Handwriting Sequence Generation from ASCII

Jason Chen
Department of Computer Science
Stanford University
chesson@stanford.edu

Abstract

This project explores the ability of recurrent neural networks to generate realistic appearing handwriting from ASCII text. We replicate the results of the Graves paper on synthesizing handwriting sequences from ASCII. The key components of Graves’ model include using deep long short term memory (LSTM) layers to capture long range dependencies in the points of handwritten text, a mixture density network (MDN) to capture the multimodal distribution of handwriting points, and an attention mechanism to focus on generating the output for a small window of one or two letters at a time. We are able to successfully reproduce the Graves’ results with model-generated outputs indistinguishable from human-generated handwriting when presented to human judges. We also dive deeper into showing exactly why the mixed density network and attention layers are so much more effective than the baseline of deep LSTMs.

1 Introduction

This project explores the ability of recurrent neural networks to generate realistic appearing handwriting from ASCII text. This is inspired primarily by the Graves paper Generating Sequences With Recurrent Neural Networks [3], and this project will attempt to replicate the results of the Graves paper. The input to the algorithm for training is a sequence of $n$ ASCII characters $\{c_1, c_2, c_3, ..., c_n\}$ and a sequence of $m$ points $[(x_1, y_1, \text{eos}_1), (x_2, y_2, \text{eos}_2), ..., (x_m, y_m, \text{eos}_m)]$ representing the points in the handwriting strokes corresponding to the ASCII input sequence, “eos”, short for end-of-stroke, is a boolean indicator representing a physical break between individual strokes where the “pen” is lifted. After training a deep network consisting of a stack of 3 LSTM layers, an attention mechanism, and a mixed density network, we feed just the ASCII character sequence and output a predicted sequence of points with the same $(x, y, \text{eos})$ tuple structure as training.

A more traditional and much more researched variation of this problem is to take handwritten text as input and output the corresponding ASCII text. This inverse problem of generating the handwriting strokes is more open-ended. There is no defined metric to evaluate the the generated output for correctness. However, our goal is for the output to be as realistic to human handwriting as possible. This project joins a whole body of other neural net generated artwork, music, literature, et cetera. One important application can be improving human-computer interaction by creating a medium of communication with more perceived personality. Another interesting application could be with historical regeneration of weathered or lost documents by seeding the algorithm with a particular style. A final application could be handwriting analysis when it comes to forgery or predicting what a person’s handwriting may look like.
2 Related Work

One of the most successful state-of-the-art implementations of this problem comes from the Graves paper [3]. Graves used a stack of LSTMs augmented by an attention layer and a MDN at the output. Each output point is used as the input to the next forward propagation sequence. Performance still degrades over longer pieces of writing so most results presented are lines under 700 points or approximately 30 characters. One of the more interesting approaches taken by Graves is "priming" the sequence by feeding the network a real sequence, then generating new extensions by continuing the stroke generation with the real sequence still "in-memory". The result are strokes that appear to take on the style of the primer sequence.

There is some existing work on deep neural network based speech synthesis done by Zen and Senior [1]. They cite limitations in existing approaches due to unimodal objective functions leading to a lack of ability to predict variances in their outputs. Their solution uses a mixture density output layer to improve the naturalness of the synthesized speech. The paper is a good parallel to evaluating and corroborating Graves’ choice to use an MDN in the handwriting generation component.

Stanford NLP researchers Luong, Pham, and Manning published a paper exploring attention-based neural machine translation [2]. The team has success with both a global and local attention mechanism that looks at either all or a subset of source words. To compare this to the Graves paper, the challenge is similar in that the length of the input and output sequences are very different, and Graves implements a differentiable soft window convolved with the ASCII input to learn an attentive alignment between the characters and the "pen" locations.

Lastly, we found two companies Bond and Handwrytten that offer services to generate custom notes and letters in personalized handwritten styles. They actually modified robotic printer setups to move a pen across paper. It is unclear whether their proprietary processes for learning each client’s handwriting styles involve manual labor by calligraphers parsing sample writing or if they involve some machine learned algorithm as we show in this paper.

3 Dataset and Features

The dataset used is the public IAM On-line Handwriting Database (IAM-OnDB) [5]. The segment of the database we are interested in contains samples of handwriting from 221 writers contributing 13,049 lines and 86,272 words from a dictionary size of 11,059. Specifically, the writers wrote pieces of text on E-Beam System electronic whiteboards and their pen locations were saved into XML files with a hierarchy of <StrokeSets>, <Strokes>, and <Points>. There are corresponding ASCII files for each of the stroke files. For this project, the raw data is parsed into a three dimensional time series where each point in the series contained point tuples of shape \((x_{coordinate}, y_{coordinate}, pen\_lifted)\). In this dataset, characters and lines consist of approximately 25 and 700 points on average, respectively.

