Realistic Handwriting Generation Using Recurrent Neural Networks and Long Short-Term Networks

Suraj Bodapati, Sneha Reddy and Sugamya Katta

Abstract Generating human-like handwriting by machine from an input text given by the user may seem as an easy task but is very complex in reality. It might not be possible for every human being to write in perfect cursive handwriting because each letter in cursive gets shaped differently depending on what letters surround it, and everyone has a different style of writing. This makes it very difficult to mimic a person's cursive style handwriting with the help of a machine or even by hand for a matter of fact. This is why signing names in cursive is preferable on any legal documents. In this paper, we will try to use various deep learning methods to generate human-like handwriting. Algorithms using neural networks enable us to achieve this task, and hence, recurrent neural networks (RNN) have been utilized with the aim of generating human-like handwriting. We will discuss the generation of realistic handwriting from the IAM Handwriting Database and check the accuracy of our own implementation. This feat can be achieved by using a special kind of recurrent neural network (RNN), the Long Short-Term Memory networks (LSTM).

Keywords Handwriting generation · Recurrent neural networks (RNN) · Long Short-Term Memory networks (LSTM) · IAM handwriting database

S. Bodapati · S. Reddy (✉) · S. Katta
Department of Information Technology, Chaitanya Bharathi Institute of Technology, Hyderabad, India
e-mail: snehareddycsr@gmail.com

S. Bodapati
e-mail: Surajbodapati97@gmail.com

S. Katta
e-mail: ksugamya_it@cbit.ac.in

K. S. Raju et al. (eds.), *Proceedings of the Third International Conference on Computational Intelligence and Informatics*, Advances in Intelligent Systems and Computing 1090, https://doi.org/10.1007/978-981-15-1480-7_55

1 Introduction

The power of deep learning can be demonstrated by recurrent neural networks (RNNs) which can be viewed as a top class of dynamic models that is used to generate sequences in multiple domains such as machine translations, speech recognition, music, generating captions for input images and determining emotional tone behind a piece of text. RNNs are used to generate sequences by processing real data sequences and then predicting the next sequence. A large enough RNN is capable of generating sequences of subjective complexity. The Long Short-Term Memory networks (LSTMs) are a part of RNN architecture which is capable of storing and accessing information more efficiently than regular RNNs. In this paper, we use the LSTM memory to generate realistic and complex sequences [1].

The handwritten text in English used to train and test handwritten text recognizers and to perform identification and verification experiments of the writer is taken from the IAM Handwriting Database which contains forms of unconstrained handwritten text, which were saved as PNG images with 256 gray levels after being scanned at a resolution of 300 dpi [2]. This dataset is very small; about 50 Mb when parsed once. The dataset is a result of contribution from 657 writers, and each dataset has a unique handwritten style. The data used in this paper is a 3D time series with three coordinate axes. The (x, y) coordinates are the normal first two dimensions, whereas the third dimension is a binary 0/1 value. The 1 in this third coordinate has around 500 pen points and an annotation of ASCII characters which is used to signify the end of a stroke.

2 Concepts

2.1 Artificial Neural Networks

Artificial neural networks try to simulate the human brain by modeling its neural structure on a much smaller scale. When you consider a typical human brain, it consists of billions of neurons in contrast to ANN which has one by 1000th processor units of that of a human brain. A neural network consists of neurons, which are simple processing units, and there are directed, weighted connections between these neurons. A neural network has a number of neurons processing in parallel manner and arranged in layers. Layers consist of a number of interconnected nodes which contain an activation function. The patterns from the input layer are processed using the weighted connections, which help with the communication with the hidden layer. The output from one layer is fed to the next and so on. The final layer gives the output. For a neuron j, the propagation function receives the outputs of other neurons and transforms them in consideration of the weights into the network input that can be further processed by the activation function [3]. Every neuron operates in two modes: training and using mode.

