Handwriting Synthesis Using Long Short-Term Memory

Contributed by

Manogna Mantripragada

and

Ninad Subhedar

Abstract

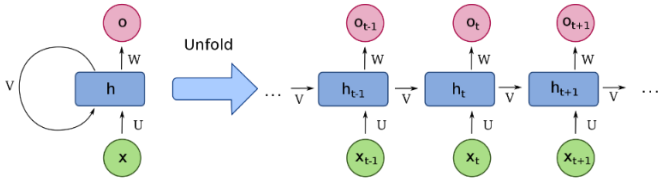
The goal is to explore deep learning algorithms for synthesizing sequence of characters that mimic handwriting. To achieve this goal, we would ingest a data set containing unique handwriting styles for training the model.

The data is a complex three-dimensional time-series data with the first two dimensions as the x and y coordinates and the third dimension is the end of stroke. There have been several models which train handwriting to convert a list of pen points to digital text. We would be training the model to do the other way where we ingest digital text and convert it into a series of hand-written points

The work is based on a paper from the 2014 on Generating Sequences from a Recurrent Neural Networks by Alex Graves. The paper shows how Long Short-term Memory recurrent neural networks can be used to generate complex sequences with long-range structure, simply by predicting one data point at a time based on its surrounding pattern.

Introduction

What is a Recurrent Neural Network?



Recurrent Neural Networks are a rich class of dynamic model and connectionists with the ability to selectively pass information across sequence steps, while processing sequential data one element at a time. They can be trained for the sequence generation by processing real data sequences one step at a time and predicting what comes next.

Assuming the predictions are probabilistic, novel sequences can be generated from a trained network by iteratively sampling from the network's output distribution, then feeding in the sample as input at the next step.

In simple terms, Recurrent Neural Networks are a type of Neural networks that take in its own predicted output as an input to predict the next output. This model can be trained for any sequence generation by processing real data sequences one step at a time and predicting what comes next. The above figure describes the high-level design of an RNN. Here the model h takes in the input x and outputs o. The output is taken and feed back into the model. The unfolded version shows the same neural network at different time steps.

But we have noticed that standard vanilla RNNs cannot store information about past inputs in their memory for a long period of time. Also, they aren't very efficient when modelling long range structures. This 'amnesia' makes them prone to instability when generating sequences.

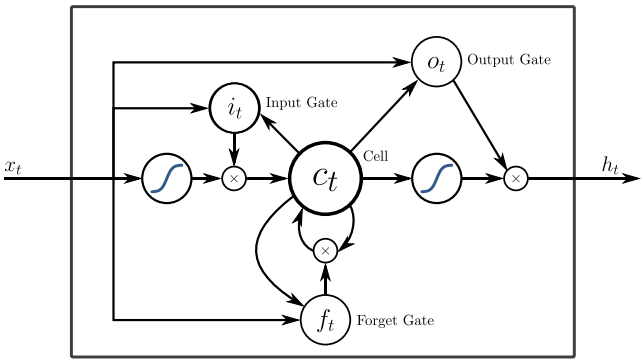
The problem with almost all generative models is that if the network's prediction works an input for generating the next prediction, there is a high probability that the errors and mistakes from the previous model get carry forwarded to the next model as well.

In situations like these, having a longer memory is beneficial as it has a stabilizing effect because even if it is difficult for the network to make sense depending on the recent history, it can trace back to the past and formulate its predictions more accurately

Need for a Long Short-Term Memory

Since RNNs could not effectively retain the information for a longer period of time, there needed to be an alternative model that could help store information for long. LSTM stands for Long Short-Term Memory and are a particular type of recurrent Neural Network which have internal contextual state cells that act as long-term or short-term memory cells and thus are better at storing and accessing information than standard RNN cells. We aim to use the memory capabilities of LSTM to synthesize more accurate sequence of characters.

Working of Long Short-Term Memory



LSTM can store memory internally and access it whenever it wants unlike the RNN.

The above is a network with 3 gates. These gates act as differentiable versions of read, write and reset operation and are the ways to optionally let information through.

They are composed of a sigmoid neural net layer and a pointwise multiplication operator. The sigmoid layer outputs between 0 and 1 describing how much of each component must be let through where 0 being nothing is to be let through and 1 being let everything through.

