Handwriting generation using recurrent neural networks (LSTM)

Rabpreet Singh Keer

Netaji Subhas Institute of Technology

***Abstract*- Handwriting is a skill developed by humans from a very early stage in order to represent his/her thoughts visually using letters and making meaningful words and sentences. Every person improves this skill by practicing and developing his/her own style of writing. Because of the distinctiveness of handwriting style, it is frequently used as a measure to identify a forgery.**

Even though the applications of synthesizing handwriting is less, this problem can be generalized and can be functionally applied to other more practical problems. Mimicking or imitating a specific handwriting style can have an extensive variety of applications like generating personalized handwritten documents, editing a handwritten document by using the similar handwriting style and also it is extended to compare handwriting styles to identify a forgery.

All the training and test data is taken from IAM online handwriting database (IAMOnDB). IAM-OnDB consists of handwritten lines of data gathered from 223 various writers using an e-smart whiteboard.

Introduction

It is important to digitize handwritten documents for efficient processing and storage. This problem is well known as Intelligent Word Recognition (IWR) and has been an area of study for decades. Our work presents an effective method to not only recognize the contents of a handwritten document, but also to generate handwritten documents from typed inputs with writing characteristics that are specific to an author's writing style. Users can train our model by providing their handwritten documents as inputs and subsequently use this model to generate documents written in their handwriting font from text inputs.

Cursive handwriting is a complex graphic realization of natural human communication. Its production and recognition involve a large number of highly cognitive functions including vision, motor control, and natural language understanding. Handwriting synthesis has many important applications to facilitate user's work and personalize the communication on pen-based devices.
Many works have been written on the generation of handwritten characters. Generative techniques can be divided into two categories: movement simulation techniques and shape simulation methods. Movement simulation techniques are mostly based on motor models whereas shape simulation techniques use the trajectories of the handwriting. Movement simulation techniques are mostly derived from the kinematic theory of human movements. However, modeling handwriting with movement simulation techniques implies a dynamic-inverse problem which is difficult to solve.
In contrast, shape simulation techniques concentrate on the trajectories of the handwritten strokes, which already embody the characteristics of an individual's personal writing style directly in their shape.

Motivation

Humans learn writing by practicing repeatedly to study the strokes. The way we learn handwriting is almost similar to any task we learn. We learn things by repeating it until it finally becomes involuntary. Even with all these Deep Learning algorithms, computers still don't know how to learn a task. And moreover, humans deal with both historical and spatial information which is also difficult for computers to handle.

Trying to solve some simple problems like handwriting may lead us to a better understanding of how humans think and can develop better algorithms for computers. This model will even help us learn to imitate the particular style of handwriting.

Problem Statement

Our aim is to generate handwritten text from typed inputs with writing characteristics that are specific to an author's writing style. Further which user trains the model by providing their handwritten texts as inputs and subsequently use the model to generate document (an image) written in their handwriting font from text inputs.

The project focuses on synthesising English Handwritten Text from ASCII Trancriptions entered by a user. Recurrent Neural Networks using LSTM (Long Short Term Memory) cells has been used to achieve the said goal.

Input to the system : Stream of ASCII character entered by a user.
Output to the system: Synthesized handwritten text of the character stream inputted by the user.

Neural Network

A *neural network* (NN), in the case of artificial neurons called *artificial neural network* (ANN) or *simulated neural network* (SNN), is an interconnected group of natural or artificial neurons that uses a mathematical or computational model for information processing based on a connectionist approach to computation. In most cases an ANN is an adaptive system that changes its structure based on external or internal information that flows through the network.

Recurrent Neural Network

The Recurrent Neural Network works on the principle of saving the output of a layer and feeding this back to the input to help in predicting the outcome of the layer.

Here, the first layer is formed similar to the feed forward neural network with the product of the sum of the weights and the features. The recurrent neural network process starts once this is computed, this means that from one time step to the next each neuron will remember some information it had in the previous time-step. This makes each neuron act like a memory cell in performing computations. In this process, we need to let the neural network to work on the front propagation and remember what information it needs for later use. Here, if the prediction is wrong we use the learning rate or error correction to make small changes so that it will gradually work towards making the right prediction during the back propagation.

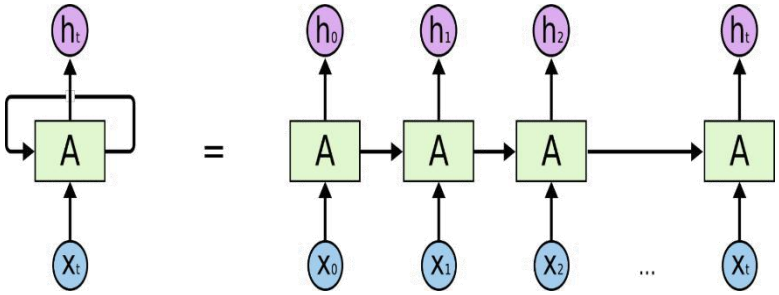

Figure 1.1: Recurrent Neural Network

Backpropagation through Time (BPTT)

Recurrent networks rely on an extension of backpropagation called Backpropagation through Time, or BPTT. Time, in this case, is simply expressed by a well-defined, ordered series of calculations linking one time step to the next, which all backpropagation needs to work.

The goal of the backpropagation training algorithm is to modify the weights of a neural network in order to minimize the error of the network outputs compared to some expected output in response to corresponding inputs. It is a supervised learning algorithm that allows the network to be corrected with regard to the specific errors made

The general algorithm is as follows:

1. Present a training input pattern and propagate it through the network to get an output.
2. Compare the predicted outputs to the expected outputs and calculate the error.
3. Calculate the derivatives of the error with respect to the network weights.
4. Adjust the weights to minimize the error.
5. Repeat.

Backpropagation through Time, or BPTT, is the application of the Backpropagation training algorithm to recurrent neural networks applied to sequence data like a time series.

A recurrent neural network is shown one input each timestep and predicts one output. Conceptually, BPTT works by unrolling all input time steps. Each timestep has one input timestep, one copy of the network, and one output. Errors are then calculated and accumulated for each timestep. The network is rolled back up and the weights are updated.

Spatially, each time step of the unrolled recurrent neural network may be seen as an additional layer given the order dependence of the problem and the internal state from the previous time step is taken as an input on the subsequent time step.

We can summarize the algorithm as follows:

1. Present a sequence of time steps of input and output pairs to the network.
2. Unroll the network then calculate and accumulate errors across each time step.
3. Roll-up the network and update weights.
4. Repeat

Vanishing and Exploding Gradient Problems

Vanishing Gradient: Vanishing Gradient Problem is a difficulty found in training certain Artificial Neural Networks. This problem becomes worse as the number of layers in the architecture increases.

This mainly occurs when the network parameters and hyper parameters are not properly set. Parameters could be weights and biases while hyper parameters could be learning rate, the number of epochs, the number of batches, etc.

Vanishing gradient problem depends on the choice of the activation function. Many common activation functions (e.g sigmoid or tanh) 'squash' their input into a very small output range in a very non-linear fashion. For example, sigmoid maps the real number line onto a "small" range of [0, 1]. As a result, there are large regions of the input space which are mapped to an extremely small range. In these regions of the input space, even a large change in the input will produce a small change in the output - hence the gradient is small.

This becomes much worse when we stack multiple layers of such non-linearities on top of each other. For instance, first layer will map a large input region to a smaller output region, which will be mapped to an even smaller region by the second layer, which will be mapped to an even smaller region by the third layer and so on. As a result, even a large change in the parameters of the first layer doesn't change the output much.

Due to Vanishing Gradient, y slope becomes too small and decreases gradually to a very small value (sometimes negative). The model may take longer to train and learn from the data and sometimes may not train at all and show error. This results in less or no convergence of the neural network.

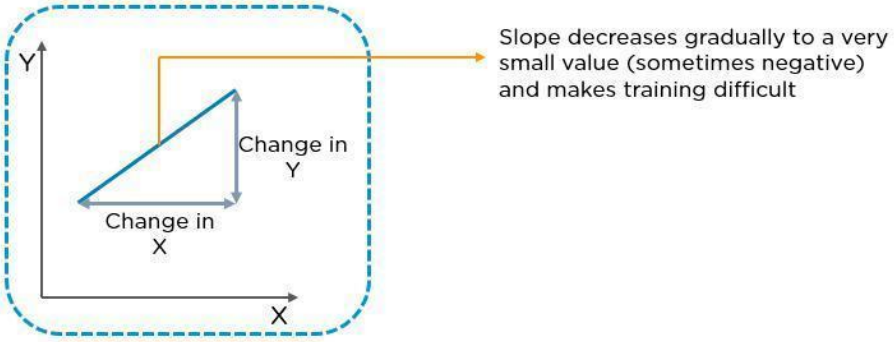

Figure 1.2: Understanding Gradient Problem

This leads to poor performance of the model and the accuracy is very low. The model may fail to predict or classify what it is supposed to do.

Solutions to Vanishing Gradient problem

LSTMs: Long Short Term Memory Networks can solve Vanishing Gradient problems when working on RNN. LTSMs help to solve long term dependencies and can memorize previous data easily.

Faster Hardware: Switching from CPUs to GPU's with faster compilation time have made standard backpropagation method feasible where the cost of the model is very less.

Other activation functions: Rectifiers such as ReLU suffer less from Vanishing Gradient problem, because they only saturate in one direction.

Exploding Gradient

Exploding gradients can cause problems in the training of artificial neural networks. Exploding gradients are a problem when large error gradients accumulate and result in very large updates to neural network model weights during training.

When there are exploding gradients, an unstable network is likely to occur, the learning cannot be completed and can cause poor prediction results or even a model that reports nothing useful whatsoever.

The values of the weights can also become so large as to overflow and result in something called NaN values. NaN values, which stands for not a number, are values that represent an undefined or unpresentable value.

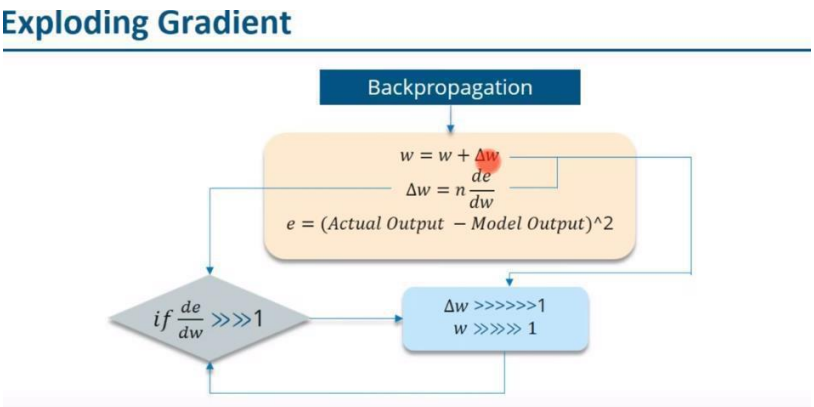

Figure 1.4: Exploding Gradient

Solutions to Exploding Gradient problem

LSTMs: Long Short Term Memory Networks are generally used to tackle Exploding Gradient problems when working on RNN. LTSMs help to solve long term dependencies and can memorize previous data easily.

Gradient Clipping: Gradient Clipping is when we check for and limit the size of gradients during the training of our network. So basically, the values of the error gradient are checked against a threshold value and clipped or set to that threshold value if the error gradient exceeds the threshold.

LSTM: The above drawback of RNN pushed the scientists to develop and invent a new variant of the RNN model, called Long Short Term Memory. LSTM can solve this problem, because it uses gates to control the memorizing process.

Long Short Term Memory networks, usually called "LSTMs", were introduced by Hochreiter and Schmiduber. These have widely been used for speech recognition, language modelling, sentiment analysis and text prediction.

Mixture Density Networks

The idea of mixture density networks [1, 2] is to use the outputs of a neural network to parameterise a mixture distribution. A subset of the outputs are used to define the mixture weights, while the remaining outputs are used to parameterise the individual mixture components.

