Handwriting Synthesis and Text Prediction using Recurrent Neural Networks

Yash Shah, *160050002*, R. Sudarsanan, *160050067*

***Abstract*—This report shows the application of a type of recurrent neural network (LSTM) to the task of online handwriting generation and text prediction. We use an ensemble of two deep LSTM networks, one trained to write with a realistic cursive handwriting by predicting real-valued data points one at a time, and the second trained to generate sequences of discrete words given a prior context. We aim to build a model that can write on its own by using the outputs of the second network as inputs for the first. Our model is a combination of two seemingly different models (by *Alex Graves* and *Mikolov et al.*) working together to accomplish a more difficult task of generating real and discrete valued sequences having long-range structure.**

***Index Terms*—handwriting synthesis, language modeling, LSTM, mixture density network, attention window.**

I. Introduction

RECURRENT Neural Networks (RNNs) have been widely used for sequence modeling in various fields, especially Natural Language Processing, due to their ability to 'remember' patterns and structure over a long range. Long Short Term Memory (LSTM) networks are a modification of vanilla RNNs designed to cope better with the problem of vanishing gradients when the model is unrolled over a long time interval. Given an input x_t, hidden state h_{t-1} and cell state c_{t-1}, a LSTM cell generates the outputs for timestep t according to these equations:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t1} + W_{ci}c_{t1} + b_i) \quad (1)$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t1} + W_{cf}c_{t1} + b_f) \quad (2)$$

$$c_t = f_t \cdot c_{t1} + i_t \cdot tanh(W_{xc}x_t + W_{hc}h_{t1} + b_c) \quad (3)$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t1} + W_{co}c_t + b_o) \quad (4)$$

$$h_t = o_t \cdot tanh(c_t) \quad (5)$$

Another important aspect to look upon (especially in our scenario) is the nature of outputs 'required'. Most supervised learning algorithms deal with classification problems, for which the outputs are either discrete-valued or chosen to be Gaussian. The task of handwriting synthesis, along with several others, requires the model to predict non-Gaussian or multivariate outputs. Such *inverse problems* cannot be modeled simply by using the Gaussian assumption as it would lead to poor predictions. In order to tackle such problems, *mixture density networks* are used.

If Gaussian components are used in the MDN, the probability distribution can be expressed as:

$$p(t|x) = \sum_{k=1}^{K} \pi_k(x) \cdot \mathcal{N}\left(t|\mu_k(x), \sigma_k^2(x)\right) \quad (6)$$

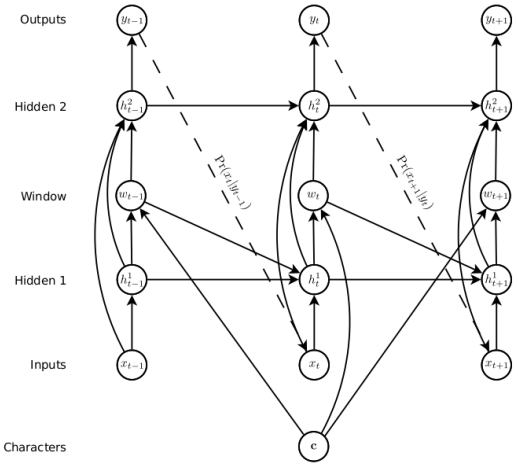


Fig. 1. *Source: Alex Graves' paper: Generating Sequences With Recurrent Neural Networks [1]*

We have used both LSTMs and MDNs in our model to effectively handle discrete words and real-valued, bivariate Gaussian outputs respectively.

II. Model Architectures

As forementioned, our model is a combination of two independent models, a *Prediction Model* for generating sentences given a prior context, and a *Synthesis Model* for generating handwriting given an input string. Both of them are described in detail below.

A. Synthesis model

Basic recurrent neural network prediction architecture is used here. Input consists of the start point for the pen in the 2D plane and the one hot encoded string. The implemented network has two LSTM hidden layer, a window layer and a MDN layer. The hidden layers are stacked on top of each other with window layer between the 1^{st} and the 2^{nd} hidden LSTM layers. There are skip connections from the input to all hidden layers and from all hidden layers to the MDN layer except from the window layer. The final output of a particular timestep becomes the input of the next timestep.

1) Input layer: The input starting point is represented as a 3×1 vector denoting the x-coordinate and the y-coordinates relative to the previous point and the end-of-stroke probability. The value of the end-of-stroke is 1 when the pen is up denoting a discontinuity in the line. The string is converted to a one-hot vector $\mathcal{C}$ with a vector mapping for each character. The encoded string and the point are fed as input to network.