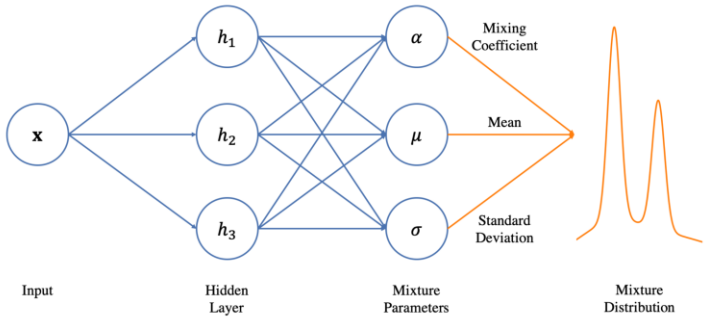
Forget gate: First we define how much information is needed to be kept in and how much to throw away. 1 being keep everything and 0 being completely get rid of this.

Input Gate: The input gate takes values between 0 and 1 and when the gate is 0, no data is passed. This is where the memory is stored in the subnetwork.

Output gate: Multiplies the output to the rest of the system. If the output gate is closed, no one can read

Mixture Density Networks

The mixture density network is used to capture randomness in the data. At its very core, the MDN concept is simple, straightforward, and appealing: **Combine a deep neural network (DNN) and a mixture of distributions.** The idea of mixture density networks is to use the outputs of a neural network to parameterize a mixture distribution. A subset of the outputs is used to determine the mixture weights, while the remaining outputs are used to parameterize the individual mixture components.



Each input vector x_t consists of a real-valued pair x_1; x_2 that defines the pen offset from the previous input along with a binary x_3 that has a value of 1 if the vector ends a stroke (that is, if the pen was lifted off the board before the next vector was recorded) and value 0 otherwise. A mixture of bivariate Gaussians was used to predict x_1 and x_2, while a Bernoulli distribution was used for x_3. Each output vector y_t therefore consists of the end of stroke probability e, along with a set of means μ^j, standard deviations σ^j, correlations ρ^j and mixture weights π^j for the M mixture components. That is:

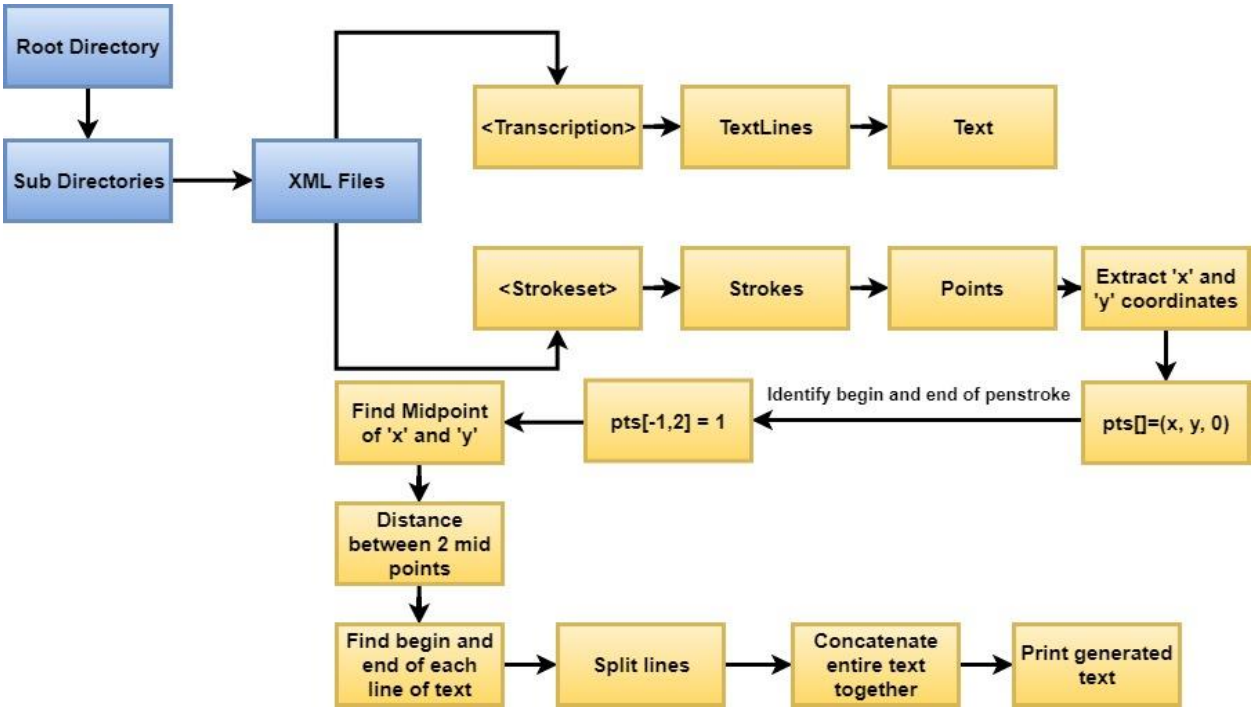
$$x_t \in R \times R \times \{0,1\}$$
$$y_t = (e_t, \{\pi_{t,}^j, \mu_t^j, \sigma_t^j, \rho_t^j\}_{j=1}^M)$$