The mixture weight outputs are normalized with a softmax function to ensure they form a valid discrete distribution, and the other outputs are passed through suitable functions to keep their values within meaningful range (for example the exponential function is typically applied to outputs used as scale parameters, which must be positive).

Mixture density networks are trained by maximizing the log probability density of the targets under the induced distributions. Note that the densities are normalised (up to a fixed constant) and are therefore straightforward to differentiate and pick unbiased samples from, in contrast with restricted Boltzmann machines [3] and other undirected models. Mixture density outputs can also be used with recurrent neural networks [4].

In this case the output distribution is conditioned not only on the current input, but on the history of previous inputs. Intuitively, the number of components is the number of choices the network has for the next output given the inputs so far.

REVIEW OF LITERATURE

The number of contributions made by the researchers on handwriting synthesis is less.
Handwriting synthesis is a common problem in the domain of machine learning.
Generative techniques can be divided into two categories:
1. Movement Simulation Techniques
2. Shape Simulation Method

Movement Simulation Techniques

Movement simulation techniques are mostly based on motor models whereas shape simulation techniques use the trajectories of the handwriting. Synthesizing handwritten characters with movement simulation techniques requires dynamic information about handwriting to be available. Movement-simulation usually requires the acquisition of online data on tablets.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<WhiteboardCaptureSession>
 <WhiteboardDescription>
  <SensorLocation corner="top_left"/>
  <DiagonallyOppositeCoords x="6512" y="1376"/>
  <VerticallyOppositeCoords x="966" y="1376"/>
  <HorizontallyOppositeCoords x="6512" y="787"/>
 </WhiteboardDescription>
 <StrokeSet>
  <Stroke colour="black" start_time="769.05" end_time="769.64">
   <Point x="1073" y="1058" time="769.05"/>
   <Point x="1072" y="1085" time="769.07"/>
   <Point x="1066" y="1117" time="769.08"/>
  </Stroke>
 </StrokeSet>
</WhiteboardCaptureSession>
```

Ilya Sutskever,James Marten, Geoffrey Hinton, 2011

Shape Simulation Techniques

Shape simulation techniques use the trajectories of the handwriting which comprise the writing style's characteristics in a direct way via handwritten shapes. Shape Simulation Techniques model the written samples themselves. Shape Simulation approaches are more practical for offline data.

Generating Text with Recurrent Neural Network

Recurrent Neural Networks (RNNs) are very powerful sequence models that do not enjoy widespread use because it is extremely difficult to train them properly. Fortunately, recent advances in Hessian-free optimization have been able to overcome the difficulties associated with training RNNs, making it possible to apply them.

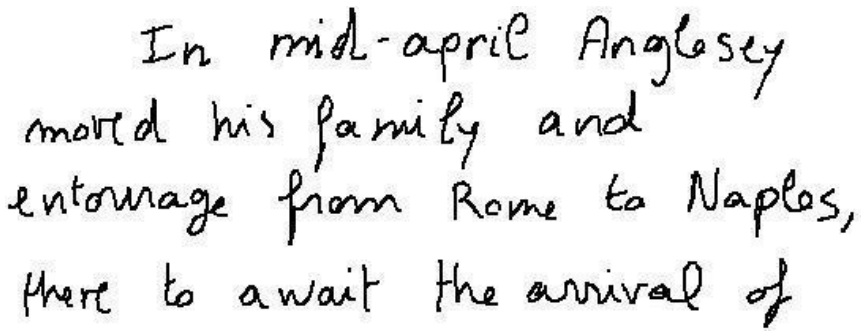

Fig 3.2: Training sample from IAM-OnDB

Data Pre-processing

Pre-processing IAM-OnDB dataset required a lot of focus. The raw input data consists of the (x, y) pen co-ordinates and the points corresponding to the action of lifting the pen off the whiteboard.

Some recording errors in the (x, y) data was corrected by interpolating to fill in for missing values. After this step, the network is directly trained to predict the (x, y) co-ordinates one point at a time.

Architecture

As seen, a traditional recurrent cell is not quite effective in storing long dependencies. So Long Short-Term Memory cells are used in its place. They maintain cell states and have longer memory by making use of various internal gates.

With the help of LSTM cells, we can have long-term dependencies but one LSTM cell may not be so effective in abstracting the details of handwriting stroke. In order to have a deeper understanding of handwriting strokes to the network, multiple LSTM cells are stacked on top of each other to create a deep recurrent neural network.

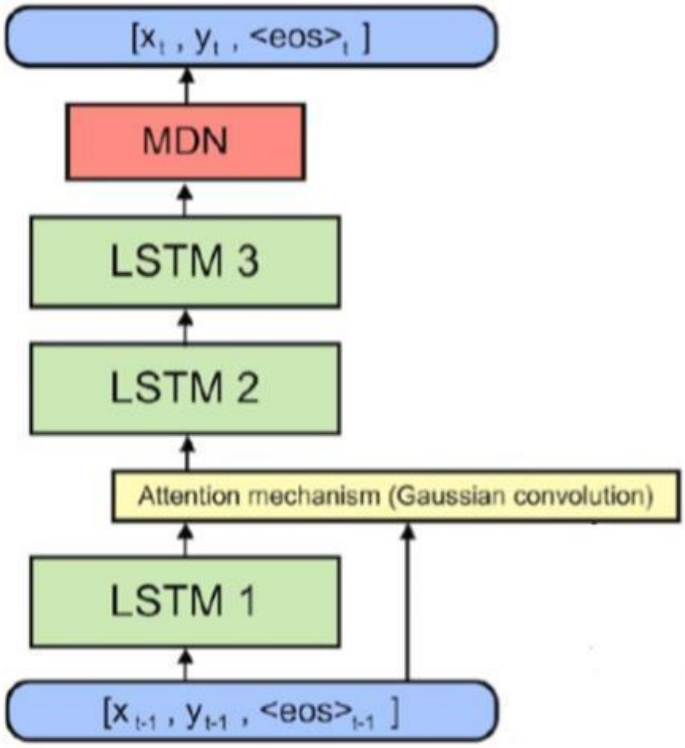

Fig 3.3: Complete architecture of handwriting synthesis

We have used three LSTM layers are used. The rolled architecture figure shows three layers of LSTM. The architecture uses one timestep of the model and does not show the looped connections inside the model. In order to understand those connections, it is better to unroll the model and visualize it as multiple architectures interconnected to each other.

