DeepScript: Handwriting Generation with GANs Architecture

Christopher Chute
Department of Computer Science
Stanford University
chute@stanford.edu

Maximilian Lam
Department of Computer Science
Stanford University
maxlam@stanford.edu

Justin Myles
Department of Physics
Stanford University
jmyles@stanford.edu

Mansheej Paul
Department of Applied Physics
Stanford University
mansheej@stanford.edu

Abstract

This paper shows how to improve results in handwriting generation using Professor Forcing, a recent framework similar to generative adversarial networks (GANs). In particular, we evaluate whether we can increase the robustness of a long short-term memory (LSTM) handwriting generator by adding a discriminator network that distinguishes between the generator's hidden states in two modes: "free-running" mode, where the generator's output $\hat{\boldsymbol{y}}_t$ is fed as input for the next timestep $t+1$; and "teacher forcing" mode, where instead the generator receives the ground-truth label $\boldsymbol{y}_t$ as input. We find that the discriminator achieves the desired effect, forcing the evolution of hidden states of the generator in free-running mode to more closely mirror those same dynamics in teacher forcing mode. After training the generator with Professor Forcing, we find that the generator can produce longer sequences of realistic handwriting in free-running mode.

1 Introduction

We investigate the problem of generating realistic handwriting given raw ASCII text. Specifically, given a sequence of ASCII characters $\mathbf{c} = c_1, ..., c_n$, we aim to generate a sequence of 2D pen tip locations $\mathbf{x} = x_1, ..., x_m$ whose trace looks like a handwritten transcription of the input text. Each x_i also includes a binary end-of-stroke indicator, where `<eos>` $= 1$ corresponds to lifting the pen. This problem has many interesting applications including automatically generating handwritten notes, forgery, and anti-forgery of handwritten text, and "style transfer" between handwritten text. It is also an intriguing step in the development of AI to endow computers with handwriting, a characteristic that we often consider a representation of human personality.

2 Related work

Long short-term memory (LSTM) units were first used to generate handwriting by Graves [1][2]. This network builds a probabilistic model of handwriting from which it samples a sequence of pen positions. It remains the state of the art solution for generating long sequences of handwritten text from ASCII. In other work, Lake et. al. generate handwritten characters from the world's alphabets in an attempt to learn human interpretable concepts like pen strokes [3]. Google Magenta uses SketchRNN, a network for generating sketches to write handwritten characters [4]. However, these approaches are geared towards character writing and learning and interpreting representations, not the practical task of translating of ASCII sequences to handwriting.

One of the main challenges in building generative models for sequences is that results decrease in quality as the sequence length increases. This is especially problematic if we want to generate long sequences of handwritten text. A recently

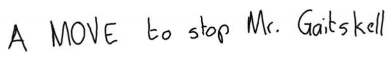


Figure 1: Example image of transcribed text from IAM dataset.

introduced technique for training recurrent neural networks (RNNs) called Professor Forcing aims to fix this issue [5]. It improves upon the standard approach to training RNNs by using a framework similar to a generative adversarial network (GAN). In Professor Forcing, a discriminator network acts as a regularizer for the generator, producing a more robust generative model for longer sequences [6]. The authors show that their method is effective for language modeling and other sequence generation tasks. In our project we improve upon Alex Graves model for handwriting generation using Professor Forcing.

3 Dataset and Features

We use the IAM Online handwriting database, which contains examples of handwriting and the corresponding annotated ASCII characters of 86,272 words in 13,049 lines from 221 individual writers [7]. The database is divided into a training set, two validation sets and a test set, containing respectively 5364, 1438, 1518, and 3859 handwritten lines. We follow Graves and combine the training set, test set and the larger of the validation sets for training and use the smaller validation set as a dev set [2]. This maximizes the size of our training set at the cost of overfitting to the dev set. This is ok as our goal is just to generate realistic looking handwriting.

The data is generated by instructing individuals to transcribe text from the 2000-word Lancaster-Oslo/Bergen (LOB) corpus on a whiteboard with an infrared imaging system [7]. The handwriting is recorded in an online format—it consists of a sequence of (x, y)-coordinates for the locations of the pen tips and a time-stamp for each tip position. Additionally for every time step there is an `<eos>` flag indicating if the pen tip is lifted or not. The raw features were pre-processed using the method of [8] including automated segmentation of words by [9]. While training, we cropped sequences to a fixed length.

The downloadable file of divided lines, including XML meta-information, is roughly 50MB of handwriting data. An example image of a line is shown in Figure 1.

