University of Southern California
Ming Hsieh Department of Electrical and Computer Engineering

EE599 Deep Learning – Handwriting synthesis using recurrent neural networks (RNN)

Project Report

Submitted By: Team 8

USC ID_1: 8791018480 (Aagam Shah)

USC ID_2: 2334973997 (Shreya Kate)

USC ID_3: 8645057576 (Maipeng Zhang)

ON: 12/10/2020

1. Abstract:

Handwriting synthesis is an important subject in the field of deep learning. In this project, we used a Long Short-term Memory recurrent neural network to generate handwritten from text, simply by predicting one data point at a time. This model can revise colours according to strokes, by introducing biasing. In order to mimic different styles, we 'prime' the network with a real sequence.

2. Introduction:

As we know that the Children learn writing by practicing repeatedly to master their own writing styles. This process is very much similar for any kind of human being to learn the activity. Even with the advancements of the Deep Learning algorithms, computers are still incapable to learn a task by themselves. We can understand in a better way that how humans think by solving simple problems such as handwriting task, in order to develop better algorithms for the computers.

The model basically imitates a particular style of writing for the handwriting generation. The basic objective of our project is to develop a system which will transform the input text to the handwritten script. Normally, the process deal with the handwriting generation, whereas in this project we aim for synthesis of handwriting i.e., recognition and generation. There is one issue with the Residual Network which is about the Skip Connections. Skip connections basically, skip between multiple layers so it fails to produce accurate results. The network predictions are generally based on the last few of the inputs which are predicted by the network which leaves very small room for the network to imitate the handwriting. This would lead to reduction in performance accuracy as well as the increase in the conversion time.

The model which we have proposed consist of the Long Short-Term Memory recurrent neural networks responsible for the generation of the complex sequences with long range structure by simply predicting a single data point at a time.

So, basically, we implement Recurrent Neural Network to generate the handwriting for the given text. RNNs can be trained for sequence generation by processing real data sequences one step at a time and predicting what comes next.[2] A Recurrent Neural Network, is a class of artificial Neural Networks where connections between nodes form a directed graph along a temporal sequence. Unlike Feedforward Neural Networks, RNNs can use their internal state (memory) to process sequences of inputs. Deep LSTM cells can be used along with a Mixture Density Network to generate artificial handwriting data. [1]

3. Methodology:

3.1 Network Architecture:

Circles represent layers, solid lines represent connections, dashed lines represent predictions.

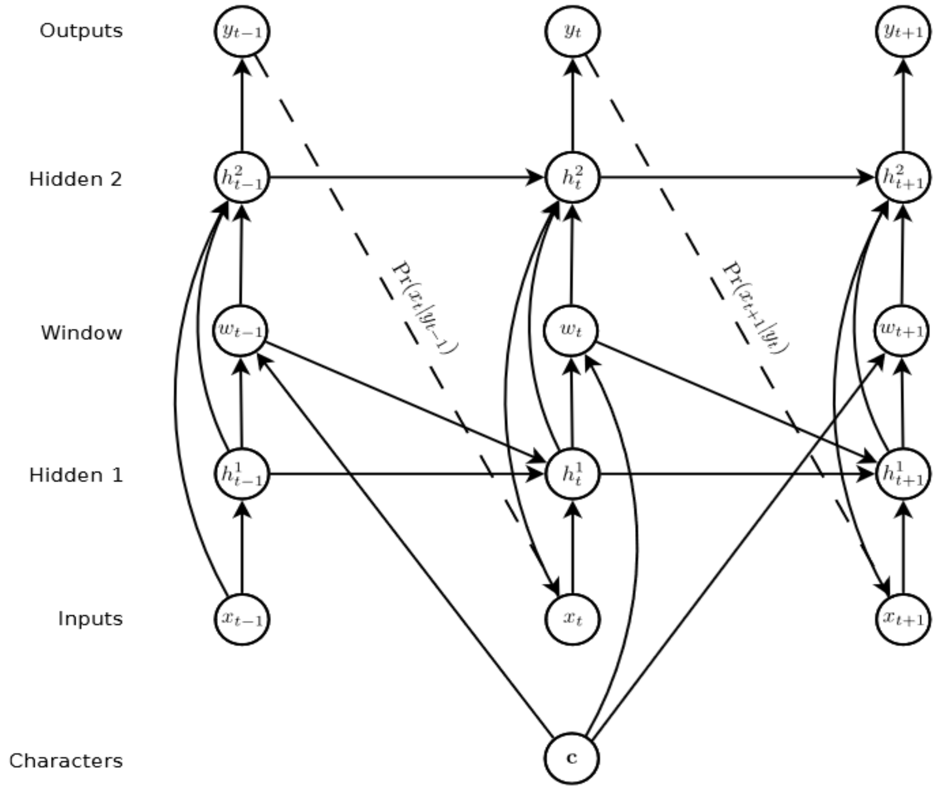


Fig. 1: Synthesis Network Architecture (taken from [1])