The first layer of LSTM was implemented and then attention mechanism part is hard coded following the mathematical equations and it is attached to the first layer of LSTM. The outputs of attention mechanism are then passed to second layer LSTM and then to final layer LSTM. Now the output of final layer LSTM is attached to Mixture Density Network.

So, in this architecture, we have all the LSTMs and attention mechanisms pass their hidden states to the next timestep [10]. The output of t-1 step will the input of the t step. Mixture Density networks don't have any loop as they are a regular neural network without any hidden states. Therefore, there will not be any hidden state transfer in the case of MDNs.

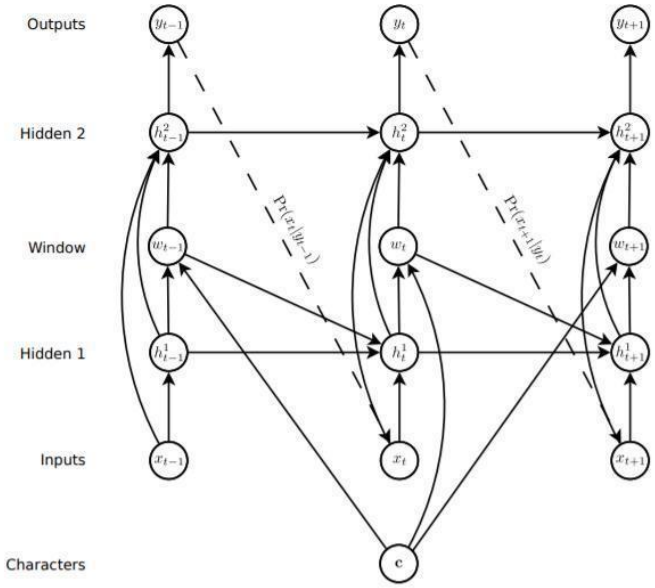

Figure 3.4: Synthesis Network Architecture

In figure 3.17 an input vector sequence x = (x1, . . . , xT) is passed through weighted connections to a stack of N recurrently connected hidden layers to compute first the hidden vector sequences h^n = ($h1^n$, . . . , h^n T) and then the output vector sequence y = (y1, . . . , yT).

The network is 'deep' in both space and time, in the sense that every piece of information passing either vertically or horizontally through the computation graph will be acted on by multiple successive weight matrices and nonlinearities.

The 'skip connections' from the inputs to all hidden layers, and from all hidden layers to the outputs make it easier to train deep networks.

The hidden layers are stacked on top of each other, each feeding up to the layer above, and there are skip connections from the inputs to all hidden layers and from all hidden layers to the outputs.

These make it easier to train deep networks by reducing the number of processing steps between the bottom of the network and the top, and thereby mitigating the 'vanishing gradient' problem.

The **wt** vectors are passed to the second and third hidden layers at time t, and the first hidden layer at t+1 (to avoid creating a cycle in the processing graph). The equations for the hidden layers are

$$h_t^1 = H(W_{ih^1}x_t + W_{h^1h^1}h_{t-1}^1 + W_{wh^1}w_{t-1} + b_h^1) \quad (1)$$

$$h_t^n = H(W_{ih^n}x_t + W_{h^{n-1}h^n}h_t^{n-1} + W_{h^nh^n}h_{t-1}^n + W_{wh^n}w_t + b_h^n) \quad (2)$$

where the W terms denote weight matrices (e.g. W_{ih^n} is the weight matrix connecting the inputs to the n th hidden layer, $W_{h^1h^1}$ is the recurrent connection at the first hidden layer, and so on), the b terms denote bias vectors (e.g. by is output bias vector) and H is the hidden layer function.

Now the output sequence is computed as follows

$$\hat{y}_t = b_y + \sum_{n=1}^{N} W_{h^n y} h_t^n \quad (3)$$

$$y_t = Y(\hat{y}t) \quad (4)$$

LSTM

Theoretically recurrent neural network, can work. But in practice, it suffers from problems discussed above vanishing gradient and exploding gradient, which make it quite unstable and unusable.

Henceforth, LSTM (long short term memory) was invented to solve this issue by explicitly introducing a memory unit, called the cell into the network. So Long Short-Term Memory (LSTM) architecture [9], uses purpose-built memory cells to store information, and is better at finding and exploiting long range dependencies in the data.

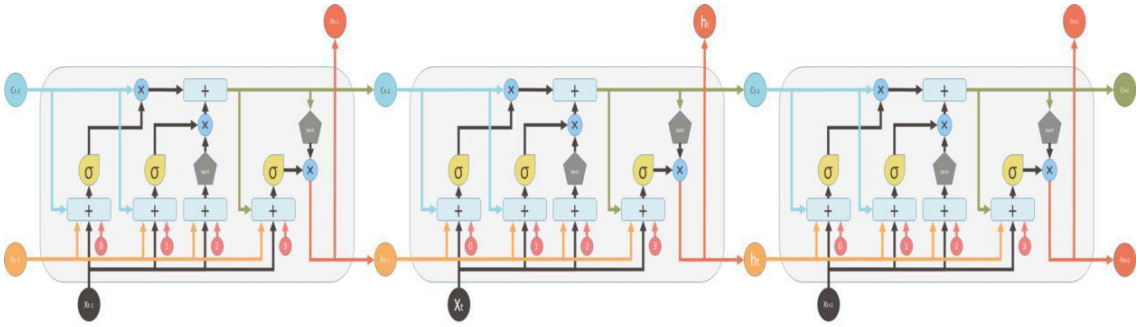

Figure 3.5: LSTM building block.

A simple LSTM cell consists of 4 gates:

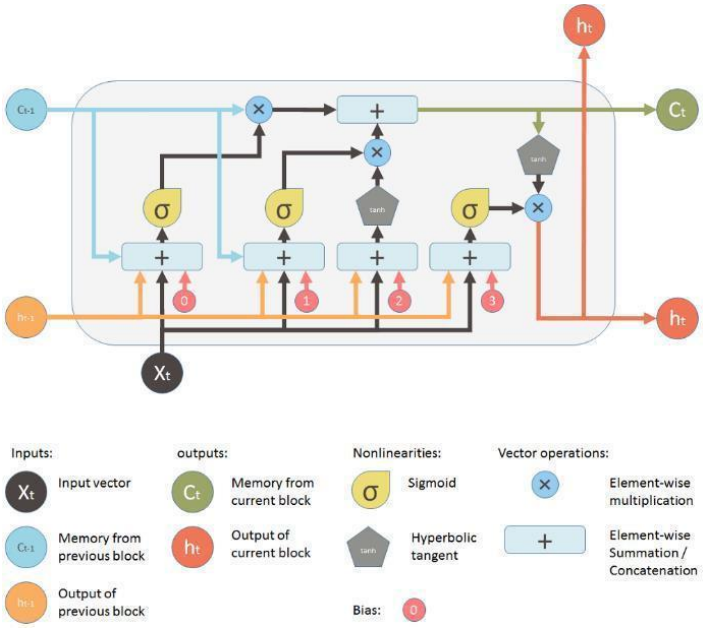

Figure 3.6: Architecture of LSTM cell

Now Inputs of the neural network is:

- Memory of the previous block ht-1
- Output of the previous LSTM block Xt,
- Input for the current LSTM block Ct-1, and
- A bias vector b0

On the LSTM diagram, the top "pipe" is the memory pipe. The input is the old memory (a vector). The first cross ✕ it passes through is the forget valve. It is actually an element-wise multiplication operation. So if we multiply the old memory Ct-1 with a vector that is close to 0, that means we want to forget most of the old memory whereas if your forget valve equals 1 then we want to keep the old memory.

Then the second operation here shall be the + operator. This operator means piece-wise summation. New memory and the old memory will merge by this operation. How much new memory should be added to the old memory is controlled by another valve, the ✕ below the + sign.

After these two operations, the old memory Ct-1 gets changed to the new memory Ct..