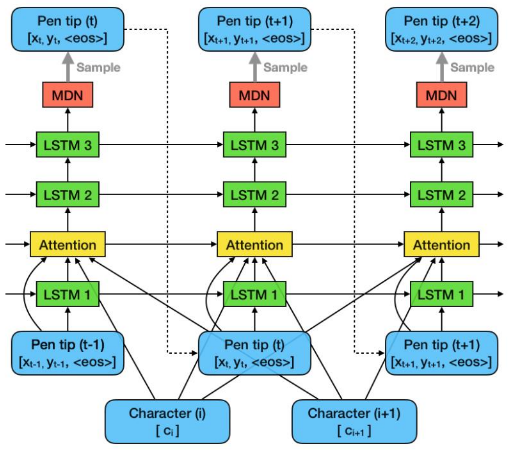
4 Methods

Our approach to handwriting synthesis is based on the approach outlined in [2]. We begin with an open source implementation of the handwriting generator network from [10]. We implement our own discriminator network, and use Professor Forcing to further train the generator to write longer and more robust sequences (see Section 4.2).

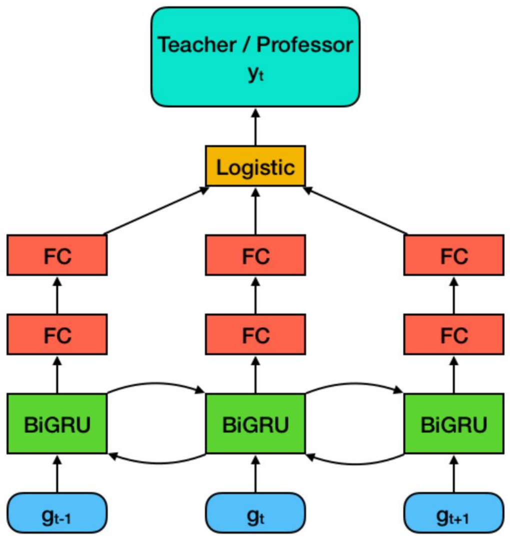
4.1 Handwriting Synthesis: Graves' Model

Graves' Handwriting synthesis network takes as its input a character sequence $\mathbf{c} = (c_0, c_1, ..., c_n)$ and models a parameterized distribution $p(\mathbf{x}|\mathbf{c})$ where $\mathbf{x} = (x_0, x_1, ..., x_m)$ is a sequence with x_i being the i-th pen tip position (represented by an (x, y) coordinate) and end of stroke (`<eos>`) flag representing whether the pen is lifted off the paper. The network is illustrated in Figure 2a.

At each time step, the first LSTM takes as its input x_t, either the previously predicted pen tip and `<eos>` flag in free generation mode, or the true data from the training set in teacher forcing mode. The output of LSTM 1 is combined with x_t and the character information from $\mathbf{c}$ using an attention window that rolls over the character sequence with a mixture of Gaussians. This is then fed into two stacked LSTMs. The output of LSTM 3 is flattened, put through a nonlinear activation and fed into a Mixture Density Network (MDN). For each time step, the MDN predicts a vector y_t, whose elements are the means, covariances and weights of a mixture of Gaussians as well as the probability of an `<eos>` flag. $y_t = \left(e_t, \left[\pi_t^j, \mu_{1,t}^j, \mu_{2,t}^j, \sigma_{11,t}^j, \sigma_{12,t}^j, \sigma_{22,t}^j\right]_{j=1}^m\right)$ where m is the number of Gaussians in the mixture, e_t is the probability of an `<eos>` flag, π_t^j is the weight of the j-th Gaussian, and the μ^j and σ^j parameterize the means and covariances of the Gaussians respectively. To obtain the next pen tip, we sample from this distribution. To train this network, for each example, we use a loss that tries to minimize the negative log likelihood of the next true pen tip given



(a) Schematics of the handwriting synthesis model in [2]. The bold arrows indicate connections with trainable weights. The dotted arrows indicate that the sampled output at time t is used as input at time $t+1$. MDN stands for mixture density network.



(b) Discriminator network used in Professor Forcing.

Figure 2

our predicted y_t and sum that over the time steps, where θ_{gen} represents the parameters of the generator.

$$NLL(\theta_{gen}) = E_{(\mathbf{c},\mathbf{x})\sim data}\left[-\log P_{\theta_{gen}}(\mathbf{x}|\mathbf{c})\right] \tag{1}$$

$$= \sum_{t=1}^{T} -\log(\sum_{j} \pi_t^j \mathcal{N}(x_{t+1}|\mu_t^j, \sigma_t^j)) - (x_{t+1})_3 \log e_t - (1 - x_{t+1})_3 \log(1 - e_t) \tag{2}$$

4.2 Professor Forcing

Traditional RNNs can quickly get off track during sequence generation. This is because they take the output of previous time steps as input for subsequent time steps, which allows errors to accumulate. To address this problem, RNNs are typically trained with the Teacher Forcing algorithm, in which the input at timestep $t+1$ is the ground-truth label x_t, rather than the output $\hat{x}_t$. While this improves sequence generation, ground-truth labels are only available during training. Thus Teacher Forcing introduces a discrepancy between training inference and sampling inference.