The above figure illustrates the basic recurrent neural network prediction architecture 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 = (h n 1 , . . . , hn T) and then the output vector sequence y = (y1, . . . , yT). Each output vector yt is used to parameterise a predictive distribution Pr(xt+1|yt) over the possible next inputs xt+1. The first element x1 of every input sequence is always a null vector whose entries are all zero; the network therefore emits a prediction for x2, the first real input, with no prior information. 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 makes it easier to train deep networks, 3 by reducing the number of processing steps between the bottom of the network and the top, and thereby mitigating the ‘vanishing gradient’ problem. [1]

Long Short-Term Memory

In most RNNs the hidden layer function H is an elementwise application of a sigmoid function. The Long Short-Term Memory (LSTM) architecture, uses purpose-built memory cells to store information, is better at finding and exploiting long range dependencies in the data.[6]

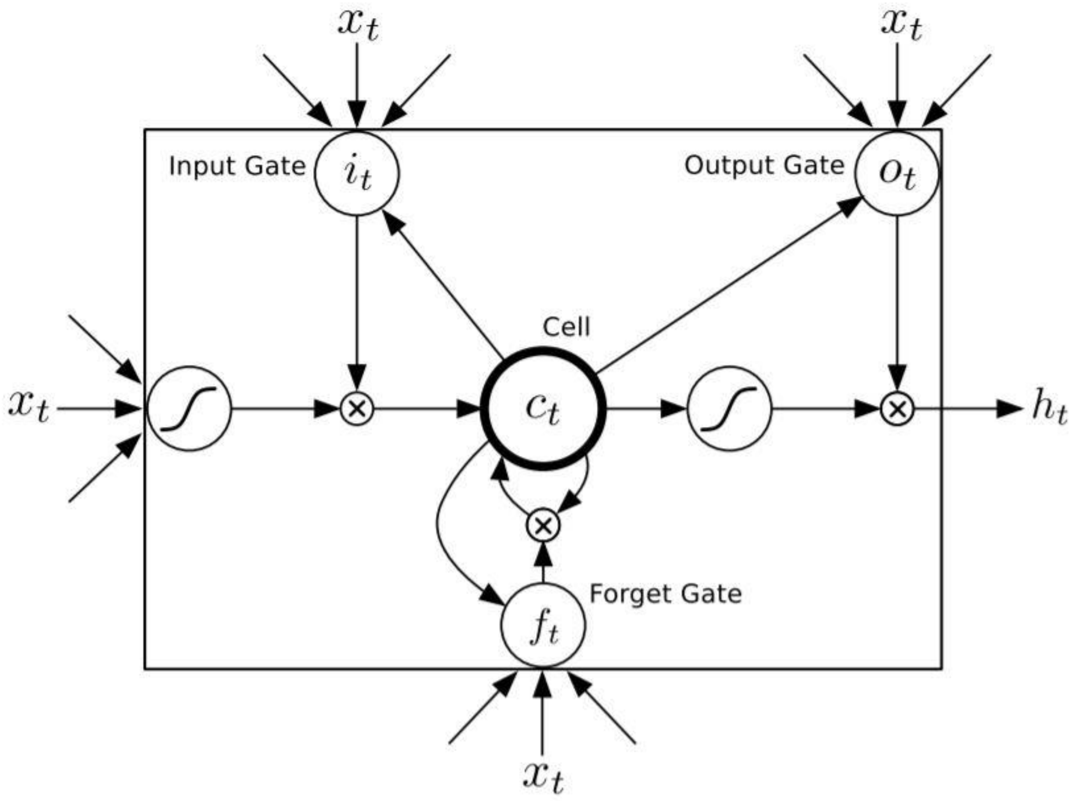


Fig. 2: Long Short-term Memory Cell

A ‘soft window’ is convolved with the text string and fed in as an extra input to the prediction network so that the network knows which character to print. The parameters of the window are output by the network at the same time as it makes the predictions, so that it dynamically determines an alignment between the text and the handwriting pen locations. This basically means that it learns to decide which character to write next.[4]

The hidden layers which have LSTM cells are stacked on top of each other. Each hidden layer feeds up to the layer above, and there are connections from the inputs to all hidden layers and from all hidden layers to the outputs.

The extra input from the character sequence, is presented to the hidden layers via the window layer. There is a delay in the connection to the first hidden layer to avoid a cycle in the network.[1]