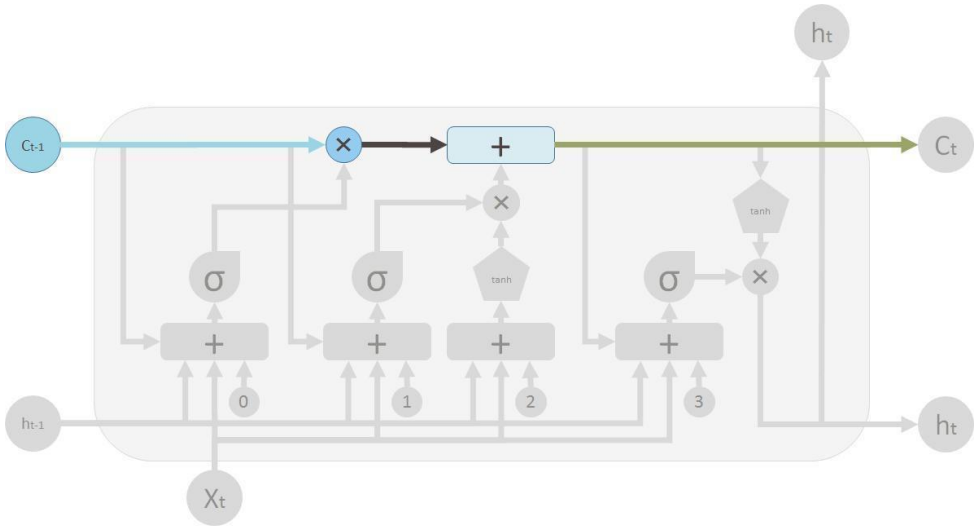

Figure 3.7: LSTM Stage 1

This neural network has a sigmoid function as activation, It's output vector is the forget valve, which will applied to the old memory Ct-1 by element-wise multiplication

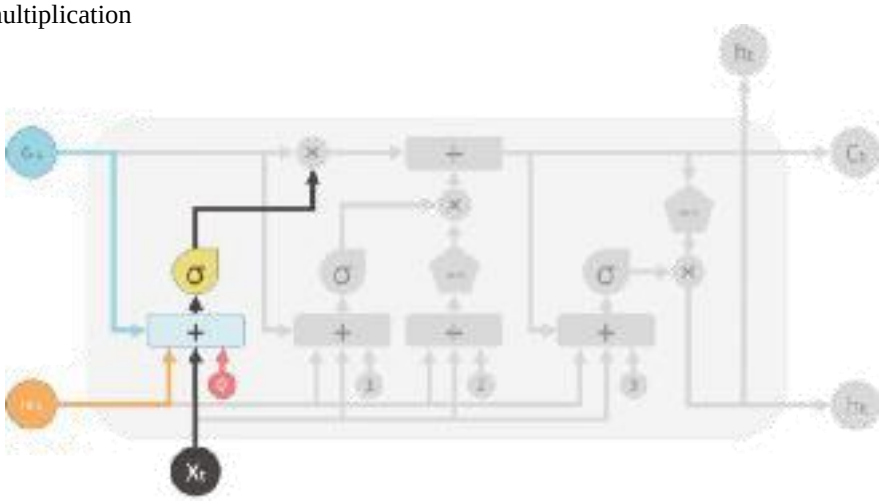

Figure 3.8: LSTM Stage 2

The second valve is called the new memory valve. It is a one layer simple neural network that takes the same inputs as the forget valve. This valve controls how much the new memory should influence the old memory.

The new memory itself, however is generated by another neural network. It is also a one layer network, but uses tanh as the activation function. The output of this network will element-wise multiple the new memory valve, and add to the old memory to form the new memory.

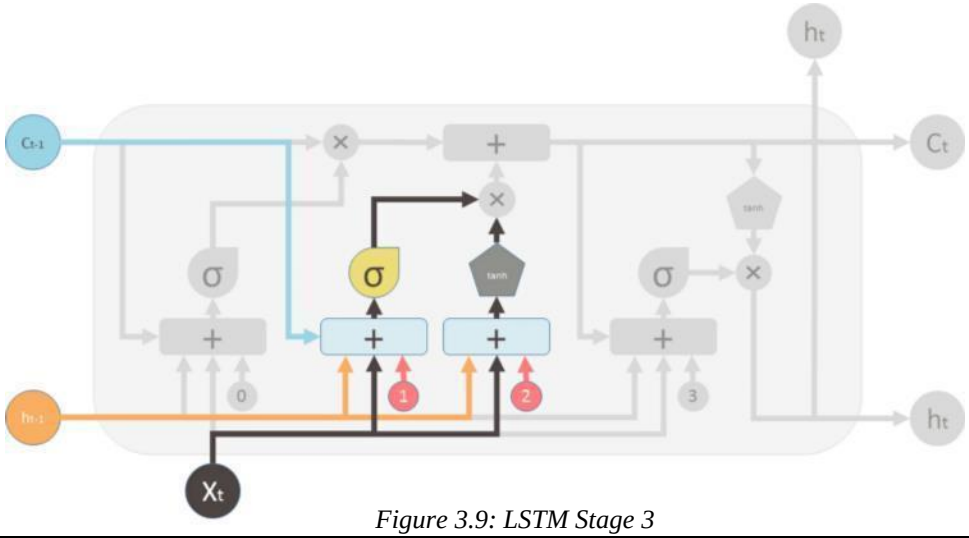

Figure 3.9: LSTM Stage 3

Finally we have to decide what to output from the cell state which will be done by our sigmoid function. This step has an output valve that is controlled by the new memory, the previous output ht-1, the input Xt and a bias vector. This valve controls how much new memory should output to the next LSTM unit. We multiply the input with tanh to crush the values between (-1,1) and then multiply it with the output of sigmoid function so that we only output what we want to.

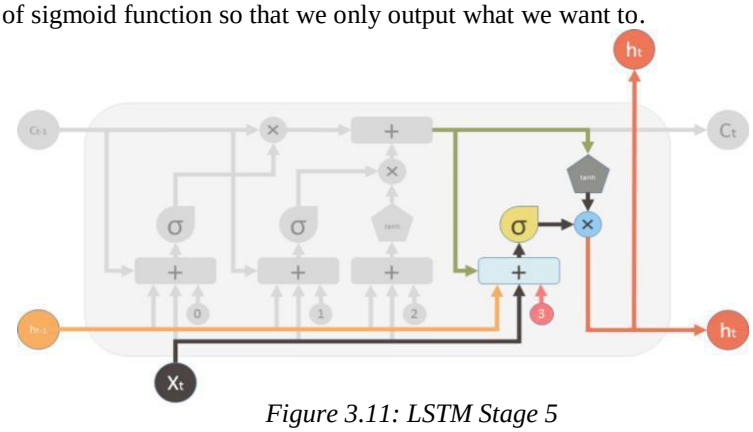

Figure 3.11: LSTM Stage 5

So, in nutshell LSTM cell looks like

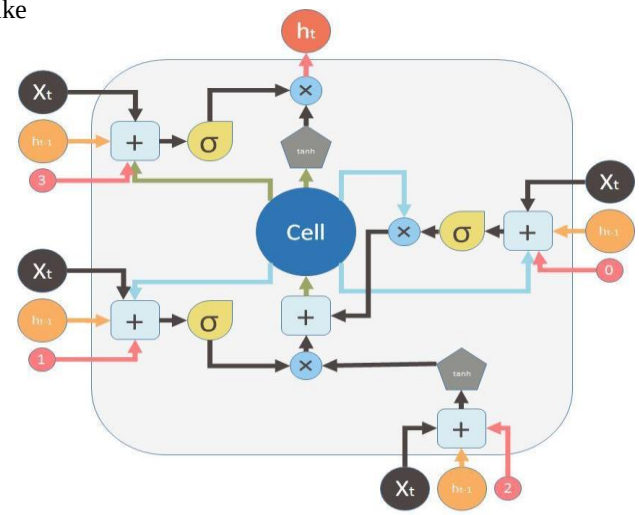

Figure 3.12: Complete setup of LSTM

Forget Gate

Tells about whether to erase the cell. It shuts the old memory.

$$ft = \sigma(Wxf\ xt + Whf\ ht\text{-}1 + Wcf\ ct\text{-}1 + bf) \quad (5)$$

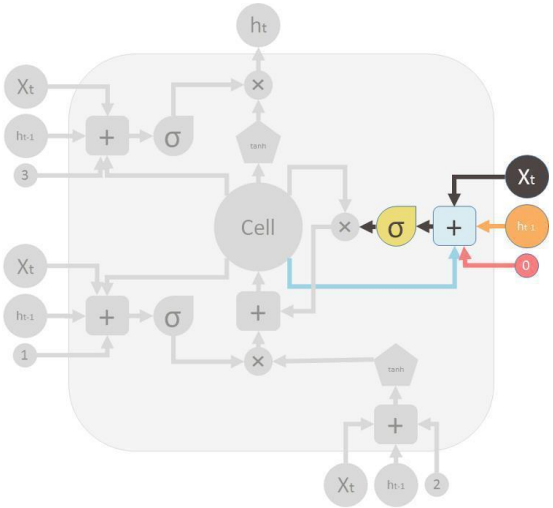

Figure 3.13: Forget Gate

Input Gate
Tell about whether to write the cell. This is the new memory valve.

$$ít = σ(Wxíxt + Whíht-1 + bc) \quad (6)$$

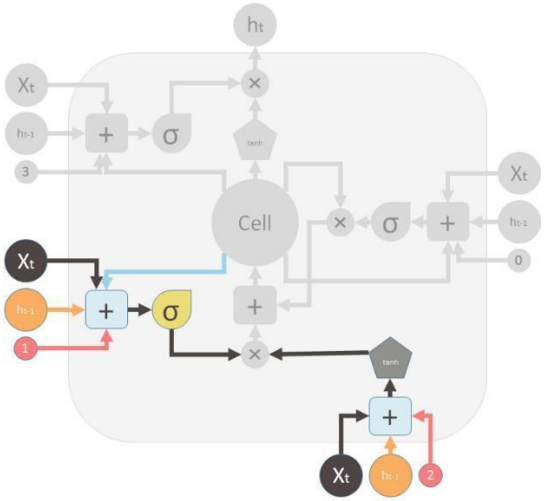

Figure 3.14: In-out Gate

Cell Gate
These are the two valves and the element-wise summation to merge the old memory and the new memory to form Ct

$$ct = ft\ ct\text{-}1 + ít\ \tanh(Wxc\ xt + Whc\ ht\text{-}1 + bc) \quad (7)$$

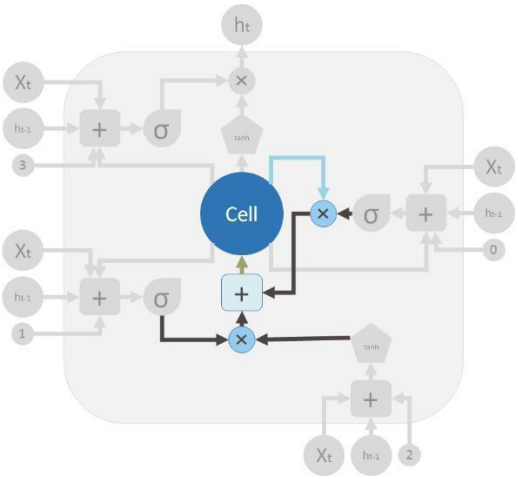

Figure 3.15: Cell Gate

Output Gate
This is the output valve and output of the LSTM unit. It tells how much to reveal the cell.

$$ot = σ(Wxo\ xt + Who\ ht\text{-}1 + Wco\ ct + bo) \quad (8)$$

$$ht = ot \tanh(ct) \quad (9)$$

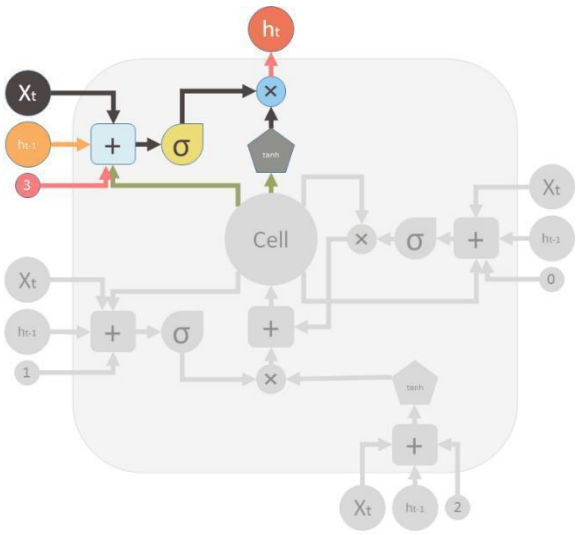

Figure 3.16: Output Gate

Here:

- σ is the logistic sigmoid function,
- *i, f, o* and *c* are respectively the input gate, forget gate, output gate, cell input activation vectors,
- h is the hidden vector.

- The weight matrix subscripts have the obvious meaning, for example Whi is the hidden-input gate matrix, Wxo is the input-output gate matrix etc. The weight matrices from the cell to gate vectors (e.g. Wci) are diagonal, so element m in each gate vector only receives input from element m of the cell vector.

Attention Mechanism

Attention mechanisms in neural networks serve to orient perception as well as memory access (Attention filters the perceptions that can be stored in memory, and filters them again on a second pass when they are to be retrieved from memory. It can be aimed at the present and the past.

Below equations describe how the attention mechanism is implemented and will give an intuition for how the parameters α, β, and κ affect the window's behavior.

$$\left(\hat{\alpha}_t, \hat{\beta}_t, \hat{K}_t\right) = W_{h^1 p} h_t^1 + b_p \quad (10)$$

$$\alpha_t = \exp\left(\hat{\alpha}_t\right) \quad (11)$$

$$\beta_t = \exp\left(\hat{\beta}_t\right) \quad (12)$$

$$k_t = k_{t-1} + \exp\left(\hat{k}_t\right) \quad (13)$$

Now given a soft window **wt** into **c** at timestep **t** ($1 \leq t \leq T$) is defined by the following discrete convolution with a mixture of K Gaussian functions.

$$\phi(t,u) = \sum_{k=1}^{K} \quad \alpha_t^k\left(-\beta_t^k(K_t^k - a^2)\right) \tag{14}$$

$$\omega_t = \sum_{u=1}^{U} \phi(t,u)C_u \tag{15}$$

Here **φ(t, u)** is the window weight of **cu** at timestep t.

Intuitively, the **κt** parameters control the location of the window,

The **βt** parameters control the width of the window and

The **αt** parameters control the importance of the window within the mixture.

The size of the soft window vectors is the same as the size of the character vectors cu (assuming a one-hot encoding, this will be the number of characters in the alphabet).