2) Window layer: It takes input form the 1st hidden lstm layer and the encoded string as the input. Given a length U character sequence **c** and the max timestep T, the soft window w_t for the timestep t ($1 \leq t \leq T$) is defined by the following discrete convolution with a mixture of **K** Gaussian functions.

$$\phi(t,u) = \sum_{k=1}^{K} \alpha_t^k exp(-\beta_t^k(\kappa_t^k - u)^2) \tag{7}$$

$$w_t = \sum_{u=1}^{U} \phi(t,u) c_u \tag{8}$$

Here $\phi(t,u)$ is the window weight of c_u at timestep t. The κ_t parameters control the location of the window, the β_t parameters control the width of the window and the α_t parameters control the importance of the window within the mixture. The window weight $\phi(t,u)$ can be loosely interpreted as the networks belief that it is writing character c_u at time t. The size 3**K** vector **p** of the window parameters is detemined as follows by the outputs of the first hidden layer of the network:

$$(\hat{\alpha_t}, \hat{\beta_t}, \hat{\kappa_t}) = W_{h^1 p} h_t^1 + b_p \tag{9}$$

$$\alpha_t = exp(\hat{\alpha_t}) \tag{10}$$

$$\beta_t = exp(\hat{\beta_t}) \tag{11}$$

$$\kappa_t = \kappa_{t-1} + \hat{\alpha_t} \tag{12}$$

3) First lstm layer: This layer has incoming links for the input, output of the window layer of the previous timestep and output of the same layer in the previous timestep. The update equation is given by:

$$h_t^1 = LSTM(W_{ih^1} x_t + W_{h^1 h^1} h_{t-1}^1 + W_{wh^1} w_{t-1} + b_h^1) \tag{13}$$

where the LSTM(.) function denotes that this is fed as the input to the LSTM cell.

4) n^{th} hidden lstm layer: All the lstm layer other than the 1st lstm layer have input, output of the previous lstm layer, output of the window layer and their own output of the previous timestep as input. The update equation is given by:

$$h_t^n = LSTM(W_{ih^n} x_t + W_{h^{n-1} h^n} h_t^{n-1} + W_{h^n h^n} h_{t-1}^n + W_{wh^n} w_t + b_h^n) \tag{14}$$

5) MDN layer: This layer has connections from all the hidden lstm layers and the 1 + 6M parameters i.e. $\hat{\epsilon_t}$ and the M sets $\{\hat{\pi_t^j}, \hat{\mu_{xt}^j}, \hat{\mu_{yt}^j}, \hat{\sigma_{xt}^j}, \hat{\sigma_{yt}^j}, \hat{\rho_t^j}\}_{j=1}^M$ are extracted from the outputs of the hidden layers concatenated together through matrix multiplications. The output vector is calculated as follows:

$$e_t = \frac{1}{1 + exp(\hat{\epsilon_t})} \Rightarrow \epsilon_t \in (0,1) \tag{15}$$

$$\pi_t^j = \frac{exp(\hat{\pi_t^j})}{\sum_{j'=1}^{M} exp(\hat{\pi_t^j})} \tag{16}$$

$$\mu_t^j = \hat{\mu_t^j} \tag{17}$$

$$\sigma_t^j = exp(\hat{\sigma_t^j}) \tag{18}$$

$$\rho_t^j = tanh(\hat{\rho_t^j}) \tag{19}$$

During training, the probability density $Pr(x_{t+1}|y_t)$ of the next input x_{t+1} given the output vector y_t is defined as follows:

$$\sum_{j=1}^{M} \pi_t^j \mathcal{N}(x_{t+1}|\mu_t^j, \sigma_t^j, \rho_t^j) \begin{cases} \epsilon_t & \text{if } (x_{t+1})_3 = 1 \\ 1 - \epsilon_t & \text{otherwise} \end{cases} \tag{20}$$

where

$$\mathcal{N}(x|\mu,\sigma,\rho) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} exp[\frac{-Z}{2(1-\rho^2)}] \tag{21}$$

$$Z = \frac{(x_1-\mu_1)^2}{\sigma_1^2} \frac{(x_2-\mu_2)^2}{\sigma_2^2} - \frac{2\rho(x_1-\mu_1)(x_2-\mu_2)}{\sigma_1\sigma_2} \tag{22}$$

When this model is used for generating handwriting, one of the gaussians is chosen from the given set with the probabilities π and then a point is sampled from it.

6) Loss calculation: We have used log-likelihood loss $\mathcal{L}(x)$ over the output of the MDN layer as follows:

$$\sum_{t=1}^{T} -log(\sum_j \pi_t^j \mathcal{N}(x_{t+1}|\mu_t^j, \sigma_t^j, \rho_t^j)) - \begin{cases} \log \epsilon & \text{if } (x_{t+1})_3 = 1 \\ \log(1-\epsilon_t) & \text{otherwise} \end{cases} \tag{23}$$

B. Prediction model

This model is a vanilla implementation of a similar one introduced by *Mikolov et al*. It sequentially generates a predictive probability distribution, given a word and surrounding context, spanning the entire training vocabulary and 'picks' the next word based on its probability mass. The picked word is used as the input for predicting the remain sequence and this is repeated till the *end-of-sentence* token is obtained.

1) Embedding layer: This layer performed the task of converting word indices to *word vectors* by multiplying the corresponding one-hot vectors by an embedding matrix W_e. This matrix was initialized randomly, and it learnt the word vectors by itself as the network trained. This *word vector* was fed to the LSTM network.

$$x_t = W_e \cdot U_{w_t} \tag{24}$$

where $U_{w_t} \in \mathcal{R}^{V \times 1}$ is the one-hot representation of input word at time t.