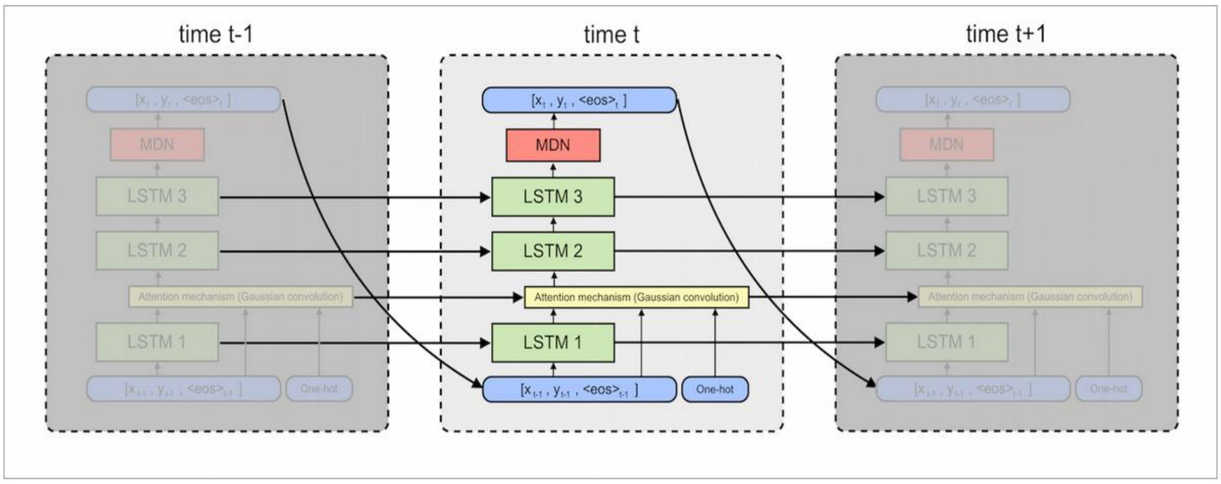


Fig. 3: The recurrent structure allows the model to feed information forward from past iterations. Arrows represent how data flows through the model (gradients flow backwards) [5]

RNNs are extremely good at modelling sequential data, so they are very good for synthesizing handwriting.
Sequence loss is calculated to keep a track of how well each step in an epoch is synthesizing the handwriting.

$$\mathcal{L}(\mathbf{x}) = -\log \Pr(\mathbf{x}|\mathbf{c})$$

Synthesis Network:

The figure 1 represents the network architecture used for handwriting synthesis. 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. The difference is the added input from the character sequence, mediated by the window layer. Given a length U character sequence c and a length T data sequence x, the soft window wt into c at timestep t (1 ≤ t ≤ T) is defined by the following discrete convolution with a mixture of K Gaussian functions.

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

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

where φ(t, u) is the window weight of cu at timestep t. 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[3]

3.2 Implementation of the Code:

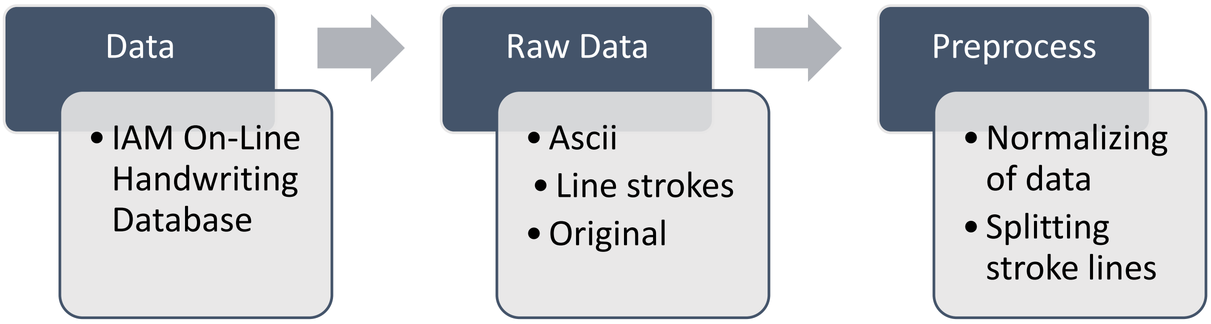
Phase 1:

(Effective data collection and pre-processing leading to huge reduction in computational time)

Step 1: In this part of implementation, i.e., phase 1 first of all we collect the data from the IAM Online Handwriting dataset which contains various forms of handwritten English text acquired on a whiteboard. The dataset is used for training as well as for the validation of handwritten text recognizers in order to perform the task of handwriting synthesis.

Step 2: From this dataset we just use the **Original** dataset as it contains all forms of strokes and styles. Infact it contains all forms of variations with the same as that of the online handwriting information but without the transcription and the writer-id. This is the data generation step.

Step 3: As the data collected from the online handwriting dataset is in the raw form so it demands for the data pre-processing step which deals with the normalization, processing of stroke sequences, distancing and separation.



Phase 2:

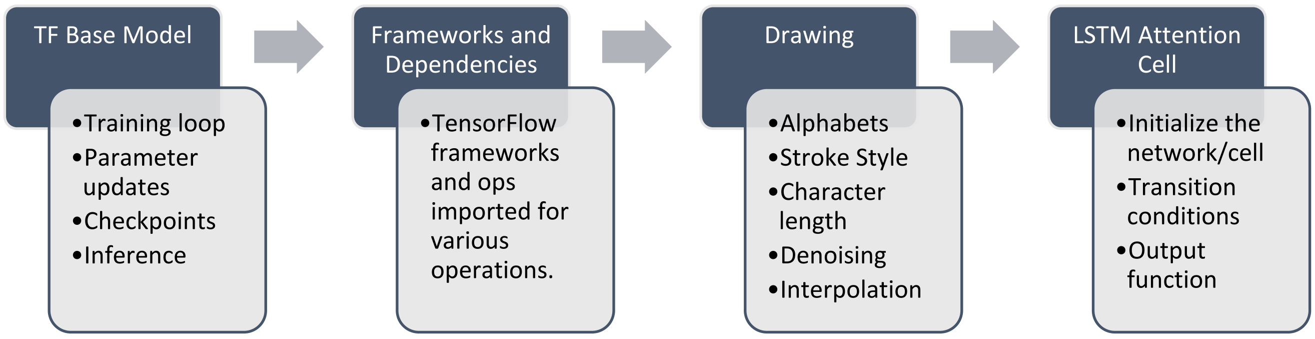
Once the data is prepared and preprocessed we move to the second phase of implementation of code. Here we define the base model, drawing styles and the LSTM-RNN network.