The location parameters κt are defined as offsets from the previous locations ct−1, and that the size of the offset is constrained to be greater than zero. Intuitively, this means that network learns how far to slide each window at each step, rather than an absolute location. Using offsets was essential to getting the network to align the text with the pen trace.

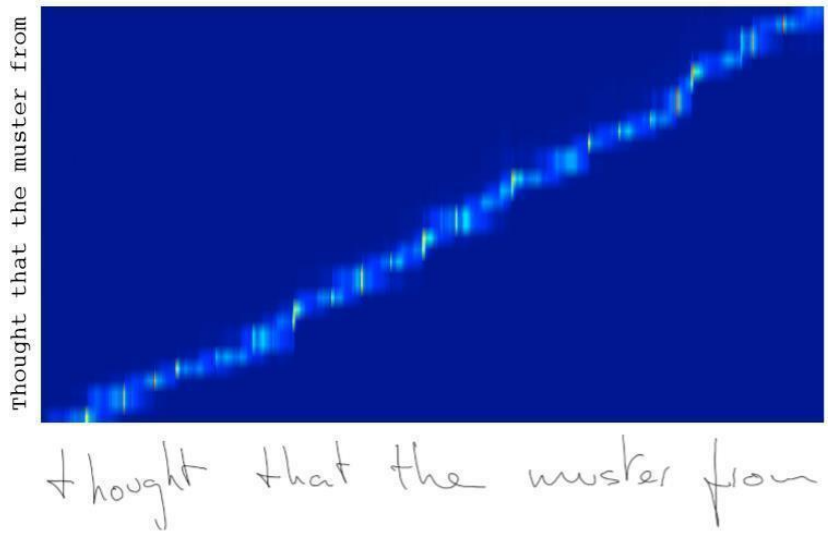

Figure 3.17: Window weights during a handwriting synthesis sequence. Each point on the map shows the value of φ(t, u), where t indexes the pen trace along the horizontal axis and u indexes the text character along the vertical axis. The bright line is the alignment chosen by the network between the characters and the writing.

Mixture Density Network for Handwriting Generation

The idea of mixture density networks [14] is to use the outputs of a neural network to parameterise a mixture distribution. A subset of the outputs are used to define the mixture weights, while the remaining outputs are used to parameterise the individual mixture components.

Each input vector xt consists of a real-valued pair x1, x2 that defines the pen offset from the previous input, along with a binary x3 that has value 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 x1 and x2, while Bernoulli distribution was used for x3. Each output vector yt consists of:

- End of stroke probability e,
- A set of means μj
- Standard deviations σj

- Correlations ρj
- Mixture weights πj for the M mixture components.

$$x_t \, \varepsilon \, R \times R \times \{0,1\} \tag{16}$$

$$y_t = (e_t, \{\pi_t^j, \mu_t^j, \sigma_t^j, \rho_t^j\}_{j=1}^{M}) \tag{17}$$

$$\hat{y}_t = \left(\hat{e}_t, \{\hat{\pi}_t^j, \hat{\mu}_t^j, \hat{\sigma}_t^j, \hat{\rho}_t^j\}_{j=1}^{M} \right) = b_y + \sum_{n=1}^{N} W_{h^n y} h_t^n \tag{18}$$

$$e_t = \frac{1}{1+\exp(\hat{e}_t)} \quad \Rightarrow \quad e(t) \, \epsilon \, (0,1) \tag{19}$$

$$\pi_t^j = \frac{\exp(\hat{\pi}_t^j)}{\Sigma_{j'=1}^{M} \exp(\hat{\pi}_t^{j'})} \quad \Rightarrow \quad \pi_t^j \, \varepsilon(0,1), \ \sum_j \pi_t^j = 1 \tag{20}$$

$$\mu_t^j = \hat{\mu}_t^j \quad \Rightarrow \quad \mu_t^j \, \epsilon \, R \tag{21}$$

$$\sigma_t^j = \exp(\hat{\sigma}_t^j) \quad \Rightarrow \quad \sigma_t^j > 0 \tag{22}$$

$$\rho_t^j = \tanh(\hat{\rho}_t^j) \quad \Rightarrow \quad \rho_t^j \, \epsilon \, (-1,1) \tag{23}$$

$$\Pr(x_{t+1}|y_t) = \sum_{j=1}^{M} \pi_t^j \, N(x_{t+1}|\mu_t^j, \, \sigma_t^j \, , \rho_t^j \,) \begin{cases} e_t & if \ x_{(t+1)3} = 1 \\ 1 - e_t & otherwise \end{cases} \tag{24}$$

$$N(x|\mu, \sigma, \rho \,) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \exp\left[\frac{-Z}{2(1-\rho^2)}\right] \tag{25}$$

$$Z = \frac{(x_1-\mu_1)^2}{\sigma_1^2} + \frac{(x_2-u_2)^2}{\sigma_2^2} - \frac{2\rho(x_1-u_1)(x_2-\mu_2)}{\sigma_1\sigma_2} \tag{26}$$

The mixture weight outputs are normalized with a softmax function to ensure they form a valid discrete distribution, and the other outputs are passed through suitable functions to keep their values within meaningful range (for example the exponential function is typically applied to outputs used as scale parameters, which must be positive).

Technologies used:

Python: Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design gives great importance to code readability with significant use of whitespace. Its language constructs and object-oriented approach aims to help programmers write clear, logical code for small and large-scale projects.
It is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming.
Python is the top choice among developers for artificial intelligence (AI), machine learning, and deep learning projects. Python [31] has an abundance of libraries and frameworks that facilitate coding and save development time.
NumPy, used for scientific computation, SciPy for advanced computation are among the most popular libraries, working alongside such heavy-hitting frameworks as TensorFlow.

Python's simple syntax means that it is also faster in development than many programming languages, and allows the developer to quickly test algorithms without having to implement them.
Matplotlib is used to generate the handwriting and display it on a graph.

TensorFlow

TensorFlow is a free and open source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for ML applications such as Neural Networks. TensorFlow offers multiple levels of abstraction so you can choose the right one for your needs.