About the Data

The data is taken from the IAM online handwriting database which contains the handwritten English text written on the whiteboard. The data contains forms of unconstrained handwritten text, acquired with the E-Beam System. It's a dataset with a lot of different writers and was recorded by the people writing on the whiteboard with a pen with an infrared sensor and the sensor recorded the pen's coordinates. The collected data is stored in xml-format, including the writer-id, the transcription and the setting of the recording. The data has multiple 'transcription' and 'strokes' tags. The Transcription tags contain the text and the Strokes tag contains the co-ordinates of each point in a single stroke.

Design and Implementation Approach

Data Preprocessing



Pre-processing:

Step 1: For pre-processing the data we extract the text and the co-ordinates from the xml file.

Step 2: Identify the end of stroke by observing the end of stroke value. If It is 1, then the pen is lifted up (end of stroke) and if its 0 it is the begin of stroke.

Step 3: Find the mid-point of each stroke

Step 4: Find the Distance between the mid points of two consecutive strokes

Step 5: Find the begin and end for each line of text

Step 6: Split the lines and concatenate the entire text together

Input Data Format

The data is stored in a root directory with multiple subdirectories and each subdirectory containing xml files.

The xml files have two main tags, the transcription tag and the StrokeSet tag.

Transcription Tags: We get text lines and text from the transcription tags

Strokeset tag: The strokeset tag contains all the x and y coordinates and the pen-stroke value which lies between 0 and 1.

Extracting Data

Text: Extracting the Text from the TextLine which is taken from the Transcription tag