Step 1: First of all, we defined all the alphabets and numbers from 0 to 9 and stored it in the ordered form of dictionary. Then, next we define another function which will take care of the correction of the global offset to the strokes to mimic the effect of random nature. Then we have also implemented the denoising function which uses the smoothing filter to mitigate some artifacts of the data collection. Lastly, the interpolation function is defined which interpolates the strokes using the cubic spline.

Step 2: Now we create the base model for the for training of the tensorflow model. This class will take care of the training loop, updation of the parameters, checkpoiniting and the inferences. These subclasses are mainly responsible for building the computational graph beginning with the placeholders and ending with loss of tensor.

Step 3: Various tensorflow's frameworks and dependencies are used to carry out several math operations, tensorflow operations, array operations and the flow control operations.

Step 4: Lastly, in this phase I define the LSTM-RNN network, also known as the LSTM attention cell which takes care of various termination condition, length of the sequence to be processed, priming and biasing conditions, stroke styles and the pace of processing of each stroke or character.
The reason it is called as Attention cell because the LSTM network has the attention encoders and decoders present at the start and the end of the network respectively, and the attention cell sequentially processes single data point at a time which make the prediction of the results very accurate.



Phase 3:

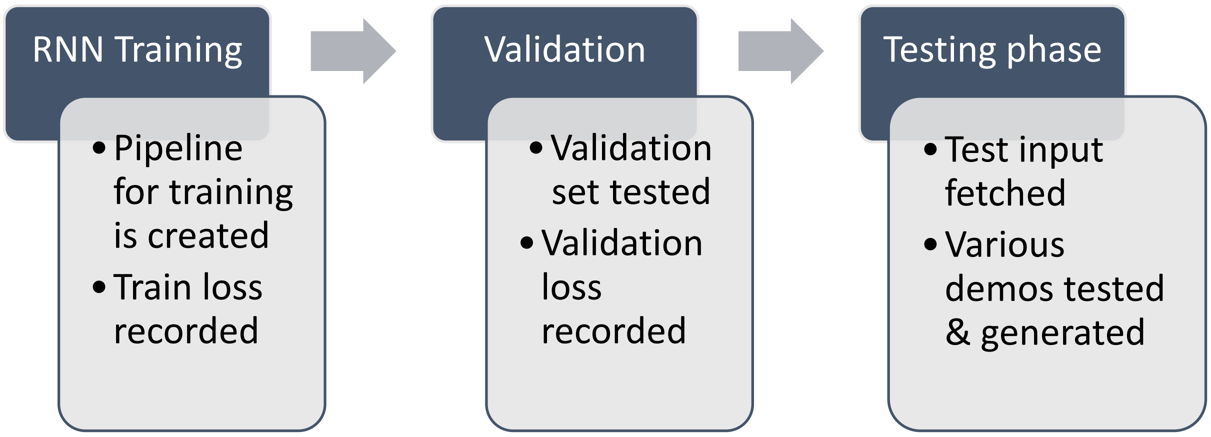
Step 1: In this step of phase 3 we perform the training and validation of the model using the data obtained from the phase 1. The model will receive the Textual Data from the IAM Handwriting Database. Then the textual data is processed using LSTM Cells and MDN. They are further sampled to generate accurate output result.

Step 2: The LSTM based generation and sampling will generate handwriting for a given text using RNN and LSTM. RNN will process the data through many layers and LSTM is used to recognize the sequence of words and generates handwritten data.

Step 3: LSTMs have an additional state called ‘cell state’ through which the network makes adjustments in the information flow. The advantage of this state is that the model can remember or forget the leanings more selectively. The Input Layer takes the sequence of words as input. The LSTM Layer computes the output using LSTM units.