We take the lines and break them down into a 90/10 train/dev split, or 11744 training and 1305 dev samples respectively. This would traditionally be bad practice due to the high possibility of overfitting on the training set. However, for the goal of generating realistic looking handwriting, overfitting is not a large issue.

Lastly, to narrow the scope of the problem we remove the long tail of rare characters by using only 57 out of 80 characters found in the dataset. Digits and most of the punctuation characters are grouped under a generic non-letter label.

![Figure 1: Handwriting sample](image1.jpg)

![Figure 2: XML raw data format](image2.jpg)
4 Methods

4.1 3 layer LSTM network

The backbone of this model is the LSTM network. We use a 3-layered LSTM network with a hidden state size of 400 each. LSTMs are a natural fit for this handwriting synthesis problem due to its affinity for sequential data and capturing long range dependencies. First, the points in handwriting are sequential data where previous points strongly influence where the next points should be. Furthermore, each character can be a long sequence of twenty to forty points. Lastly, handwriting contains delayed strokes like crossing t’s, which perform worse without LSTMs extended memory cells. Layering the LSTMs allowed the model to capture even more complex relationships in the data.

That said, a LSTM network on its own will perform quite poorly on this problem. LSTMs trained to minimize cross-entropy errors are designed to learn averages of target data conditioned on inputs [4]. However, this means they perform suboptimally when the relations are multimodal, where each input can have multiple correct target values. Intuitively, this is because "the average of several correct target values is not necessarily itself a correct value”.

4.2 Mixed density network and loss function

MDNs augment classical neural networks with a mixture density model and have shown to be useful when modeling multimodal distributions [1]. The mixture density model consists of a collection of distributions, and the final target probability is calculated by taking a weighted combination of each distribution.

For our MDN, we use $M$ mixture components consisting of bivariate Gaussian distributions for the $x$ and $y$ offsets and a Bernoulli distribution for the end-of-stroke indicator. The Gaussian distributions are defined by means $\mu_x$ and $\mu_y$, standard deviations $\sigma_x$ and $\sigma_y$, correlation $\rho$, mixture weight $\pi$. The Bernoulli distribution is defined by probability $p_{cos}$. This yields a total of $7M$ MDN parameters. We learn these parameters by setting them to be outputs from the final network layer.

\[
\begin{align*}
\hat{y}_t &= (\hat{e}_t, \{\hat{\rho}_t^j, \hat{\pi}_t^j, \hat{\sigma}_t^j\}_{j=1}^M) = b_y + \sum_{n=1}^N W_{hny} h_t^n + b_y \\
\hat{e}_t &= \frac{1}{1 + \exp(\hat{\epsilon}_t)} \\
\pi_t^j &= \exp(\hat{\beta}_t) \\
\mu_t^j &= \hat{\mu}_t^j \\
\sigma_t^j &= \exp(\hat{\sigma}_t^j) \\
\rho_t^j &= \tanh(\hat{\rho}_t^j)
\end{align*}
\]
The following equation gives the final probability density function (pdf) to predict the network output tuple \((x, y, \cos)\). Note that \(N\) expands to the pdf for a bivariate Gaussian distribution [6].

\[
Pr(x_{t+1} \mid y_t) = \sum_{j=1}^{M} \pi^j_t \mathcal{N}(x_{t+1} \mid \mu^j_t, \sigma^j_t, \rho^j_t) \begin{cases} e_t & \text{for } (x_{t+1})_3 = 1 \\
1-e_t & \text{otherwise} \end{cases}
\]

Since the mixed density model is also the final output layer of the network, we take the expected output during training and compare it to the predicted mixture to determine the sequence loss.

\[
\mathcal{L}(x) = -\log \left( \sum_{j} \pi^j_t \mathcal{N}(x_{t+1} \mid \mu^j_t, \sigma^j_t, \rho^j_t) \right) - \begin{cases} \log(e_t) & \text{for } (x_{t+1})_3 = 1 \\
\log(1-e_t) & \text{otherwise} \end{cases}
\]

### 4.3 Attention mechanism

Attention mechanisms in neural networks have been used extensively with success in domains like neural machine translation and image recognition [2]. Attention attempts to segment inputs to a model and provide a narrower window of “focused” content, similar to how human visual processing gives higher attention to certain regions. In this project, we build the attention layer to generate a sliding window over the letters being translated to handwriting. As longer lines of ASCII text can require up to 700 handwritten output points, giving the model an idea of several characters to focus on can greatly improve the quality of the output.