TensorFlow provides a direct path to production. Whether it's on servers, edge devices, or the web, TensorFlow lets us train and deploy your model easily, no matter what language or platform we use.

Jupyter Notebook

The Jupyter Notebook is an open-source web application that allows to create and share documents that contain live code, equations, visualizations and narrative text. Jupyter Notebook is an open-source web application that allows users to create and share codes and documents. It provides an environment, where we can document the code, run it, look at the outcome, visualize data and see the results without leaving the environment.

Outputs

The outputs with different styles and biases are shown.

- bias (float) - with higher bias generated handwriting is more clear so to speak
- style - style of handwriting, int from 0 to 7.

The output is finally created using the resultant set of points(an array of n values each containing 3 points) generated from the output of Mixture Density Networks.
x1 and x2 are used to specify the coordinates while x3 is used to identify whether the point is an eos marker or not.

The input text which is to be synthesized is "This is our thesis project."

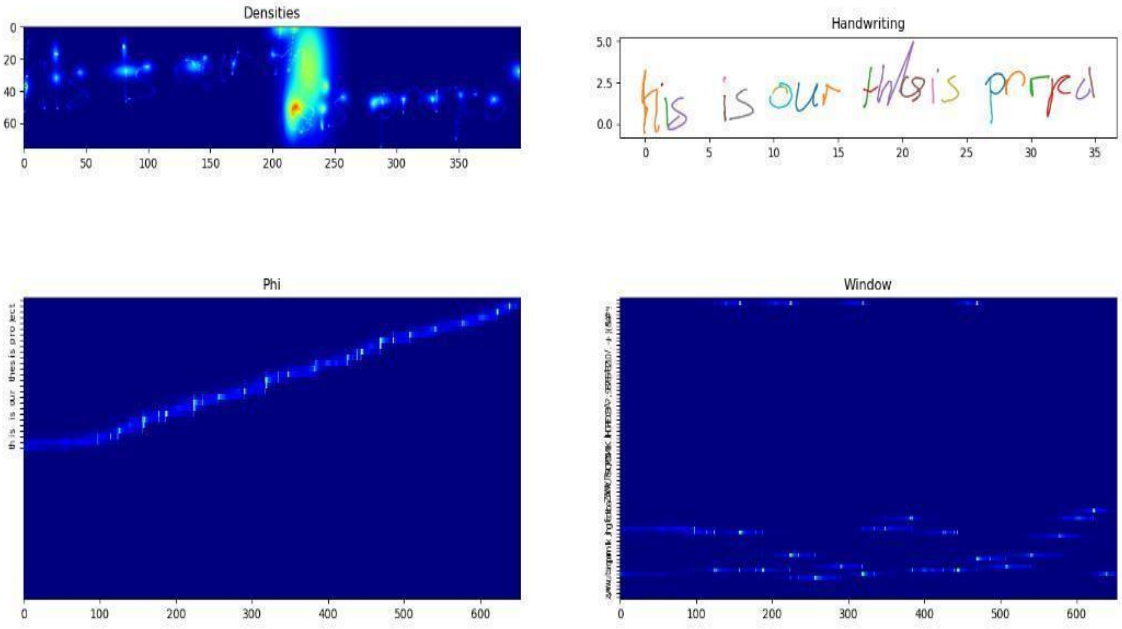

Fig 3.2.1: Synthesized Handwriting in style-0 with bias set to 0

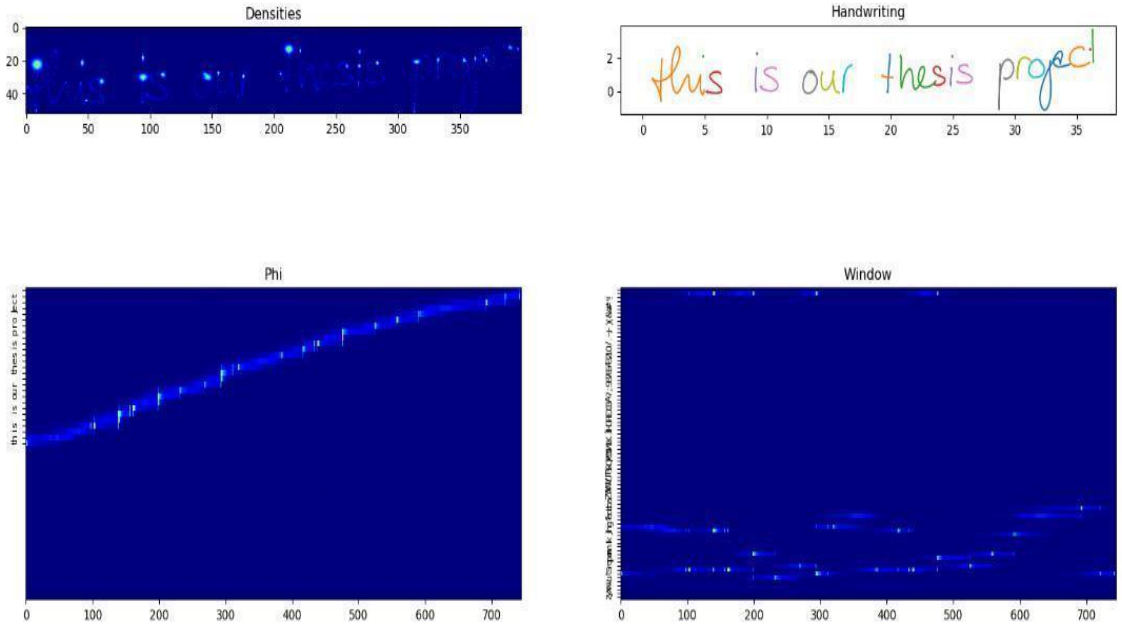

Fig 3.2.2: Synthesized Handwriting in style-0 with bias set to 10

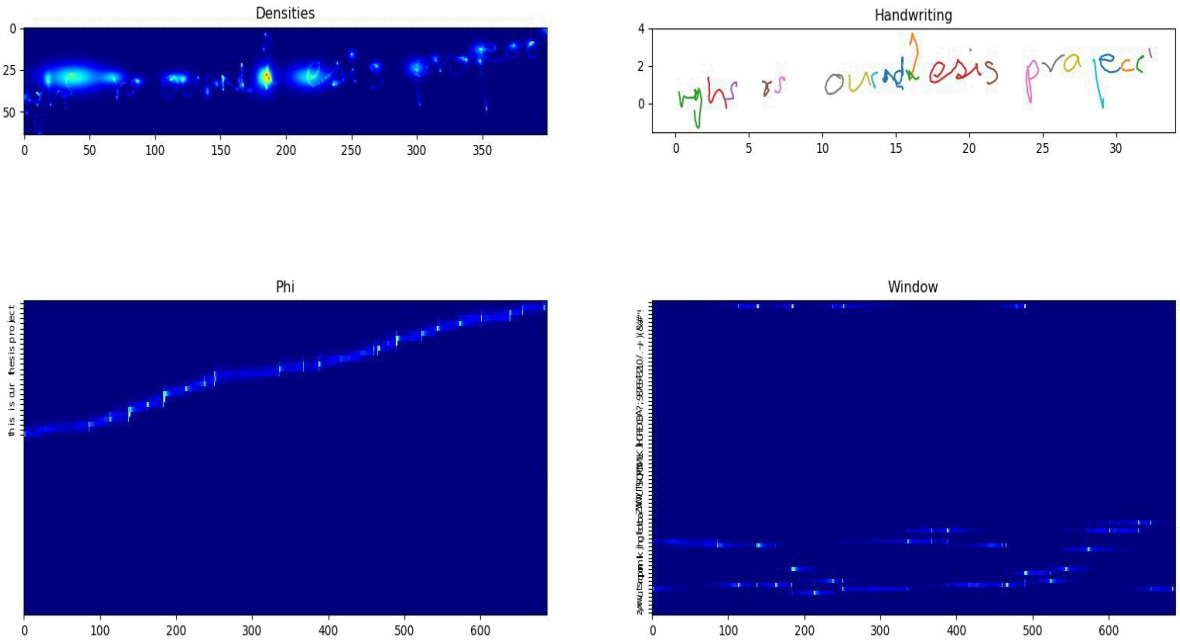

Fig 3.3.1: Synthesized Handwriting in style-1 with bias set to

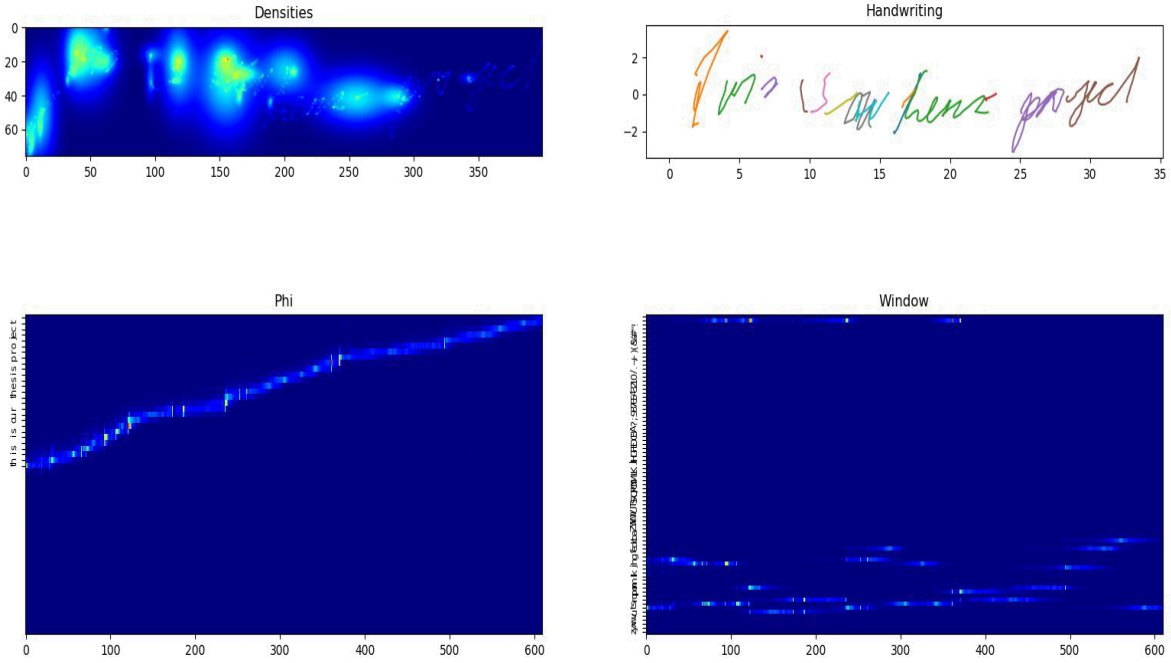

Fig 3.3.2: Synthesized Handwriting in style-1 with bias set to 10

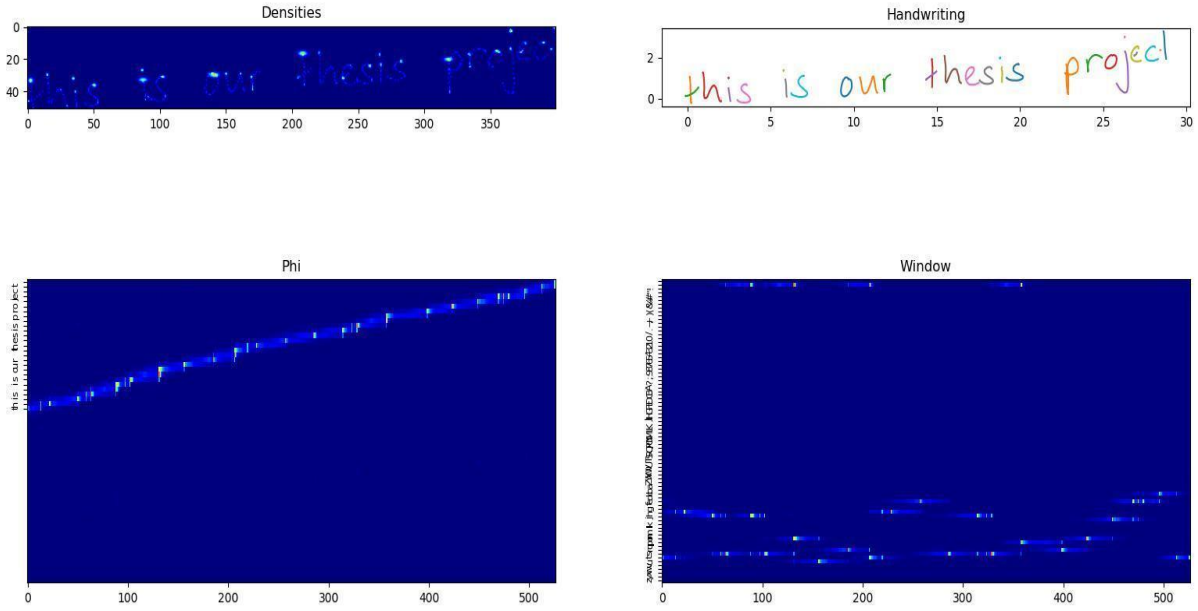

Fig 3.4.1: Synthesized Handwriting in style-2 with bias set to 0

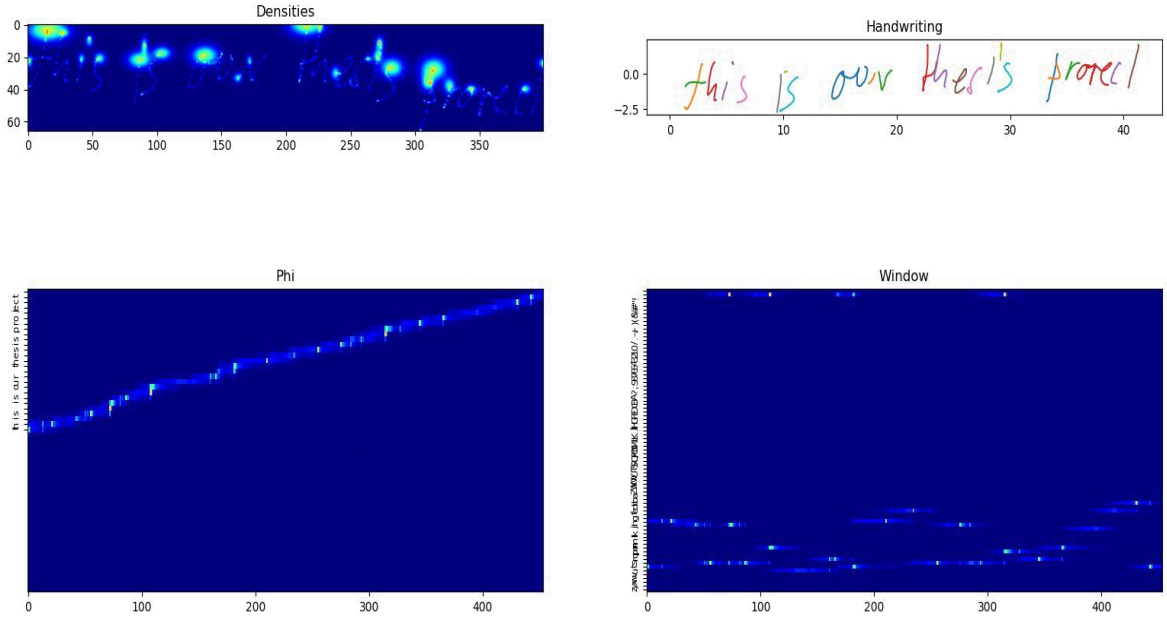

Fig 3.4.2: Synthesized Handwriting in style-2 with bias set to 10

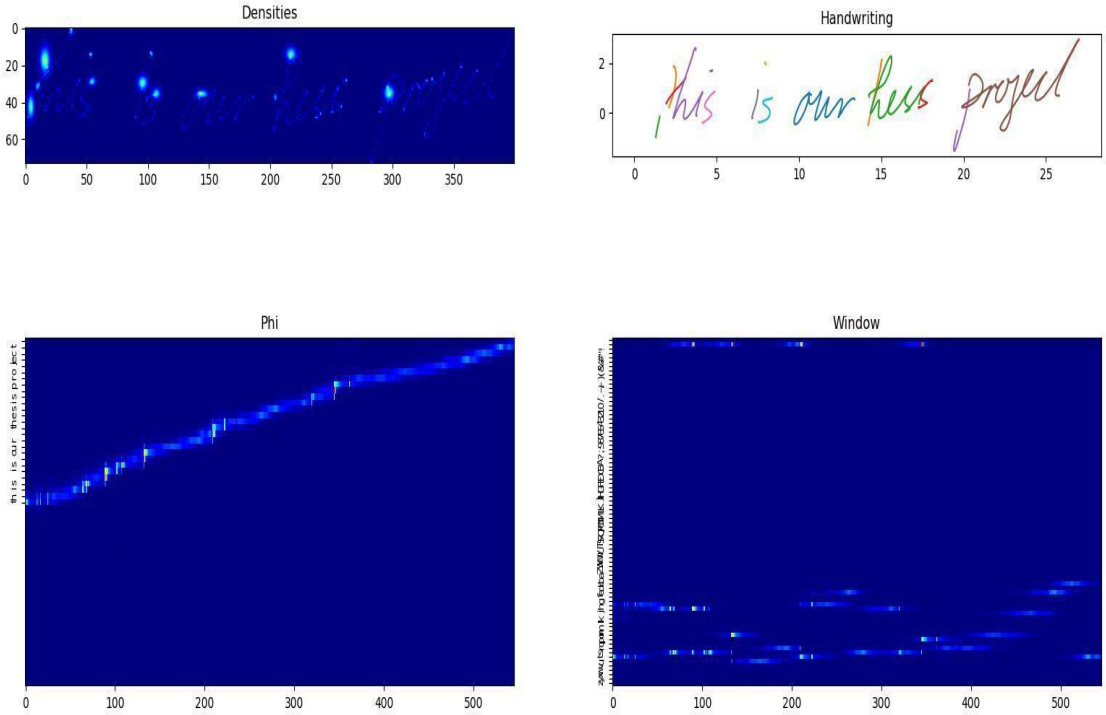

Fig 3.5.1: Synthesized Handwriting in style-3 with bias set to 0

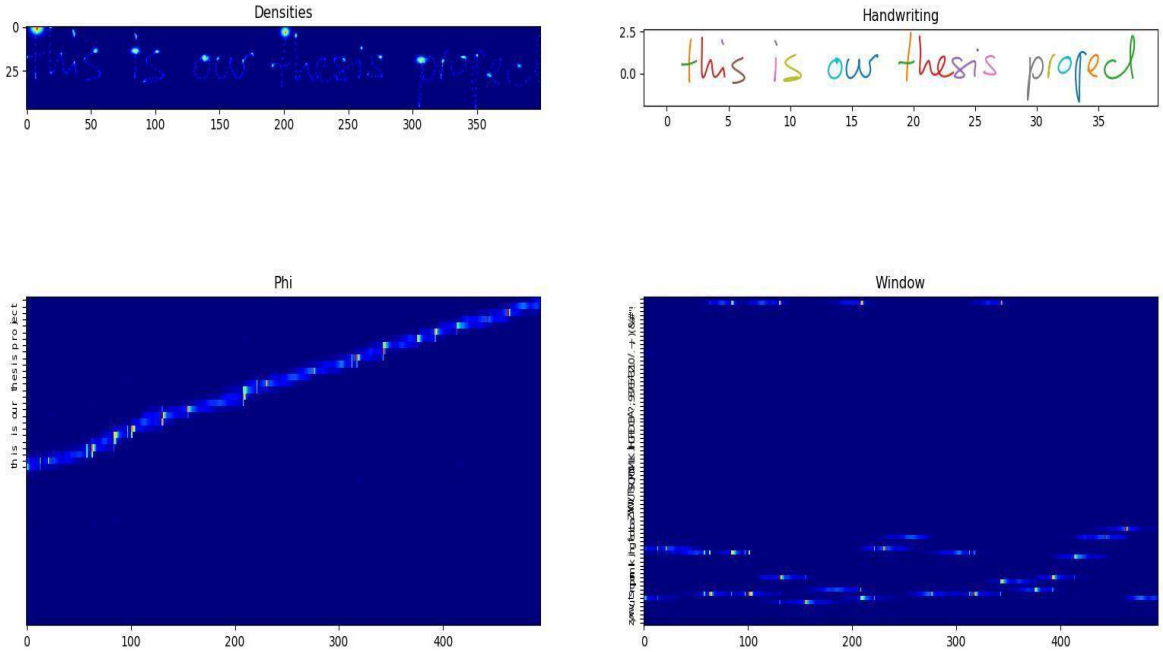

Fig 3.5.2: Synthesized Handwriting in style-3 with bias set to 10

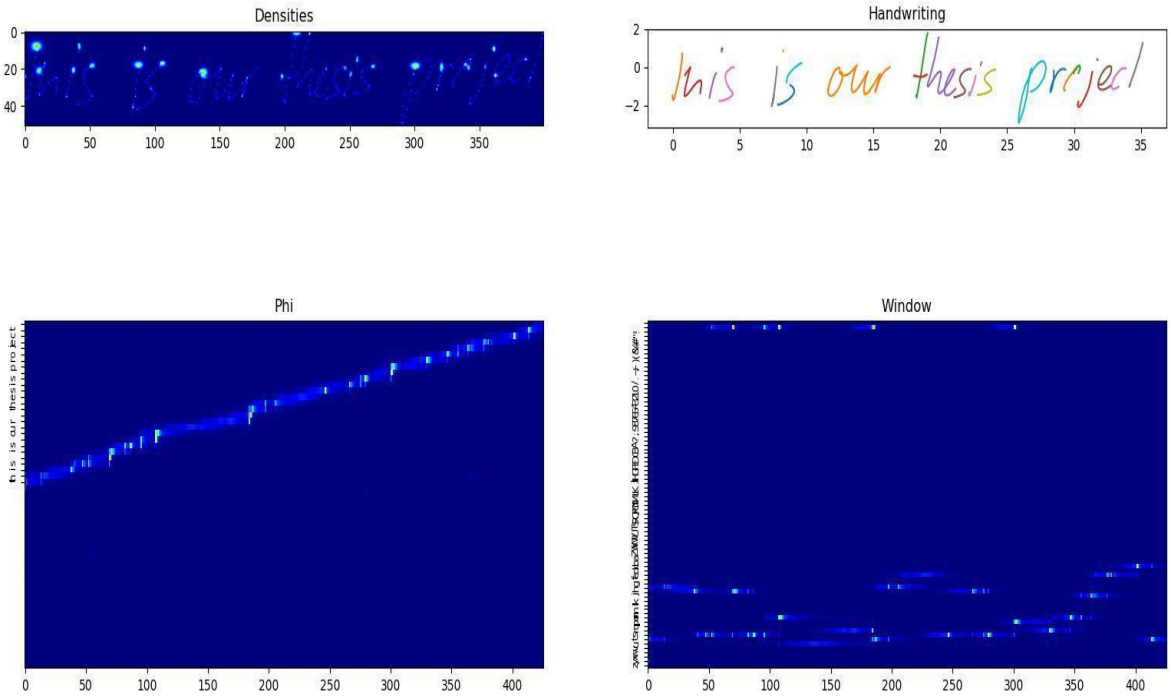

Fig 3.6.1: Synthesized Handwriting in style-4 with bias set to 0

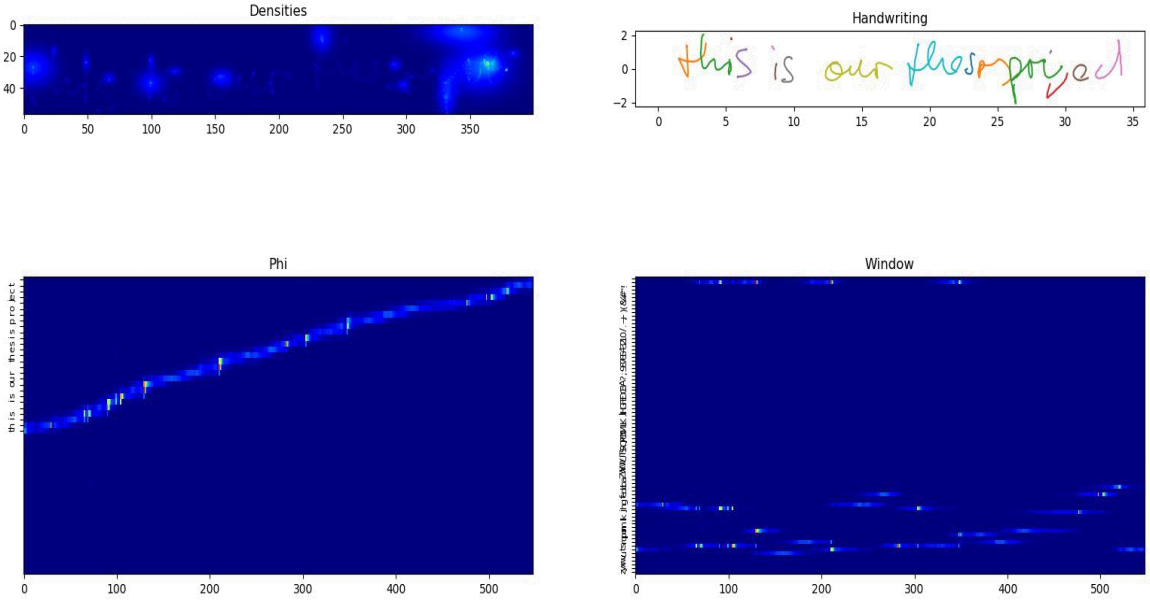

Fig 3.6.2: Synthesized Handwriting in style-4 with bias set to 10

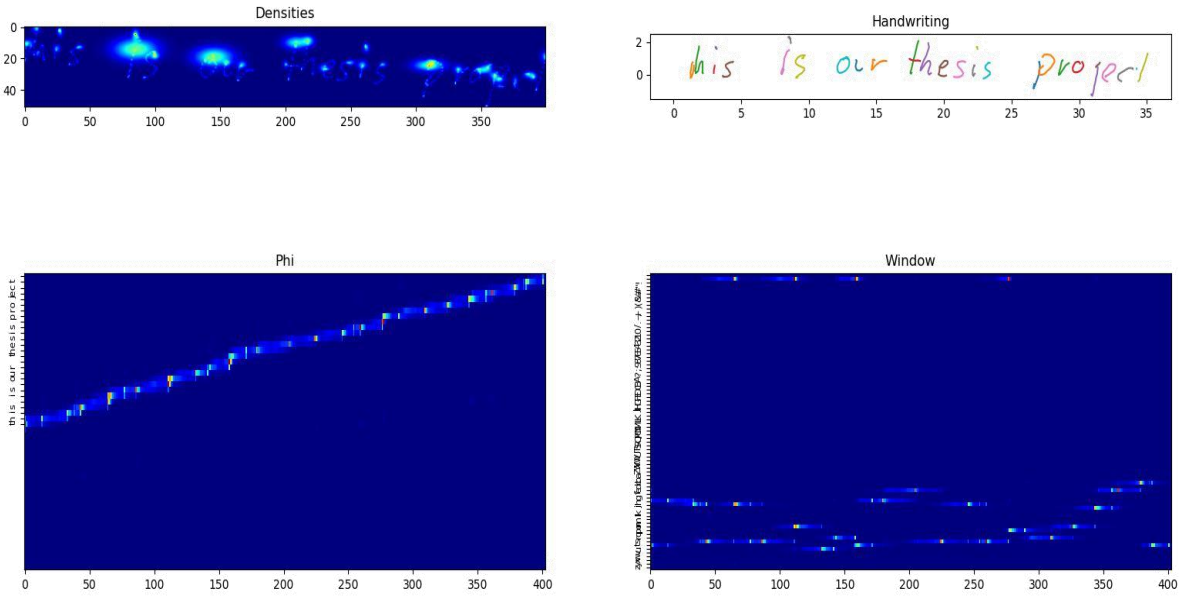

Fig 3.7.1: Synthesized Handwriting in style-5 with bias set to 0

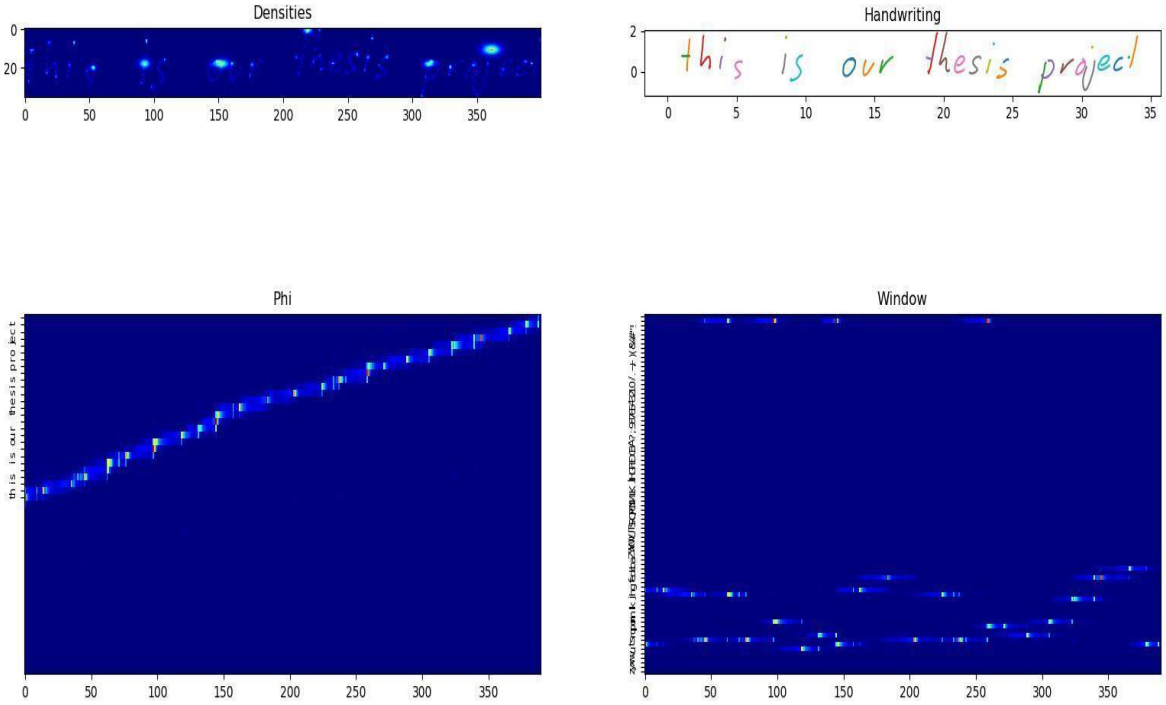

Fig 3.7.2: Synthesized Handwriting in style-5 with bias set to 10

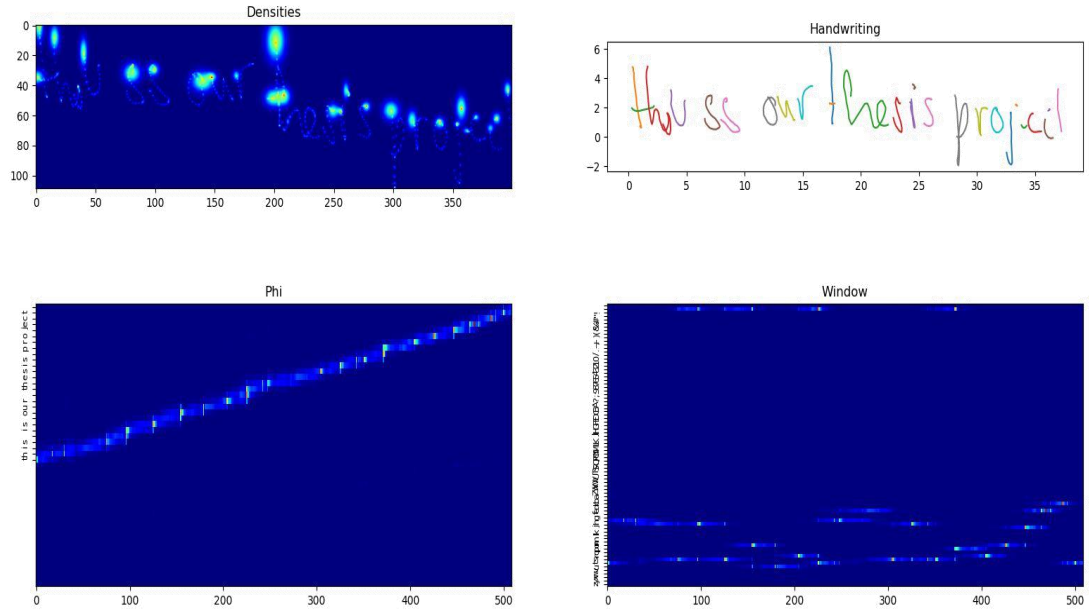

Fig 3.8.1: Synthesized Handwriting in style-6 with bias set to 0

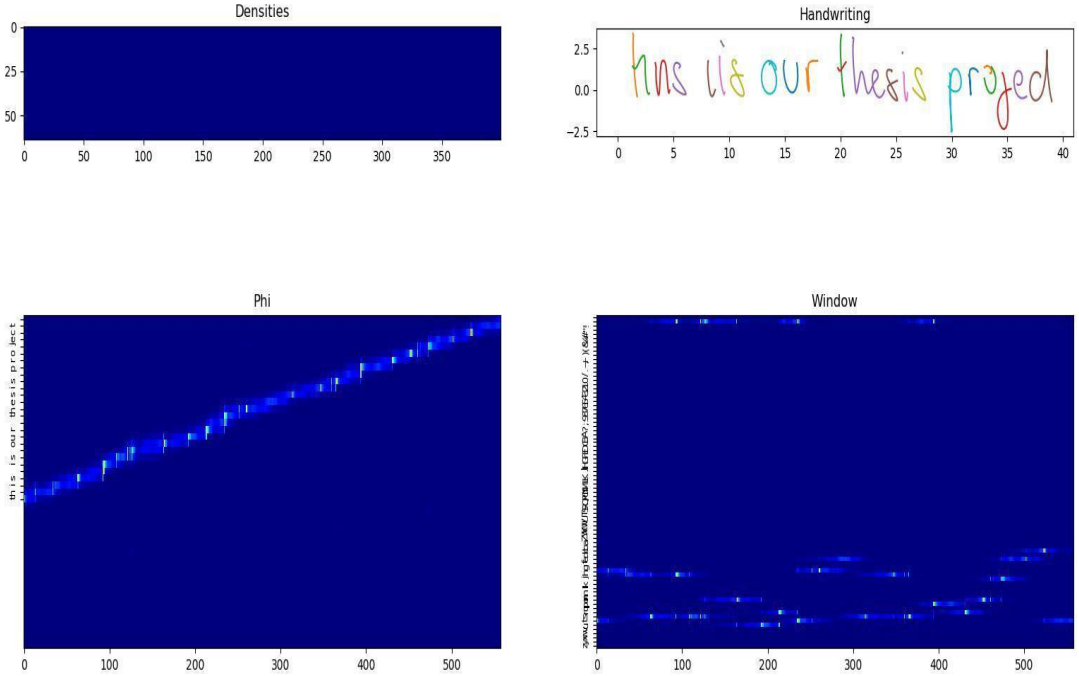

Fig 3.9.1: Synthesized Handwriting in style-7 with bias set to 0

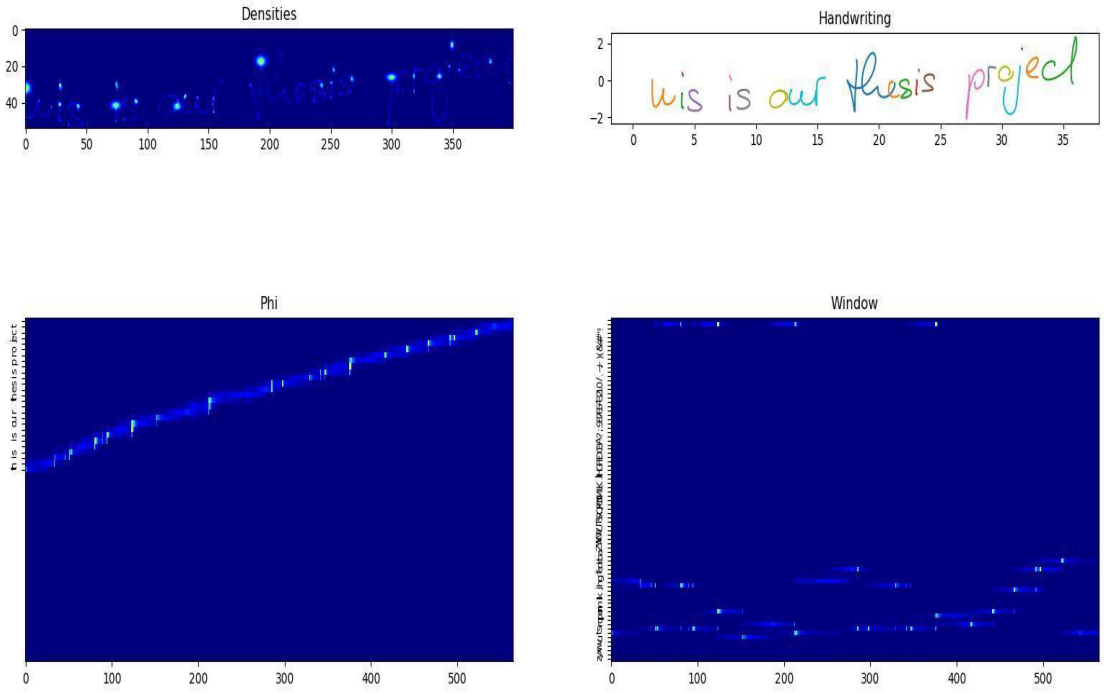

Fig 3.9.2: Synthesized Handwriting in style-7 with bias set to 10

Conclusions

Handwriting generation is a complicated task and it is even more difficult to mimic the particular style. With the explained model, we can achieve satisfying results. Results of the model totally depend on its hyper parameters. Tuning them properly is necessary. Few limitations observed with the model is that it cannot generate longer strings at once because of LSTMs limit. And also, the model needs more data for efficient results.

This project introduced a novel convolutional mechanism that allowed a recurrent neural network with LSTM to condition its predictions on an auxiliary annotation sequence, and used the approach to synthesize diverse and realistic samples of online handwriting.

Beyond that, here no segmentation was used and the network was trained to predict the x, y coordinates and the end-of-stroke markers one point at a time which contrasts with most approaches to handwriting recognition and synthesis that rely on sophisticated pre-processing and feature-extraction techniques. We eschewed such techniques because they tend to reduce the variation in the data (e.g. by normalising the character size, slant, skew and so-on) which we wanted the network to model the same.

Team work, exposure to new state-of-art technologies and algorithms in the field of Machine Learning and Computer Vision are the learning outcomes of this project.

Future Scope:

Extending the method to understand offline data too can be the future work of the project. Here offline data means scanned images of handwriting documents.

Use of more sophisticated Neural Networks and advanced methods like Multidimensional Long-Short Term Memory (MDLSTM) and attention mechanisms can result in better accuracy results, but at the expense of increased training time and increased complexity.