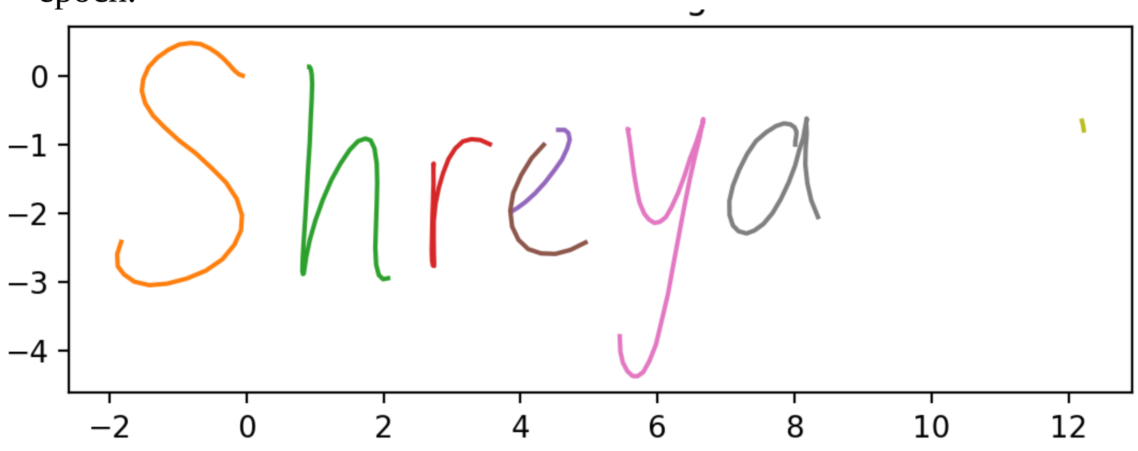
Step 4: The Dropout Layer is a regularization layer which randomly turns-off the activations of some neurons in the LSTM layer. It basically, helps in preventing over fitting.

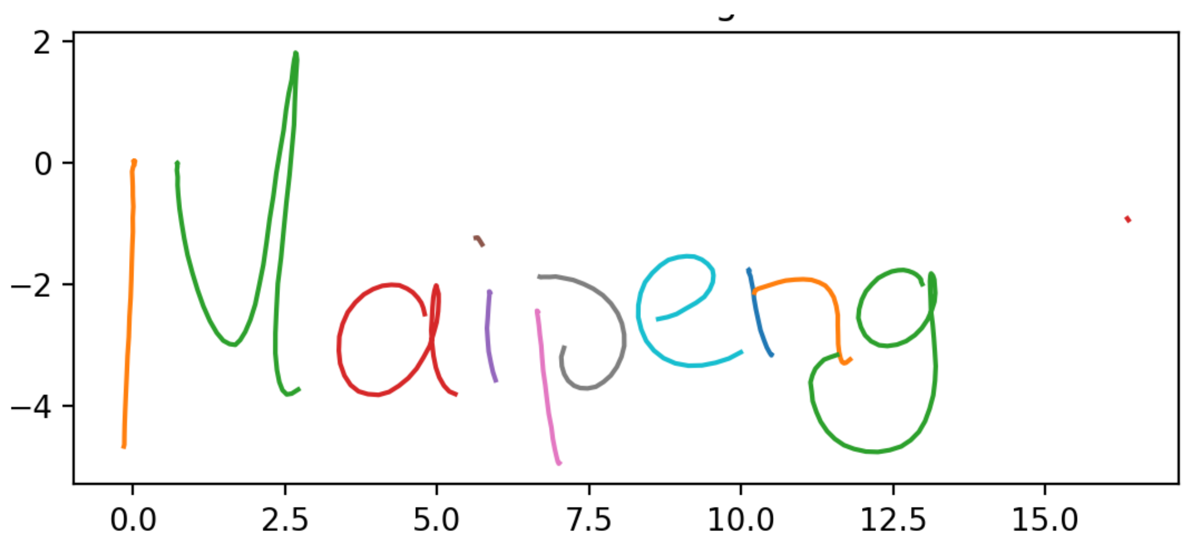
Step 5: The output can be obtained in real-time once the test sequence of the predefined length is fed into the system. The legibility and the diversity of the characters needs to be in good balance to visualize a good and readable output.

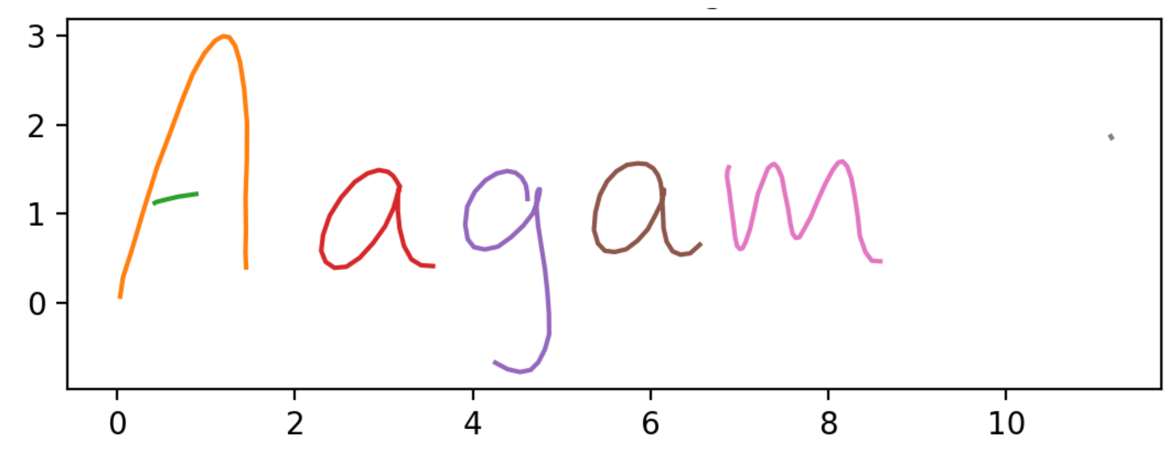


4. Experiments/Results:

These are the results after training for 30 epochs with 1000 steps in each epoch.