<**TextLine** id="a01-001z-01" **text="By Trevor Williams. A move**">

Co-ordinates: Extract co-ordinates from the Strokeset

<StrokeSet>

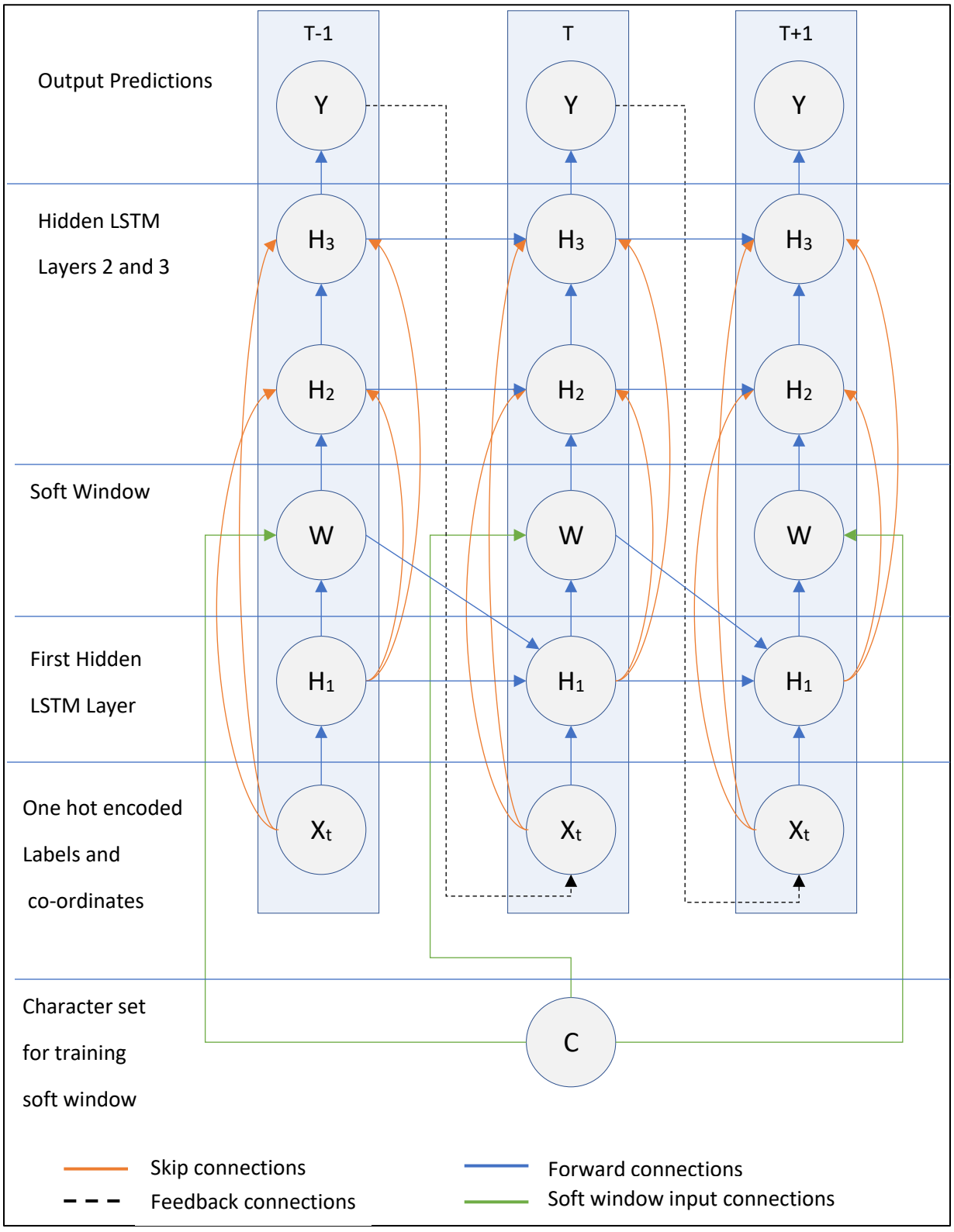
<**Stroke** colour="black" start_time="13090871.35" end_time="13090871.78">

<**Point x="895" y="992"** time="13090871.35"/>

Normalizing Data

We need to normalize the data to bring the co-ordinates into a specific range. This will help us plot our text in a concise way. For this we find the difference between the values and the mean and divide this difference with the standard deviation so as to get values ranging from 0 and 1

Prediction Model



Soft Window Layer

The processed data is feed into a soft window to guide the neural network by representing a relation between the handwriting strokes and the text characters. The relation is defined by the below equation:

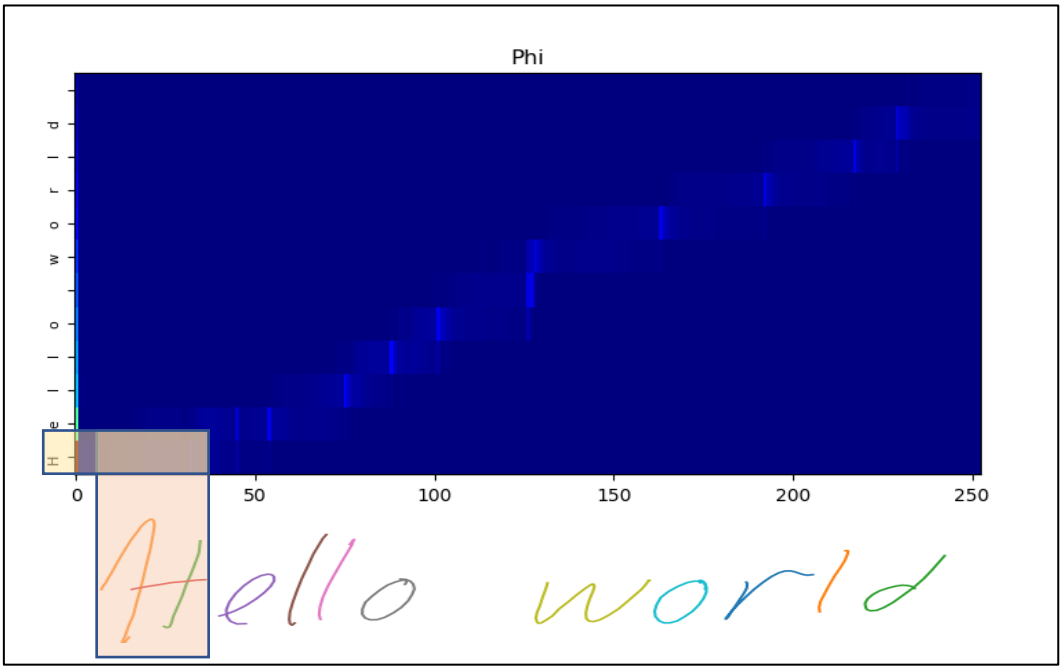
$$window = \sum_{u=1}^{U}\left(\sum_{k=1}^{K}\alpha_t^k e^{\left(-\beta_t^k\left(\kappa_t^k-u\right)^2\right)}\right)c_u$$