2.2 Recurrent Neural Networks

The major difference between a traditional neural network and an RNN is that the traditional neural networks cannot use its reasoning about previous events in the process to inform the later neurons or events. Traditional neural networks start thinking process from scratch again. RNNs address this issue, by using loops in their network, thus making information persist. The RNN obtains input x_t and gives output value h_t [4]. The passing of information from one step of the network to the next is allowed by a loop. An RNN network can be taken as one of many copies where each one of the copy passes information to the previous. In RNNs, each word in an input sequence in RNN will be mapped with a particular time step. This results in the number of time steps equal to the maximum sequence length. A new component called a hidden-state vector h_t is associated with each time step, and this constitutes each iteration output. h_t seeks to encapsulate and summarize all of the information that was seen in the previous time steps when seen from a high level. In similar fashion, x_t is a vector that encapsulates all the information of a specific input word; the hidden-state vector at the previous time step is a function of both the hidden-state vector and current word vector. The two terms sum that will be passed through an activation function is denoted by sigma. Final hidden layer output can be given as $h_t = \sigma(W^{H*} h_{t-1} + W*^{X} x_t)$, where Wx is weight matrix to be multiplied with input x_t and is variable. W^H is recurrent weight matrix that is multiplied with hidden-state vector of earlier step. W^H is a constant weight matrix. These weight matrices are updated and adjusted via an optimizing process known as backpropagation through time. Sigma denotes the two terms sum is passed via an activation function, usually sigmoid/Tanh.

2.3 Long Short-Term Memory Networks (LSTMs)

Long Short-Term Memory networks (LSTMs) are capable of learning the long-term dependencies and are a special kind of RNN. LSTMs are modules that you can place inside an RNN which are explicitly used to avoid the long-term dependency problem. The long-term dependency problem can be described as a situation where the gap between the point where the network is needed and the relevant information becomes very large, and as a result, the RNNs become incapable to connect the previous information. The computation in LSTMs can be broken down into four components, an input gate which determines mount of emphasis to be put on each gate, a forget gate determines the unnecessary information, an output gate which will determine the final hidden-vector state based on intermediate states and a new memory container [5].

2.4 Backpropagation

A backpropagation neural network which is backward propagation of error uses the delta rule which is one of the most used rules of neural networks. A neural network assigns a random weight when it is provided with an input pattern, and later, based on its difference from output, it changes the weight accordingly. The delta rules use supervised method of learning that occurs with each cycle of epoch where error is calculated using backward error propagation of weight adjustments and forward flow of outputs. The network runs in forward propagation mode only once the neural network is trained. The new inputs are no longer used for training the network instead they are processed only in forward propagation mode to obtain the output. Backpropagation networks are usually slower to train when compared to some other types of networks and sometimes require a very large number of epochs.

2.5 One-Hot Encoding

One-hot encoding is used when categorical variables need to be converted into a form which can be given to the machine learning algorithms for regression or classification which helps in better prediction. For training the model, binarization of category is done using the one-hot encoder and including its features.

3 Model Implementation

The dataset used in this paper was originally used to as training dataset for handwriting recognition models, which inputs a series of pen points and identifies the corresponding letter [6]. Our goal is to reverse this model using RNNs and train a model which takes letters as inputs and produces a series of points which can then be connected to form letters. In this paper, we use a model which can be viewed as a three sub-models stacked on top of one another. The models mentioned are trained using gradient backpropagation (Fig. 1).

The model used above is a combination of three different types of models. The sequences of pen points of the input handwriting are produced using the RNN cells. The one-hot encoding of the input text is given to the attention mechanism. The style and variation in the handwriting can be generated by MDM which achieves this by modeling the randomness in the handwriting. The MDN cap of the model predicts the x, y coordinates of the pen by drawing them from the Gaussian distribution, and the handwriting can be varied by modifying the distribution. We start with the inputs and work our way upwards through the computational graph. Step 1: The first step in implementing this model is to download and preprocess the data. The script was written for preprocessing searches for local directory for the files