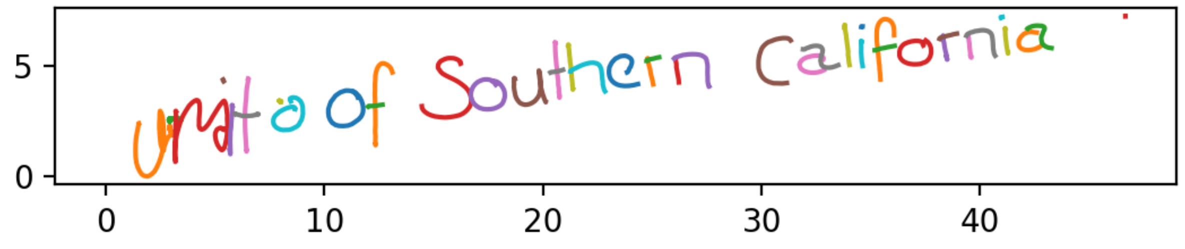




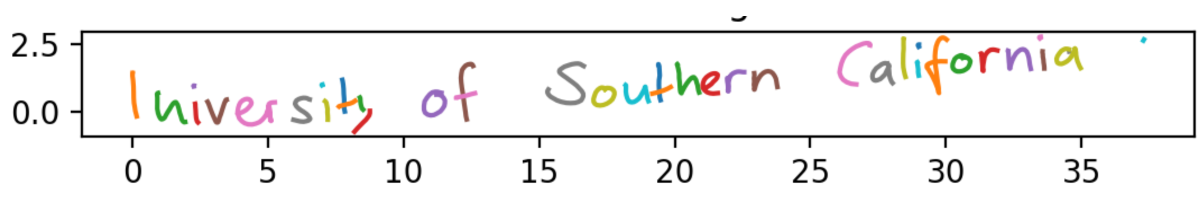


Each stroke is shown with a different color to visualize how the handwriting is synthesized. The network has been trained to synthesize handwriting in 8 different styles. These are displayed below:

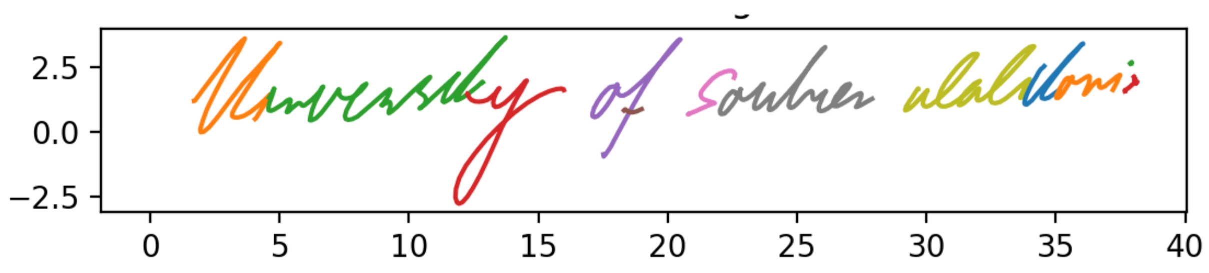
Style 0



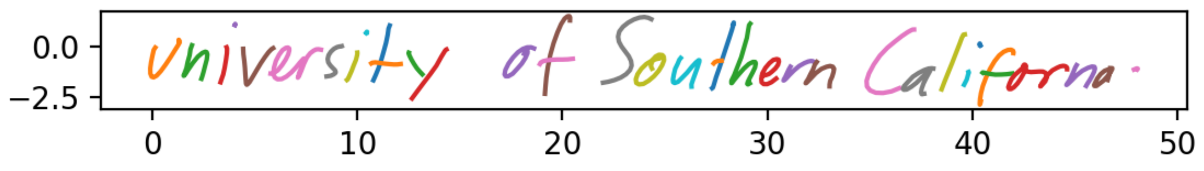
Style 1



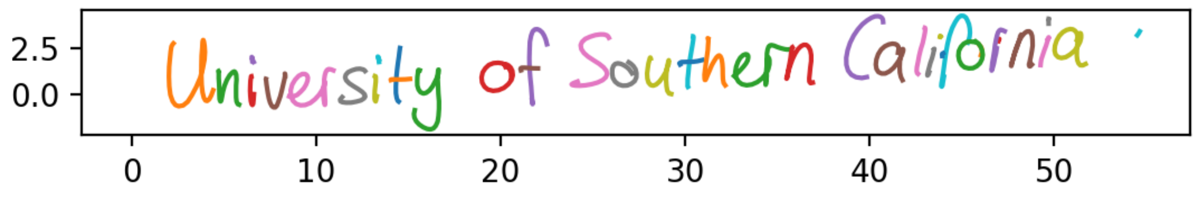
Style 2



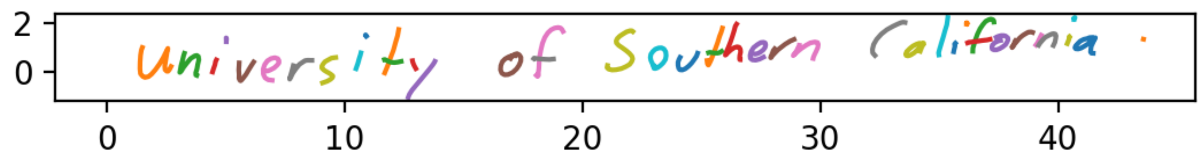
Style 3



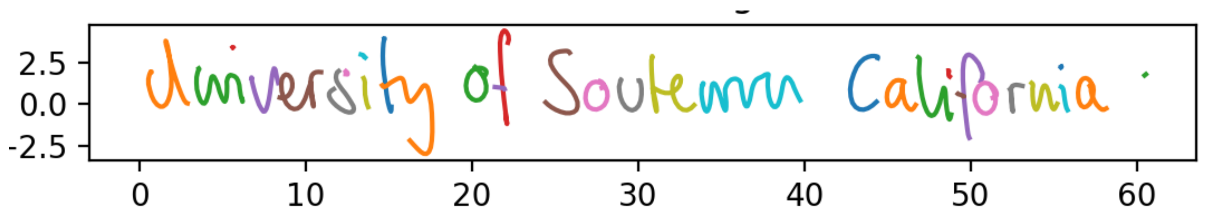
Style 4



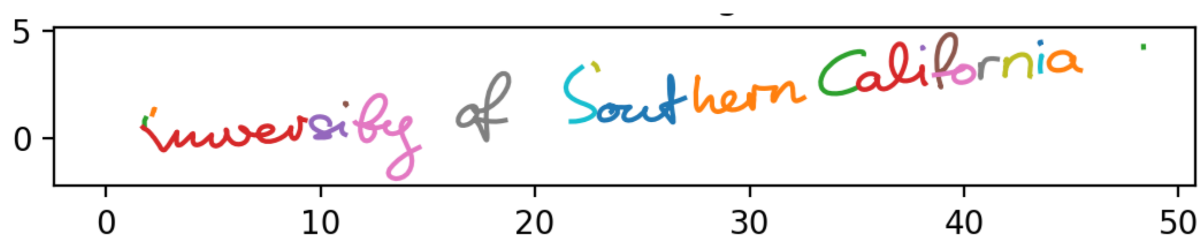
Style 5



Style 6



Style 7



The loss function for training is shown:

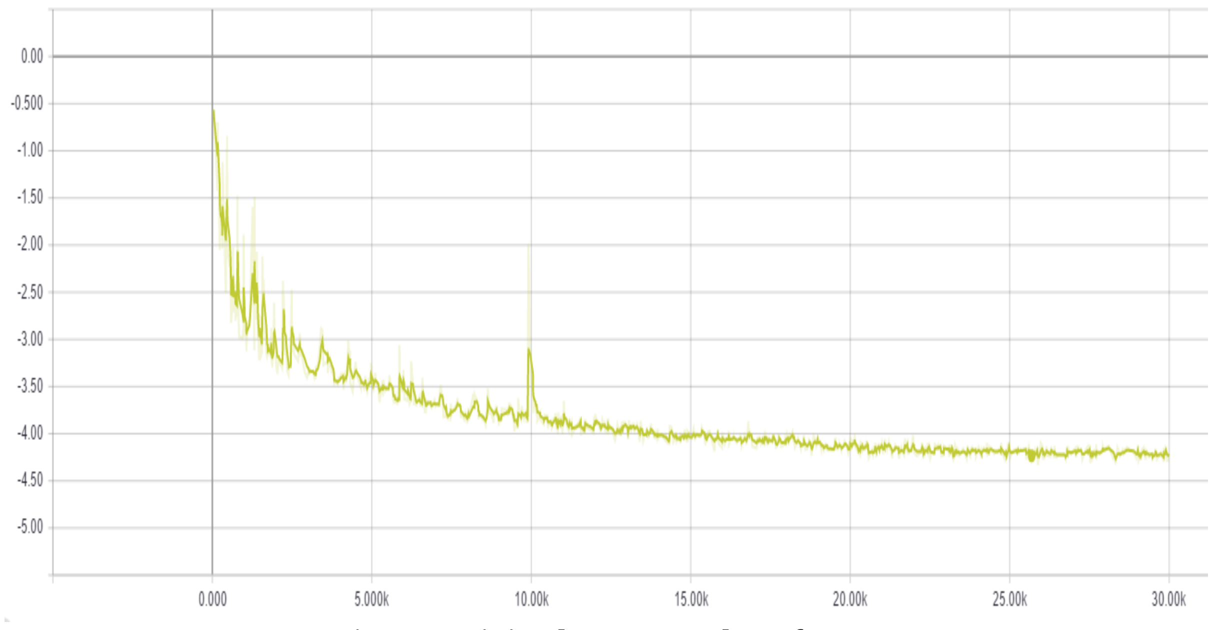
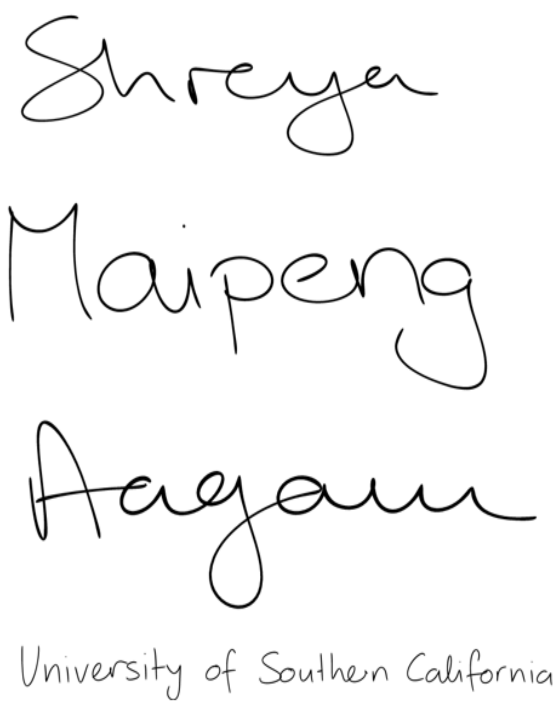


