actually see that the model is working as intended – it is correctly identifying the syllabic representation of a word, but the issue lies more in the inherent ambiguity of certain combinations of syllables.

Even though it’s slightly outside the initial scope of using image recognition, one way we could improve on transcribing Gregg shorthand to English is to actually process the surrounding sentence/paragraph/higher level language construct as a whole, as opposed to the individual Gregg symbols. This way, we can use semantic analysis on the predicted words to get better predictions of what was meant based on the context in which it was said. To accomplish this, we can modify the word retrieval module so that instead of scoring words on the distance metric to the hypotheses, we can actually consider the history of words we’ve seen before, and use a contextually sensitive method like a Markov analysis to factor in a score based on which words would statistically be more likely to appear at the current word’s position.

6 Reference

[1] Zhai F., Fan Y., Verma T., Sinha R. and Klakow D.: A Dataset and a Novel Neural Approach for Optical Gregg Shorthand Recognition. *International Conference on Text, Speech, and Dialogue.* Springer, Cham, 2018.

[2] The CMU Pronouncing Dictionary. http://www.speech.cs.cmu.edu/cgi-bin/cmudict

Individual contributions

Brian Chu

I had the original idea for the topic of this project, so I did most of the preliminary research into models that we could base our approach upon, and datasets that we could use to train our model on. Because Google Colab has limits on how much time users have to run notebooks, I helped Mason run the model on my account so that we could finish training the model when his account timed out. I also contributed to this final report, especially the introductory sections about our motivations and about Gregg shorthand, and to the final presentation.

Mason Sawtell

I was mostly responsible for taking the model architecture from the original paper and changing it to handle pronunciation. I also searched for appropriate ways to obtain pronunciation, and eventually settled on the CMU dictionary. Along with Brian, I trained the forwards and backwards model, setting up the Colab environment to do so. Finally, I handled the evaluation of the model, testing different combinations of evaluation parameters.

Yi Dai

Limited by network connection problems fro China, I did not involve in running the codes. I contributed more to providing research ideas and to this teport. More specifically, I provided the idea of using syllabic units for prediction and also improvement of the evaluation procedure by researching the properties of Gregg Shorthand. In this report, I mainly contributed to Results, part of Methods, and input and tidying up figures and tables.