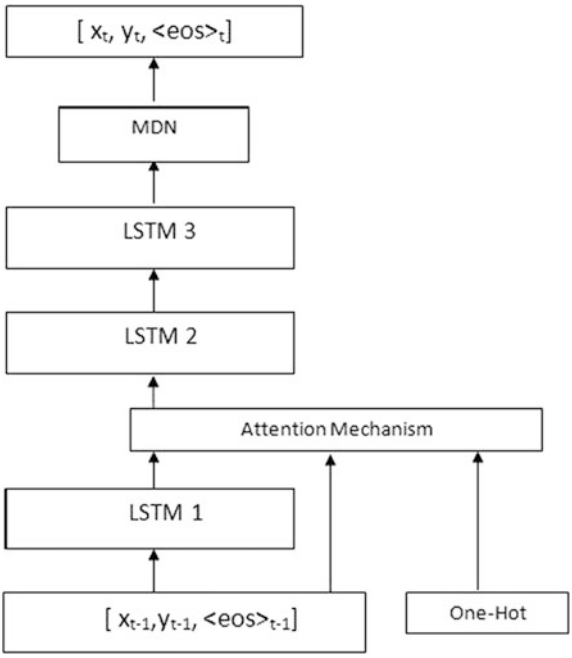


Fig. 1 Model overview

containing data we downloaded and performs preprocessing tasks such as normalizing the data, data cleaning, data transformations, splitting strokes in lines. Step 2: Build the model.

3.1 The Long Short-Term Memory Cells

This paper deals with three LSTM-RNN cells for developing a model. These three LSTM cells act as the backbone of the model. The above-mentioned three LSTM networks can keep the trail of independent patterns by using a differentiable memory. LSTMs used in this model use three different tensors to perform read, write and erase tasks on a memory tensor. We use a custom attention mechanism which digests a one-hot encoding of the sentence we want the model to write. We can add some natural randomness to our model with the help of the Mixture Density Network present on the top which chooses appropriate Gaussian distributions from which we are able to sample the next pen point. We start with the inputs and move upwards through the computational graph. Tensorflow seq2seq API is used to create the LSTM cells present in this model [7].

3.2 Attention Model

In this model, we use a differentiable attention mechanism to get the information about characters that make up a sentence. An attention mechanism can be described as a Gaussian convolution over a one-hot encoding of input texts using mixtures of Gaussians. As the model writes from character to character, it learns to shift the window. This is possible as the attention mechanism tells the model what is to be written and since all the parameters of the window are differentiable. The final output is a soft window into the one-hot encoding of the character the model thinks it is drawing. A heat map is obtained on stacking these soft windows vertically over time. This attention mechanism obtains inputs by looping with the output states of LSTM-1. Next, we concentrate the outputs of this attention mechanism to LSTM's state vector and concatenate the original pen stroke data for good measure (Figs. 2 and 3).

3.3 Mixed Density Networks (MDN)

Randomness and style in handwriting in this model are generated by using mixture density networks. MDNs can parameterize probability distributions hence are a very good method to capture randomness in the data. At the beginning of the strokes, a Gaussian with diffuse shapes is chosen, and another Gaussian is chosen at the middle of strokes with peaky shapes. They can be seen as neural networks which can measure their own uncertainty (Fig. 4).