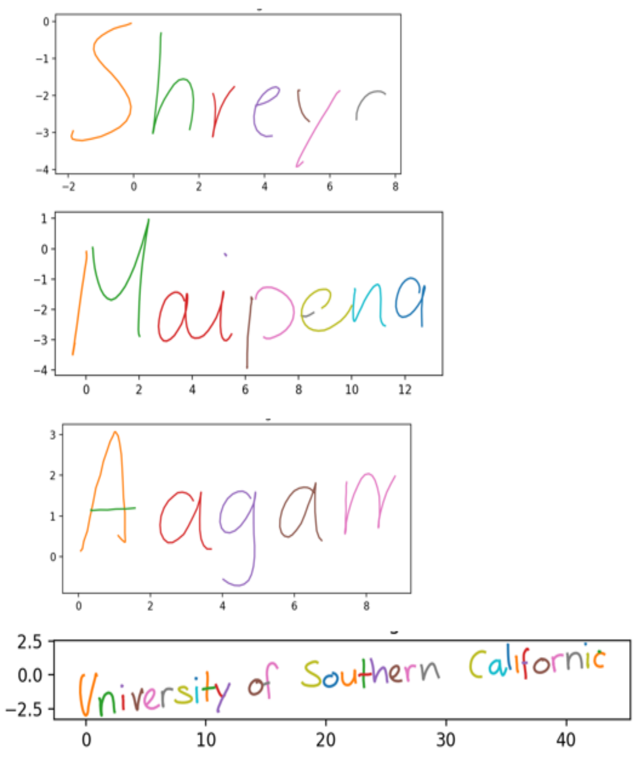
Fig. 3: Training loss vs. number of steps

Comparison with original paper results:

Original paper results



Our results



Inferences:

1. Original paper had 10000 epochs and we trained for 30 epochs, thus reducing computational time. It reduced the time from 48 hours to 10 hours for training on the same computational resources on an EC2 p3 instance.
2. Original paper used more data, we only used part of the data for training to get comparable results
3. Reduction in computational time at the cost of marginal decrease in accuracy

5. Conclusion:

In conclusion, this project shows how to use Long Short-term Memory neural network to generate handwritten fonts with various strokes and styles, simply by processing the actual data sequence step by step at a time to predict what will happen next. After pre-processing the data (Splitting stroke lines and Normalizing) and reducing the computational time by reducing the training epochs, we finally got a feasible code to achieve the above goal. In order to make our model be a good 'writer', we introduce biases to make the handwriting more readable. Furthermore, to be able to introduce a personal writing style into the network without incurring too much cost, we 'prime' the network with a real sequence, then generating an extension.
We compared our results with the models proposed in the original paper. The resulting sequences are convincing enough that they are indistinguishable from the real handwriting. Our model can also distinguish each stroke by the colour, just like a human writer.

6. References:

1. Handwriting Synthesizing and Generation Using Deep Recurrent Neural Network, Durairaj S , Samuel J John , Santhosh M , Dr. Chakaravarthi S
2. Alex Graves. Generating Sequences With Recurrent Neural Networks.
3. Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is diffiffifficult. IEEE Transactions on Neural Networks, 5(2):157–166, March 1994.
4. F. Gers, N. Schraudolph, and J. Schmidhuber. Learning precise timing with LSTM recurrent
5. M. Liwicki and H. Bunke. IAM-OnDB - an on-line English sentence database acquired from handwritten text on a whiteboard. In Proc. 8th Int. Conf. on Document Analysis and Recognition, volume 2, pages 956– 961, 2005.

6. I. Sutskever, J. Martens, and G. Hinton. Generating text with recurrent neural networks. In ICML, 2011.

Link for the Repository:

https://github.com/Aagamshah9/EE_599_Deep_Learning_Final_Project_Team-8

Thank You !!