We can use the application of the network to speech synthesis, which is likely to be more challenging than handwriting synthesis due to the greater dimensionality of the data points.

We can widen our dataset by incorporating writing styles of various writers which can lead to further refinement and better practical results.

With the advancements in the RNNs and model can also be improved to mimic any style with less training data. This method is not limited to handwriting data. It can be functional to any sequential data with few tweaks. Moreover, in future, this designed model can be applied in a much more useful real-time application.

Also we could even incorporate Generative Adversarial Net approaches to recurrent neural networks. As this could train a network to discriminate between fake handwriting and real one, and another network to generate fake handwriting to fool the discriminator networks.

REFERENCES:

[1] Bishop, Christopher M. *Mixture density networks*. Technical Report NCRG/4288, Aston University, Birmingham, UK, 1994.
[2] Bishop, Christopher M. *Neural networks for pattern recognition*. Oxford university press, 1995.
[3] Nair, Vinod, and Geoffrey E. Hinton. "Rectified linear units improve restricted boltzmann machines." *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010.
[4] Schuster, Mike. "Better Bidirectional recurrent mixture*Processing Systems*. 2000.generativemodels for sequential data problems:density networks." *Advances in Neural Information*
[5] Elanwar, Randa I. "The state of the art in handwriting synthesis." *2nd International Conference on New Paradigms in Electronics & information Technology (peit'013),*
Luxor, Egypt. 2013.
[6] Haines, Tom SF, Oisin Mac Aodha, and Gabriel J. Brostow. "My text in your handwriting." *ACM Transactions on Graphics (TOG)* 35.3 (2016): 26.
[7] Rajat Shah, Shritek Jain, Sridhar Swamy. "Handwriting Recognition, Learning and Generation." , 2014
[8] Graves, Alex. "Generating sequences with recurrent neural networks." *arXiv preprint arXiv:1308.0850* (2013).
[9] Bezine, Hala, Adel M. Alimi, and Nabil Derbel. "Handwriting trajectory movements controlled by a beta-elliptic model." *tc* 1 (2003): 0.
[10] Lin, Zhouchen, and Liang Wan. "Style-preserving English handwriting synthesis." *Pattern Recognition* 40.7 (2007): 2097-2109.
[11] Varga, Tamás, Daniel Kilchhofer, and Horst Bunke. "Template-based synthetic handwriting generation for the training of recognition systems." *Proceedings of the 12th Conference of the International Graphonomics Society*. 2005.
[12] Liwicki, Marcus, and Horst Bunke. "IAM-OnDB-an on-line English sentence database acquired from handwritten text on a whiteboard." *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*. IEEE, 2005.
[13] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." *Neural computation* 9.8 (1997): 1735-1780.
[14] Sutskever, Ilya, James Martens, and Geoffrey E. Hinton. "Generating text with recurrent neural networks." *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. 2011.
[15] Messina, Ronaldo, and Jerome Louradour. "Segmentation-free handwritten Chinese text recognition with LSTM-RNN." *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, 2015.
[16] Olah, Christopher. "Understanding lstm networks, 2015." *URL http://colah. github. io/posts/2015-08-Understanding-LSTMs*(2015).
[17] Srivastava, Pranjal. "Essentials of Deep Learning: Introduction to Long Short Term Memory." *Analytics Vidhya* 23 (2017).
[18] Bishop, Christopher M. *Mixture density networks*. Technical Report NCRG/4288, Aston University, Birmingham, UK, 1994.
[19] Mike Dusenberry, "Mixture Density Network", 2017
[20] Plamondon, Rejean, and Frans J. Maarse. "An evaluation of motor models of handwriting." *IEEE Transactions on systems, man, and cybernetics* 19.5 (1989): 1060-1072.
[21] Guerfali, Wacef, and Réjean Plamondon. "The delta lognormal theory for the generation and modeling of cursive characters." *Proceedings of 3rd International Conference on Document Analysis and Recognition*. Vol. 1. IEEE, 1995.
[22] Singer, Yoram, and Naftali Tishby. "Dynamical encoding of cursive handwriting." *Biological Cybernetics* 71.3 (1994): 227-237.
[23] Wang, Jue, et al. "Learning-based cursive handwriting synthesis." *Proceedings Eighth International Workshop on Frontiers in Handwriting Recognition*. IEEE, 2002.
