CS230

# Generating Realistic Handwriting With Deep Learning

Github Link

**Saumya Pathak**
Google
saumyap@stanford.edu
saumyap@google.com

**Shipra Banga**
Google
sbanga@stanford.edu
shiprab@google.com

## Abstract

In this project, we aim to generate human-like handwriting, given a dataset of human hand-writings in different styles. We use Recurrent Neural Networks composed of LSTM cells to learn complex patterns in hand-writing sequences, and further use Attention Windows to synthesize a given text in new generated hand-writings. We use MDNs to represent model outputs as parametrised mixture distributions.

## 1 Introduction

Human hand-writings vary extensively in style and form, and hence studying human hand-writings and being able to learn the long range structure present in them is a fairly challenging problem. This work has various real-life applications like modelling patterns, generating new fonts, writing realistic personalised letters, formulating digital signatures, etc. In order to preserve information over a long time duration, we employed LSTMs. The input to our algorithm is a point on the whiteboard, representing the pen location, with an extra signal to represent whether the point is an end-of-stroke point or not. A point is recorded as an end of stroke, when the writer lifted his/her pen from the whiteboard. We then use a composition of LSTM network, Attention Windows and Mixture Density Networks to output a predicted point on the whiteboard, with end of stroke probability. By predicting one single point at each step, our model is able to generate and write down an input text in human-like handwriting.

## 2 Related work

There has been significant work in the field of handwriting recognition and synthesis. Broadly they can be grouped into two categories:

- Online learning, which can go on online while user is writing.
- Offline learning which is done after the user is done writing the text

We modeled our approach, very similar to **Alex Graves'**[1] implementation, using LSTM units to predict next point given a dataset of hand-writing strokes, represented as a collection of points. We felt that the strengths of this approach, lied in the fact that it learnt handwriting as a pattern on a point-level, and hence could generalize well to any language, or any type of handwriting. There was also less hand-designing of components involved. The weakness was that RNNs are a relatively slow

CS230: Deep Learning, Winter 2019, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

learning algorithm and harder to train due to the problem of vanishing and exploding gradients.
Many computational models proposed an algorithm to relate velocity and force to the handwriting trajectory. A typical example is the delta log-normal model. **Li et al.**[2] presented an approach to represent the velocity of handwriting trajectory and encode the trajectory by a group of parameters. **Bezine et al.** [3] proposed a beta-elliptic model to estimate the correlation between geometry and kinematics in fast handwriting generation. These models prove to be successful for representing, compressing, and reconstructing captured handwriting, but not synthesizing novel handwriting

## 3 Dataset and Features

We used the IAM handwriting online dataset which contains forms of handwritten English text. The database was built using an E-beam system[4], where the collected data is represented by a set of strokes on a whiteboard, designated by collection of points. The strokes are identified by noting time when the writer lifted his/her pen.
The database[5] is structured as follows:

**Number of writers: 221**
**Isolated and labeled text lines: 13'049**
**86'272 word instances from a 11'059 word dictionary**

We processed the given xml data into a list of points on the whiteboard, denoted by three coordinates - x , y and eos - which is 1 if point is end of a stroke and 0 otherwise. Once our data is structured in this format, we divided this randomly into mini-batches with each sample containing points at t timesteps.
Each mini-batch: **[Number of samples][Timesteps][3]**
Number of mini-batches: **Data size / Number of samples in each mini-batch**
When selecting these collection of points from a line, we ignored lines smaller than Timesteps size. Also, we took offset of these point coordinates from the whiteboard corners, and normalized values so that they lied within a fixed range. We also built our validation data set, picking every 20th sample input point as a validation data sequence. Hence,

**Training data size: 6263 lines of tstep points**
**Validation dataset size: 328 lines of tstep points**

Few examples from our dataset are shown below:

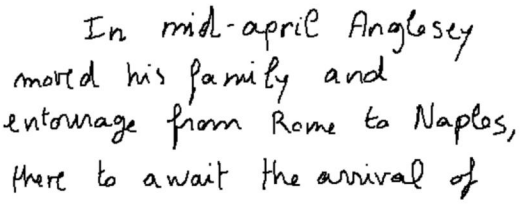


Figure 1: Examples from Dataset

## 4 Methods

### 4.1 Model

Our model could be divided into two sub-tasks:

- **Prediction Network** responsible for learning features from a pattern and predicting the best next point given a sequence
- **Synthesis Network** responsible for synthesizing input text to the model in the form of generated handwriting by predicting the next point at each step