2) Hidden LSTM layers: This layer basically contains stacked LSTM cells enclosed within *dropout* wrappers. This wrapping made the model more robust and easily generalizable. We had used *dropout* over four parameters - input to LSTM stack, output of one LSTM layer that was fed to the next layer, hidden state and final output. Denoting the *dropout* masks as $\mathbf{D}_i$, $\mathbf{D}_j$, $\mathbf{D}_s$ and $\mathbf{D}_o$ respectively, the equations for outputs at time t become:

$$a_1 = \text{LSTM}(\mathbf{D}_i \odot \mathbf{x}_t; \mathbf{D}_s \odot \mathbf{h}_{t-1}; \mathbf{D}_s \odot \mathbf{C}_{t-1}) \tag{25}$$

$$a_k = \text{LSTM}(\mathbf{D}_j \odot \mathbf{a}_{k-1}; \mathbf{D}_s \odot \mathbf{h}_{t-1}; \mathbf{D}_s \odot \mathbf{C}_{t-1}) \tag{26}$$

$$o_t = \mathbf{D}_o \odot \text{LSTM}(\mathbf{D}_j \odot \mathbf{a}_{n-1}; \mathbf{D}_s \odot \mathbf{h}_{t-1}; \mathbf{D}_s \odot \mathbf{C}_{t-1}) \tag{27}$$

3) Output layer and loss calculation: We used the standard *cross-entropy loss* over the expected output (which is a one-hot vector representation of the expected word) and the logits obtained after applying *softmax* function on the activations of the previous layer, that was summed over all time-steps. Thus,

$$\mathcal{L}(\mathbf{x}) = \sum_{t=1}^{T} \sum_{i=1}^{V} -\mathbf{y}_{it} \cdot log(\mathbf{o}_{it}) \tag{28}$$

$$\mathbf{o}_{it} = \frac{\exp(W_o \mathbf{a}_{it} + b_o)}{\sum_{j=1}^{V} \exp(W_o \mathbf{a}_{jt} + b_o)} \tag{29}$$

III. Implementation

We used `python3` language for implementing all the code for program. `tensorflow-gpu` package was used for training the network. The synthesis model was trained using preprocessed IAM Online Handwriting dataset, and the prediction model was trained using the PTB dataset.

A. Synthesis model

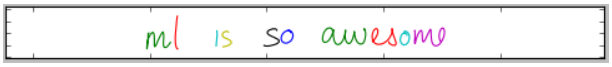
In the generation model two LSTM layers were used with LSTM size of 200. The window layer and the MDN layer had sets of 10 and 20 gaussians respectively. Adam optimizer was used for learning and the starting learning rate was 1e-3 and decaying to half every thousand steps. The network was trained for 200 epochs which takes approximately 6hrs to train using GPU.

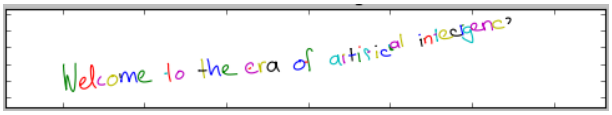
B. Prediction model

The prediction model network had 2 hidden LSTM layers of size 650. The input, intra, state and the output drop probabilities are 0.5, 0.5, 0 and 0.5 respectively. SGD optimizer was used and the network was trained for 50 epochs, which took roughly 8hrs to train on GPU.

IV. Results

The following handwriting samples were obtained by plotting the outputs of the synthesis model:





These samples were generated by the prediction model:

- Prior sentence: **a girl likes**, number of words: **8**
  `a girl likes fiber fossil jittery guber-peters express line-item ambrosiano accords`
- Prior sentence: **he was**, number of words: **5**
  `he was matching neatly punishment colombian tramp`
- Prior sentence: **the sun**, number of words: **7**
  `the sun retains alexander admits suspension destroyed intend really`

As we can see, the prediction model gives less meaningful sentences. This is because the dataset which we used (PTB) is quite small and the model implementation is quite basic. Moreover, the generated sentences are rarely punctuated, making them even less meaningful.

V. Conclusion

This project served as a great learning experience for us. We learnt in detail about the working of and math behind recurrent neural networks, LSTMs in particular, and their usage in generating discrete and real valued sequences. We also learnt how to tackle *inverse problems*, the problems where non-Gaussian or multivariate output is required, using a special variant of Mixture Density Networks, *Mixture of Gaussians*. We were also introduced to the concept of **attention**, and its application to this task.

Acknowledgment

We would like to sincerely thank our professor, Prof. Amit Sethi, for his constant support and guidance throughout this project and for providing us an opportunity to explore and apply the knowledge gained in the course to practical scenarios.

References

[1] G. Alex, *Generating sequences with Recurrent Neural Networks*, 2014.
[2] Mikolov et al., *Recurrent Neural Network based Language Model*, 2010.
[3] C. Bishop, *Pattern Recognition and Machine Learning*, 2006.