[24] Choi, Hyunil, Sung-Jung Cho, and JinHyung Kim. "Generation of handwritten characters with bayesian network based on-line handwriting recognizers." *7th International Conference on Document Analysis and Recognition, Edinburgh, Scotland*. 7th International Conference on Document Analysis and Recognition, Edinburgh, Scotland, 2003.
[25] Xu, Songhua, et al. "Automatic generation of artistic Chinese calligraphy." *IEEE Intelligent Systems* 20.3 (2005): 32-39.
[26] Mogren, Olof. "C-RNN-GAN: Continuous recurrent neural networks with adversarial training." *arXiv preprint arXiv:1611.09904* (2016).
[27] "CSS framework", 2019 Wikipedia :
https://en.wikipedia.org/wiki/CSS_framework+
[28] Jakub Protasiewicz, "Why Is Python So Good for AI, Machine Learning and Deep Learning?", 2018
[29] "HTML", 2019 *Wikipedia* : https://en.wikipedia.org/wiki/HTML
[30] "Bootstrap (front-end framework)", 2019 *Wikipedia* : https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework)
[31] Tania Rascia, "what is bootstrap and how do i use it", 2015
[32] Elarian, Yousef, et al. "An Arabic handwriting synthesis system." *Pattern Recognition* 48.3 (2015): 849-861.
[33] Andrychowicz, Marcin, et al. "Learning to learn by gradient descent by gradient descent." *Advances in Neural Information Processing Systems*. 2016.
[34] Kandala, Harish, B. K. Tripathy, and K. Manoj Kumar. "A framework to collect and visualize user's browser history for better user experience and personalized recommendations." *International Conference on Information and Communication Technology for Intelligent Systems*. Springer, Cham, 2017.

[35] Razvi, Salma Abid, et al. "Implementation of graphical passwords in internet banking for enhanced security." *2017 International Conference on Intelligent Computing and Control Systems (ICICCS)*. IEEE, 2017.
[36] Bengio, Samy, et al. "Scheduled sampling for sequence prediction with recurrent neural networks." *Advances in Neural Information Processing Systems*. 2015.
[37] Ghosh, Arna, Biswarup Bhattacharya, and Somnath Basu Roy Chowdhury. "Handwriting profiling using generative adversarial networks." *Thirty-First AAAI Conference on Artificial Intelligence*. 2017.
[38] Goodfellow, Ian, et al. "Generative adversarial nets." *Advances in neural information processing systems*. 2014.
[39] Williams, Ronald J., and David Zipser. "Gradient-based learning algorithms for recurrent." *Backpropagation: Theory, architectures, and applications* 433 (1995).
[40] Tieleman, Tijmen, and Geoffrey Hinton. "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude." *COURSERA: Neural networks for machine learning* 4.2 (2012): 26-31.
[41] Robinson, Tony. "An application of recurrent nets to phone probability estimation." *IEEE transactions on Neural Networks* (1994).
[42] Graves, Alex, and Jürgen Schmidhuber. "Offline handwriting recognition with multidimensional recurrent neural networks." *Advances in neural information processing systems*. 2009.
[43] Hinton, Geoffrey E. "A practical guide to training restricted Boltzmann machines." *Neural networks: Tricks of the trade*. Springer, Berlin, Heidelberg, 2012. 599-619.
[44] Graves, Alex, and Jürgen Schmidhuber. "Framewise phoneme classification with bidirectional LSTM and other neural network architectures." *Neural Networks* 18.5-6 (2005): 602-610
[45] Bengio, Yoshua, Patrice Simard, and Paolo Frasconi. "Learning long-term dependencies with gradient descent is difficult." *IEEE transactions on neural networks* 5.2 (1994): 157-166.
[46] Gers, Felix A., Nicol N. Schraudolph, and Jürgen Schmidhuber. "Learning precise timing with LSTM recurrent networks." *Journal of machine learning research* 3.Aug (2002): 115-143.
[47] Liu, Gang, et al. "A new approach for synthesis and recognition of large scale handwritten Chinese words." *2010 12th International Conference on Frontiers in Handwriting Recognition*. IEEE, 2010.
[48] Jawahar, C. V., et al. "Retrieval of online handwriting by synthesis and matching." *Pattern Recognition* 42.7 (2009): 1445-1457.
[49] Varalakshmi, Aparna, Atul Negi, and Sai Krishna. "DataSet generation and feature extraction for Telugu hand-written recognition." *Int. J. Comput. Sci. Telecommun.* 3.3 (2012): 57-59.
[50] Siddiqi, Imran, and Nicole Vincent. "Writer identification in handwritten documents." *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*. Vol. 1. IEEE, 2007.
[51] Plamondon, Réjean, and Sargur N. Srihari. "Online and off-line handwriting recognition: a comprehensive survey." *IEEE Transactions on pattern analysis and machine intelligence* 22.1 (2000): 63-84.