$$\alpha_t = e^{\widehat{\alpha_t}} \rightarrow Importance\ of\ the\ window$$

$$\beta_t = e^{\widehat{\beta_t}} \rightarrow Width\ Of\ the\ window$$

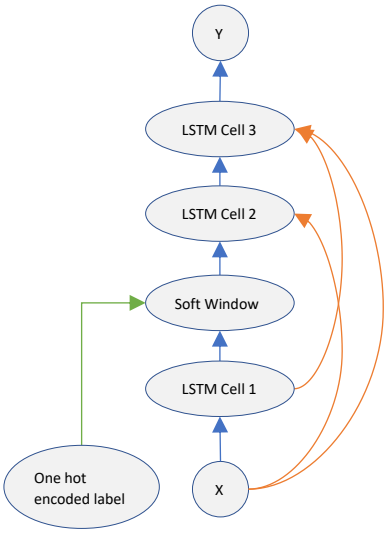
$$\kappa_t = \kappa_{t-1} - e^{\widehat{\kappa_t}} \rightarrow offset\ for\ the\ window$$

Below figure illustrates how the window co-relates the text to the set of strokes. The α, β, κ values are forwarded to

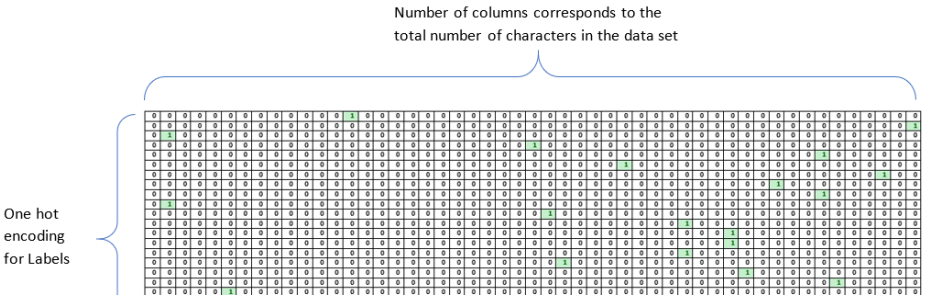


LSTM Layer

The main LSTM layer was designed using the standard LSTM Cell in TensorFlow library. Three Hidden LSTM Cell each consisting of 400 Cell were stacked to form the LSTM layer. The input to these layers comes from the coordinates of the penpoint and its corresponding labels along with the output of the soft window. Every layer has skip connections from the inputs to all hidden layers and from all hidden layers to the outputs. This makes it easier to train the networks by reducing the number of processing. This also helps in handling the `vanishing gradient' problem. The below figure shows the high-level design of the LSTM layer. The connections marked in orange are the skip connections. And the connection marked green is the input from the one hot encoded text.



The text data had to be one hot encoded into a matrix such that each sentence has its own matrix and the rows signify a single character while the column indicates which character is to be selected. The following figure shows a representation of the one hot encoded matrix.