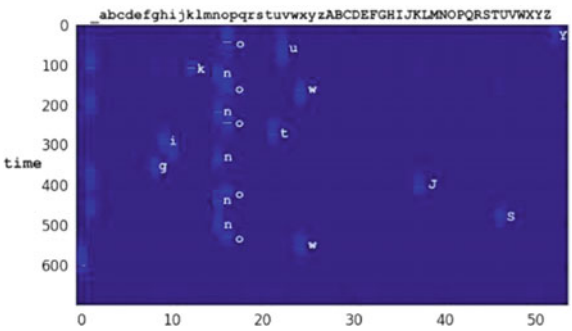


Fig. 2 Heat map by stacking soft windows vertical over time

Fig. 3 Modified LSTM State vector

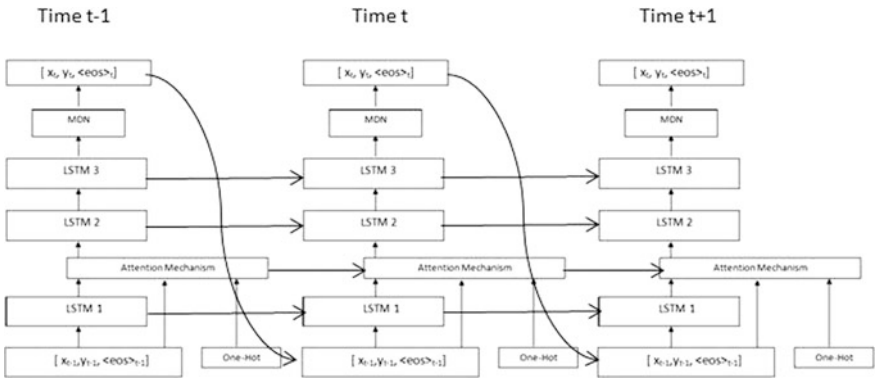


Fig. 4 Recurrent model to feedforward information from past iterations

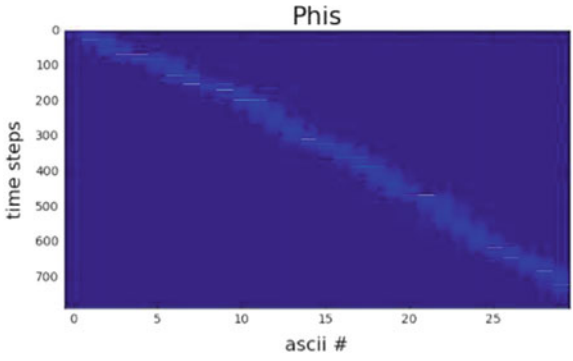
Our main motive is to have a network that predicts an entire distribution. In this paper, we are predicting a mixture of Gaussian distributions by estimating their means and covariance with the output from a dense neural network.

This will result in the network being able to estimate its own uncertainty. When the target is noisy, it will predict diffuse distributions, and where the target is really likely, it will predict a peaky distribution.

3.4 Load Saved Data

After we complete building our model, we can start a session and load weights from a saved model. We can start generating handwriting after loading and training the weights (Figs. 5 and 6).

Fig. 5 Phis plot



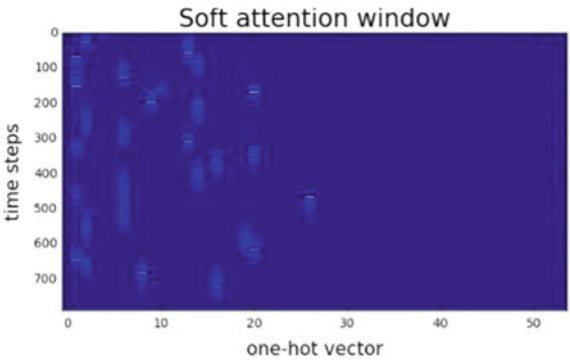


Fig. 6 Soft attention Window plot

3.5 Plots

After we complete building our model, we can start a session and load weights from a saved model. We can start generating handwriting after loading and training the weights.

Phis: It is a time series plot of the window's position with vertical as time and horizontal axis as sequence of ASCII characters that are being drawn by the model. We complete building our model, and we can start a session and load weights from a saved model. We can start generating handwriting after loading and training the weights.

Soft Attention Window: This is a time series plot of one-hot encodings produced by the attention mechanism with horizontal axis as one-hot vector and vertical axis as time.

Step 2: On building the model given above, we can start a session and load weights from a saved model. We can start generating handwriting after loading and training the weights. We write script train.py to train our model. Running the script will launch training with default settings (we can use argparse options to experiment). A summary directory with separate experiment directory is created by default for each run. We need to provide a path to the experiment we would like to continue if we wish to restore training.

$$python\, train.py{-}restore = summary\backslash experiment - 0$$

Losses can be visualized in command line or by using tensor board.

Using tensorflow 1.2 and GTX 1080 on default settings, the training time was approximately five hours.

Step 3: Generate handwriting using the model:

A script generate.py is used to test the working of the model after training of the model (Fig. 7).