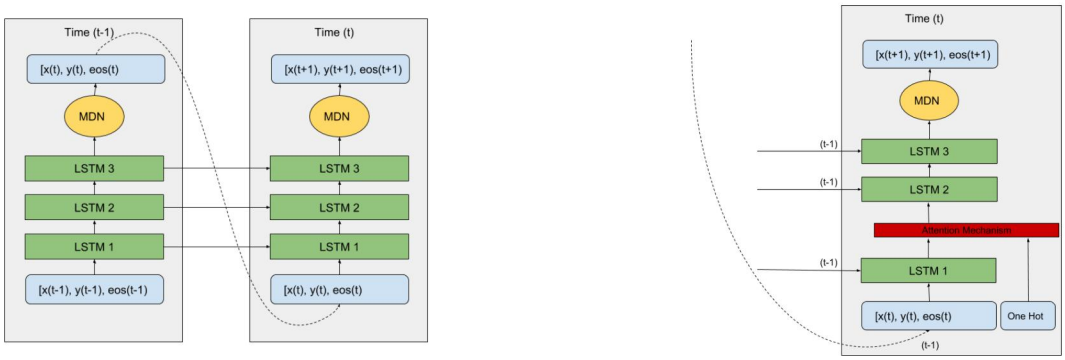


Figure 2: Prediction and synthesis architecture

#### 4.1.1 Prediction Network

Our model was composed of three LSTM cells

We tested different approaches for our model architectures starting with - 2 LSTM layers, 3 LSTM layers and 3 LSTM layers coupled with Mixture Density Networks[5]. We decided to use MDN in our architecture, since rather than trying to predict the expected value of a data point, it allows us to predict the entire probability density function of the data. This is much more useful when there is a lot of randomness in the data. We used Gaussian distributions to model x and y coordinates of the predicted point, and used Bernoulli distribution for modelling the eos probability. We used 20 mixtures, outputting 121 values, 6 parameters for each mixture (also including the weight for each mixture) and 1 value to indicate predicted eos value. So,
Given,

$$x \in R \times R \times (0, 1) \tag{1}$$

$$Output : (e_t, \{\pi_t^j, \mu_t^j, \sigma_t^j, \rho_t^j\}_1^M) \tag{2}$$

where: $e_t$ : end of stroke probability $\pi^j$ : Weight $\mu^j$: Mean $\sigma^j$: Variance $\rho^j$ : Correlation

Using these parameters, we predict our next point by taking a weighted sum of each of the mixtures with its end of stroke probability.

#### 4.1.2 Synthesis Network

Here, we used a 'soft window' which was convolved with input text string and fed in as an extra input to the prediction network, in order to condition the generated prediction on the input string. The parameters of window are output by the network during the process of prediction, where it dynamically aligns the window with the input text and the pen stroke points. The difference from the prediction network, as you can see above, are the extra character input from the input-sequence and extra window layer in the middle.

Given character sequence $c$ of length $U$, data sequence of length $T$, the soft window $w_t$ into c at timestep t $(1 <= t <= T)$ , is defined by the following discrete convolutions with a mixture of $K$ Gaussian functions.

$$\phi(t, u) = \sum_{k=1}^{K} \alpha_t^k \exp\left(-\beta_t^k (\kappa_t^k - u)^2\right) \tag{3}$$

$$w_t = \sum_{u=1}^{U} \phi(t, u) c_u \tag{4}$$

where $\phi(t, u)$ is the window weight of $c_u$ at timestep t. Here, drawing from intuition, $\kappa_t$ influences location of the window, $\beta_t$ influences width of the window and $\alpha_t$ influences importance of the window.

These window parameters : $(\alpha, \beta, \kappa)$ are determined by the output of the first layer of the hidden neural network as follows:

$$(\hat{\alpha_t}, \hat{\beta}_t, \hat{\kappa_t}) = W_{h^1 p} h_t^1 + b_p \tag{5}$$
$$\alpha_t = \exp(\hat{\alpha_t}) \tag{6}$$
$$\beta_t = \exp(\hat{\beta}_t) \tag{7}$$
$$\kappa_t = \kappa_{t-1} + \exp(\hat{\kappa_t}) \tag{8}$$

The $w_t$ vector is then passed to the second and third hidden layer at time t, and to the first layer at time t+1.

### 4.2 Training Loss

While training, we selected our loss function from what Alex Graves[1] implemented on x and y, which was :

$$\mathcal{L}(\mathbf{x}) = \sum_{t=1}^{T} -\log\left(\sum_j \pi_t^j \mathcal{N}(x_{t+1}|\mu_t^j, \sigma_t^j, \rho_t^j)\right) - \begin{cases}\log e_t & \text{if}(x_{t+1})_3 = 1 \\ \log(1-e_t) & \text{otherwise}\end{cases} \tag{9}$$

This loss equation determines the sequence loss for the entire sequence, where the predicted point is a function of both $x_{1:t}$ and c (input character sequence).