Mixture Density Layer

The output from the LSTM layer is forwarded to a Mixture Density Layer which converts the output to a mixture of distributions which is used to create a probability distribution that represents the coordinates for the pen point and the end strokes. The mixture consists of a bivariate Gaussian which can be used to predict the coordinates and a Bernoulli distribution used to predict the end stroke probability. This layer generates the following elements which are used to generate the above-mentioned distributions.

$$e_t = \frac{1}{(1 + \exp(\hat{e}_t))} \quad \rightarrow \quad End\ stroke\ probablity$$

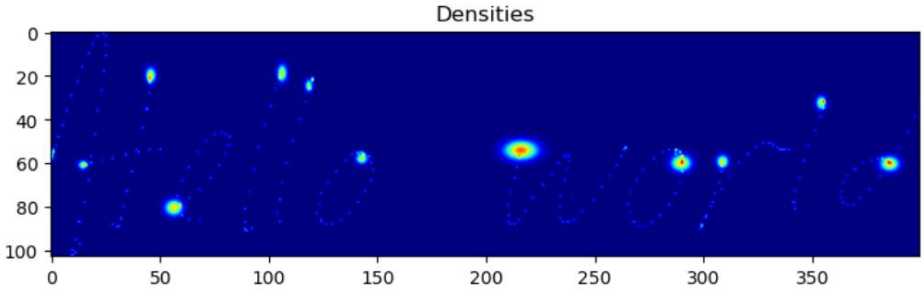
$$\pi_t = \frac{\exp(\hat{\pi}_t)}{\sum_{j=1}^{M} \exp(\hat{\pi}_t)} \quad \rightarrow \quad Mixute\ weights$$

$$\mu_t = \hat{\mu}_t \quad \rightarrow \quad Set\ of\ means$$

$$\sigma_t = exp(\hat{\sigma}_t) \quad \rightarrow \quad Set\ of\ Standard\ deviations$$

$$\rho_t = tanh(\rho_t) \quad \rightarrow \quad Co - relations\ between\ the\ co - ordinates$$

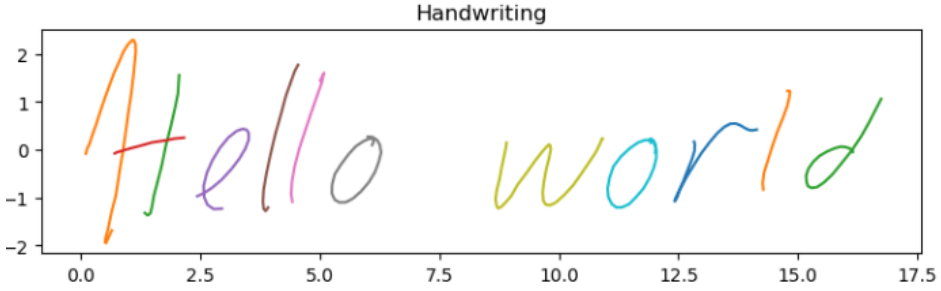
These variables are used to generate probability distributions and then pull random samples from the distributions to get the predicted co-ordinates and the predicted end strokes. The following figure shows the mixture distributions for each coordinate.



Handwriting generation

Prediction and Loss function

The output of the Mixture density layer can be sampled, and the coordinates can be generated along with the end of strokes for the text. Each co-ordinate can be plotted on a graph to represent a stroke. To generate the coordinates the trained model is feed with one hot encoded text and then the model outputs the probability densities from which random samples can be taken to generate a set of coordinates. Below is an example of the coordinates generated from the model and then plotted onto a graph. The colors in the graph represent individual strokes.



The probability density function was defined as follows:

$$\Pr(x_{t+1}|y_t) = \sum_{j=1}^{M} \pi_t^j \, \mathcal{N}(x_{t+1}|\mu_t^j, \sigma_t^j, \rho_t^j) \begin{cases} e_t & \text{if } (x_{t+1})_3 = 1 \\ 1 - e_t & \text{otherwise} \end{cases}$$

where

$$\mathcal{N}(x|\mu, \sigma, \rho) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \exp\left[\frac{-Z}{2(1-\rho^2)}\right]$$