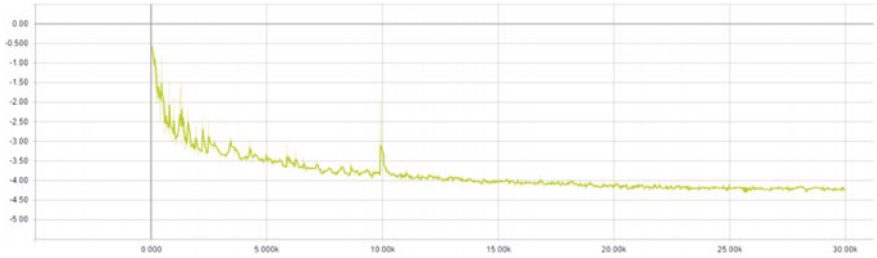


Fig. 7 Loss plot

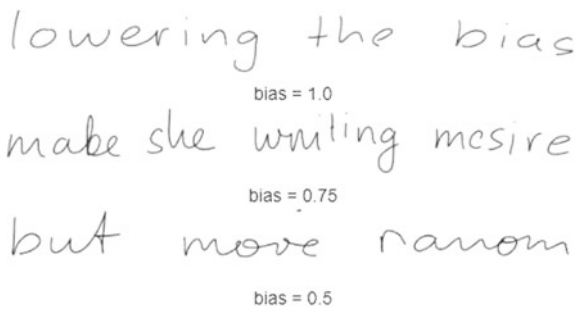


Fig. 8 Results for Bias = 0.5, 0.75, 1.0

$$\text{python generate.py} -\text{model} = \text{path\_to\_model}$$

The text argument is used to input the text and test the working of the model. Different options for text generation:

- bias: We introduce a bias term which redefines parameters according to the bias term, and higher the bias, the clearer the handwriting is (Fig. 8).
- noinfo: Generated handwriting without attention window is plotted.
- animation: To animate the writing
- style: Specifies handwriting style.

4 Results and Discussions

In the final step of plotting, we will be able to generate some handwriting. Since the LSTMs' states start out as zeros, the model generally chooses a random style and then maintains that style for the rest of the sample. On repeatedly sampling for a couple of times, we will be able to see the change from everything messy, scrawling cursive to a neat print.

After a few hours, the model was able to generate readable letters and continuing to train for a day, and the model was able to write sentences with a few trivial and

Fig. 9 Results for bias 1, output of (1)

minor errors. After the model has generated few readable letters, as compared to most of the training sequences (as generated above) that were 256 points long in this model, we were able to sample sequences up to 750 points long. The handwriting can look cleaner or messier. This can be done by drawing the coordinates (x, y) from the Gaussian distribution network that has been predicted by the MDN cap of the model described. We introduce a bias term which redefines parameters according to the bias term.

Run command *Python generate.py –noinfo –text="this was generated by computer" –bias=1* (1), on running *python generate.py –noinfo –animation –text="example of animation" –bias=1* we can specify the animation style (Fig. 9).

5 Conclusion

Deep learning is a thriving field that is seeing lots of different advancements. With help of recurrent neural networks, Long Short-Term Memory networks, convolution networks and backpropagation to name a few, this field is quickly turning explosive and successful. The model used in this paper is a proof of the strength of deep learning in general. A text to speech generator can be generated by using the same model used in the paper; this can be achieved by using appropriate dataset and increasing the parameter number. The possibilities of deep learning are limitless.

References

1. Getting Started-DeepLearning 0.1 documentation. Deeplearning.net, 2017 [Online]. Available http://www.deeplearning.net/tutorial/gettingstarted.html. Accessed 19 Feb 2018.
2. Bridle, J.S. 1990. Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition. In *Neurocomputing: Algorithms, Architectures and Applications*, ed. F. Fogleman-Soulie, and J. Herault, 227–236. Springer.
3. Perwej, Y., and A. Chaturvedi. 2011. Machine Recognition of Hand Written Characters Using Neural Networks. *International Journal of Computer Applications* 14 (2), 6–9.
4. LeCun, Y., L. Jackel, P. Simard, L. Bottou, C. Cortes, V. Vapnik, J.S. Denker, H. Drucker, I. Guyon, U. Muller and E. Sackinger. 2018. Learning Algorithms for Classification: A Comparison on Handwritten Digit Recognition. *Neural Networks: The Statistical Mechanics Perspective* 261 (1), 276–309.

5. Plamondon, R., and S. Srihari. 2000. On-Line and Off-Line Handwriting Recognition: A Comprehensive Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22 (1): 63–78.
6. http://www.fki.inf.unibe.ch/databases/iam-handwriting-database.
7. Leverington, D. Neural Network Basics, Webpages.ttu.edu, 2009. [Online]. Available http://www.webpages.ttu.edu/dleverin/neural_network/neural_networks.html. Accessed 19 Feb 2018.