## 5 Experiments/Results/Discussion

**Learning rate**
We tried our algorithm with 0.01, 0.001 and 0.0001 learning rate, and figured out that our algorithm ran best with a learning rate of 0.001. We also applied exponential rate decay and gradient clipping, to avoid problem of exploding gradients at later epochs in learning.

**Mini-batch size**
We changes our mini-batch size in powers of 2 ranging from 32, 64 to 128. When we were training our model on a small subset of our dataset in order to overfit our model, we kept mini-batch size as 32, and increased it 128 as we generalised our model and trained it on all the data.

**Optimisation**
We experimented with both Adam Optimiser and RMSProp, and stuck with RMSProp since it gave us better results. We also initialized our weights and bias for window layer and mixture density layer randomly, and used initialization inspired from Alex Graves' work, to get best results.

**Primary Metrics**
We verified the performance of our model from training loss and validation loss. We did not have traditional metrics in this problem, since the problem was to generate a new handwriting each time, and there was no set quantitative result we could base our performance metric on. Apart from monitoring the loss, we measure our model's performance on the results its achieved in writing down new hand-writings.
In the first example mentioned in the table below, we overfit to our training data set, with less regularisation and also kept a smaller dataset. In later iterations, after we achieved a good training loss, we expanded our dataset and increased the regularisation to achieve best results.

We also decreased the number of hidden layers to 2 layers, and did not notice any major differences in loss achieved and results, however the training happened faster and was less computationally expensive.

| Batch Size | #Minibatches | Dropout Rate | #Hidden Layers | Train Loss | Valid Loss |
|---|---|---|---|---|---|
| 32 | 15 | 0.98 | 3 | -4.23 | -2.36 |
| 128 | 48 | 0.95 | 3 | -2.56 | -.2.34 |
| 128 | 48 | 0.95 | 2 | -2.51 | -.2.28 |

Table 1: Train Loss & Validation Loss to measure performance of different models

### 5.1 Results

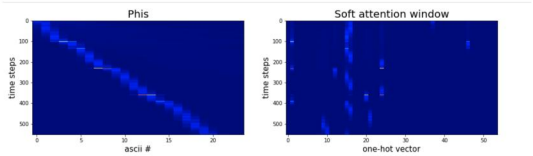


Figure 3: Each point on the map shows the value of $\phi(t, u)$, where t indexes the pen trace along the vertical axis and u indexes the text character along the horizontal axis. The bright line is the alignment chosen by the network between the characters and the writing.


Figure 4: Set of generated images

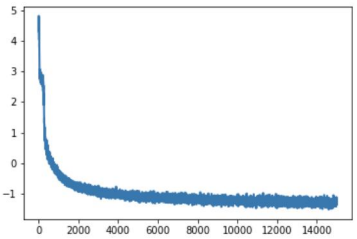


Figure 5: Decreasing loss with number of epochs with mini-batches

## 6 Conclusion/Future Work

- Train the model to generate text from any language, even symbols.
- Improve the model to perform better for deep-in-time scenarios. Example: generating a very long sentence.
- Tackle the exploding gradient problem better
- RNN with LSTM can learn and generate complex long-range structures using next-step prediction.
- Interestingly, we observed the results didn't change much even after using 2 LSTM layers instead of 3, which reduced the number of variables greatly
- The model shows potential to generate sensible next characters if it is forced to generate beyond the specified sentence.

## 7 Contributions

Saumya Pathak worked on building the LSTM model with MDN layer and building the sampling process. Shipra Banga worked on the dataprocessor layer and adding attention window mechanism to it for synthesis. Both of us collaborated on iterative training of models with different set of runtime arguments and compared results with each other to formulate the best designed model. In order to learn mixture networks, LSTMs and attention windows, we split our areas of study, studied individually and gave small tutorial sessions to each other to cover a good understanding on each of these topics.

## References


[1] Alex Graves. Generating Sequences with Recurrent Neural Networks. ArXiv 2013 arXiv:1308.0850.

[2] X. Li, M. Parizeau, and R. Plamondon, Segmentation and reconstruction of on-line handwritten scripts, Pattern Recogtion 31(6) (1998) 675-684.

[3] H. Bezine, A.M. Alimi, and N. Derbel, Handwriting trajectory movements controlled by a beta-elliptical model, Proc. Seventh Int'l Conf. Document Analysis and Recognition, Edinburgh, Scotland (2003) 1228-1232.

[4] http://www.e-beam.com/education/ebeam-edge/overview.html.

[5] http://www.fki.inf.unibe.ch/databases/iam-handwriting-database.

[6] Realistic handwriting with tensor flow https://github.com/greydanus/scribe.

[7] https://www.youtube.com/watch?v=Rkl30Fr2S38.

[8] https://mikedusenberry.com/mixture-density-networks.

[9] Coursera videos on RNNs.