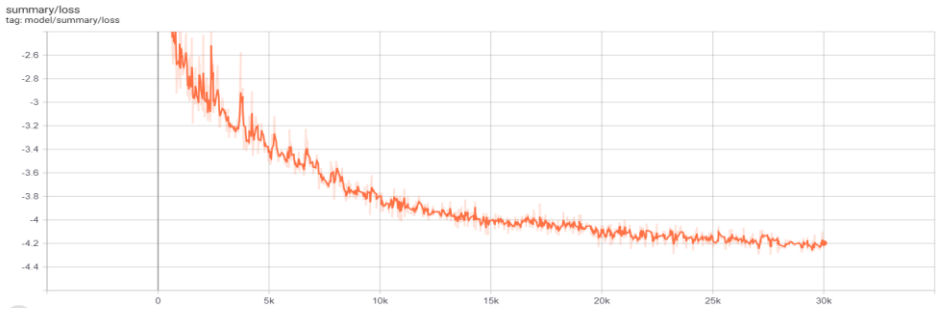
with

$$Z = \frac{(x_1 - \mu_1)^2}{\sigma_1^2} + \frac{(x_2 - \mu_2)^2}{\sigma_2^2} - \frac{2\rho(x_1 - \mu_1)(x_2 - \mu_2)}{\sigma_1\sigma_2}$$

All the components derived from the Mixture density layer are used to calculate the probability density function. And the loss function is a negative log loss defined as below

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

This function was used for optimizing the learning process for the neural network. The optimizer used for training was Adam Optimizer and the learning rate was set to 0.001. The complete model was trained for 30 epochs and each epoch had 1000 batches. Below graph shows the loss values for each step. The loss decreased from -2.431 to -4.197



Observations & Conclusion

The model learns from the handwriting data provided and does generate handwriting co-ordinates with individual strokes. For long sequences of text, the model does not tend to maintain the angle of handwriting. For example, in the below figure the text starts to bend downwards.

This could possibly be due to a smaller number of long sentences and due to this model is not trained to learn long continuous handwriting sequences. Furthermore, the model outputs random styles of handwriting while generation. This could be solved by priming the model just before generation which is not covered in this scope. The model works well with block letters as compared to cursive handwriting.

System Challenges

To run this model, we started with a simple Virtual Machine on Google Cloud Platform without any GPU which took us 3 days to run the model. We them moved to using the Google Cloud Platform's Jupyter Notebooks from their Machine Learning Library.

We had to first select the machine type with specifying the cores of the CPU and the RAM. We then had to choose what kind of GPU we needed to run the model efficiently. We first started with a Nvidia Tesla T4, which still took time to process, so we upgraded it to Nvidia Tesla P100 and also tried Nvidia Tesla V100 and found not much difference between the P100 and V100, hence we used Nvidia Tesla P100.

Contribution

The 60% of the design and implementation was derived from the original paper Generating Sequences from a Recurrent Neural Networks by Alex Graves [1] and handwriting generation with the use of recurrent neural networks in TensorFlow [7]. Remaining 40 % includes modification to the code to write individual layers and introducing a 3 layered LSTM with an Adam Optimizer for training the neural network.

Citations

1. Generating Sequences with Recurrent Neural Networks (https://arxiv.org/abs/1308.0850)
2. Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM) (https://youtu.be/WCUNPb-5EYI)
3. Long short-term memory (https://en.wikipedia.org/wiki/Long_short-term_memory)
4. IAM On-Line Handwriting Database (http://www.fki.inf.unibe.ch/databases/iam-on-line-handwriting-database)
5. MIT 6.S191: Recurrent Neural Networks (https://youtu.be/_h66BW-xNgk?list=PLtBw6njQRU-rwp5__7C0oIVt26ZgjG9NI)
6. Scribe: realistic handwriting with TensorFlow (https://greydanus.github.io/2016/08/21/handwriting/)
7. handwriting generation with use of recurrent neural networks in tensorflow (https://github.com/Grzego/handwriting-generation)

8. Deep Learning Lecture 13: Alex Graves on Hallucination with RNNs (https://www.youtube.com/watch?v=-yX1SYeDHbg&t=2217s)
9. The magic of LSTM Neural networks (https://towardsdatascience.com/learn-how-recurrent-neural-networks-work-84e975feaaf7)
10. How Recurrent Networks Work (https://towardsdatascience.com/learn-how-recurrent-neural-networks-work-84e975feaaf7)

License