Specifically, given a length \(U\) character sequence \(c\) and a length \(T\) data sequence \(x\), the soft window \(w_t\) into \(c\) at time \(t\) is defined by the following \(K\) distributions [3]:

\[
\phi(t, u) = \sum_{k=1}^{K} \alpha^k_t \exp(-\beta^k_t (\kappa^k_t - u)^2) \\
w_t = \sum_{u=1}^{U} \phi(t, u) c_u
\]

\(w_t\) is a \(U\)-length vector representing the attention weight of each character. \(\phi(t, u)\) gives the window weight of one-hot vector \(c_u\) at time \(t\). \(\alpha\) weights the influence of the \(k\)th distribution, \(\beta\) controls the width of the window, and \(\kappa\) directs the location of the window within the sequence. We learn these \(3K\) parameters by setting them to be outputs from the first hidden layer of the network:

\[
(\hat{\alpha}_t, \hat{\beta}_t, \hat{\kappa}_t) = W_{h^1_{\text{att}}} h^1_t + b_{\text{att}} \\
\alpha_t = \exp(\hat{\alpha}_t) \\
\beta_t = \exp(\hat{\beta}_t) \\
\kappa_t = \kappa_{t-1} + \exp(\hat{\kappa}_t)
\]

### 5 Results and Discussion

#### 5.1 Handwriting synthesis

After training the model, we can synthesize new handwriting from ASCII by initializing a \((x_0, y_0, \cos_0)\) tuple and feeding it once through the network. Then, we feed the previous output as the input for the next iteration such that \(x_t = y_{t-1}\). We define a stopping heuristic that checks whether the attention mechanism’s window weight thinks the network is past the final character. Note that since this generative problem has no test metric, the training loss is relatively inconsequential.

(Train the model to synthesize new handwriting from ASCII by initializing an \(x, y, \cos\) sample and feeding it once through the network. Then we feed the previous output as the input for the next iteration. We define a stopping heuristic that checks whether)
5.2 Primed sampling

Priming the model to mimic a specific style is also possible. To do so, we start off with the stylized \( T \)-length input sequence \( x \) mapping to its ASCII sequence \( c \). For the first \( T \) steps, we feed the network \( x_t \) instead of the output from the previous timestep, effectively "clamping" the model to a specific style. Afterwards, the network will proceed as usual, feeding as inputs the outputs from the previous timestep.

Priming the model to mimic a specific style is also possible.
To do so, we start off with the stylized \( T \)-length input sequence

I do not know if I do not know

This line is the original style

5.3 Biasing MDNs

We can bias the model towards better handwriting by providing a bias term that reduces the standard deviations in the MDN Gaussian distributions. We are effectively giving preference to "better" handwriting closer to the mean of each distribution.

We can bias the model towards better handwriting
by providing a bias term that reduces the standard
deviations in the MDN Gaussian distributions. We are effectively
giving preference to better handwriting closer to the mean.

6 Conclusion and Future Work

We are fairly satisfied with being able to replicate the Graves paper with such success. With a some MDN bias, the output is very clean and legible. At the same time, the style priming worked well and was able to capture distinctive elements of each style.

One of biggest possible improvements for this project would come from increasing the size of the dataset. The dataset was published in 2005 and despite being frequently used it has not been updated since then. Today, we could easily create an app to record new handwriting samples on tablets, and I would make this the primary focus of any future work. It would give us more data to train with, as well as allow for greater interactivity by allowing anyone to contribute new samples and styles.

Another direction for future work would be applying the model towards speech synthesis. The problem domain is very similar to handwriting synthesis, but it would be potentially much more challenging due to the higher dimensionality of speech and audio data when compared to the points of handwriting data.

Lastly, we replicated all the training exactly according to the Graves paper, but we would like to do more hyperparameter searching to see if even better results are possible.

7 Contributions

This was a solo project, and I was responsible for doing the initial research, setting up and optimizing the Google Cloud environment, building and training the three models, and creating the poster and writing the final paper.

I consulted an existing implementation of the Graves paper [7] and reused boilerplate code to preprocess data, transform output into actual handwriting images, and handle saving and reloading checkpoints of the model during training.
References