Professor Forcing, a recently introduced technique for training RNNs, rectifies this by incorporating the idea of a generative adversarial network (GAN) to generative sequential models. A discriminator network aims to distinguish between the generator's output and hidden state in two different modes: while sampling in "teacher forcing" mode and sampling in "free-running" mode [5]. The generator is then trained to fool the discriminator into predicting that it is in the "teacher forcing" mode while it is actually sampling in "free-running" mode. This forces the hidden state and outputs of the generator in the two different modes to be similar. As a result, even when the generator is freely sampling, its dynamics mirrors that of when it is "pinned-down" by the ground truth labels. Professor Forcing also acts as a regularizer, reducing risk of over-fitting the data and resulting in a generator that produces examples more representative of the distribution from which the training sample is selected. Figure 3 shows the Professor Forcing setup.

The discriminator we use is shown in Figure 2b. It consists of a layer of bi-directional GRUs that take as input a sequence of hidden states of the generator [11]. The output of the GRUs is then fed into a two layer perceptron whose weights are shared across time. The output activations of these perceptrons are then combined across time and a logistic unit is used to predict the probability that the hidden state comes from the teacher forcing mode. During training of the discriminator, we aim to minimize the cost function C_{disc} [5]:

$$C_{disc}(\theta_{disc}|\theta_{gen}) = E_{(\mathbf{c},\mathbf{x})\sim data}\left[-\log D(g(\mathbf{c},\mathbf{x},\theta_{gen}),\theta_{disc})\right] \tag{3}$$

$$+ E_{\mathbf{x}\sim P_{\theta_{gen}}(\mathbf{x}|\mathbf{c})}\left[-\log(1 - D(g(\mathbf{c},\mathbf{x},\theta_{gen}),\theta_{disc})\right], \tag{4}$$

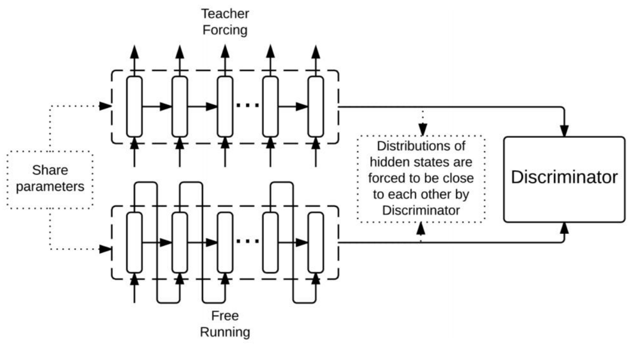

Teacher Forcing
Share parameters
Distributions of hidden states are forced to be close to each other by Discriminator
Discriminator
Free Running

Figure 3: The setup for training RNNs as generative models using Professor Forcing [5]. Image taken from paper.

Hyperparameter	Value
Generator time steps	150
Attention Gaussian Mixtures	8
MDN Gaussian Mixtures	8
LSTM state size	100
Discriminator Bi-GRU state size	100
Discriminator FC Layer Output Sizes (1 & 2)	100 & 10
Optimizer	Adam ($lr = 10^{-4}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$)
Dropout while training	0.85
Batch size	32

Table 1: Hyperparameters for the generator.

where θ_{disc} and θ_{gen} are the parameters of the discriminator and generator respectively. $\mathbf{c}$ is the character sequence and $\mathbf{x}$ is the sequence of pen-tip positions. $g(\mathbf{c}, \mathbf{x}, \theta_{gen})$ represents the hidden state and output of the generator. In our implementation, instead of using the full hidden state activation, we just use the activations of the mixture density network as this proved to be sufficient for the discriminator and computationally efficient. Finally, $D(g)$ is the output of the discriminator—the probability that g is in teacher forcing mode. The above cost reduces to logistic loss of whether the discriminator distinguishes between teacher forcing mode and free running mode. During professor forcing training of the generator, we have an additional cost for the generator C_{free},

$$C_{free}(\theta_{gen}|\theta_{disc}) = E_{\mathbf{c}\sim data, \mathbf{x}\sim P_{\theta_{gen}}(\mathbf{x}|\mathbf{c})}\left[-\log D(g(\mathbf{c}, \mathbf{x}, \theta_{gen}), \theta_{disc})\right] \tag{5}$$

Minimizing this cost corresponds to fooling the discriminator into predicting that the free running generator is sampling in teacher forcing mode. The generator is trained to simultaneously minimize $C_{free}(\theta_{gen}|\theta_{disc}) + NLL(\theta_{gen})$. An important practical consideration is that if the discriminator is too accurate or inaccurate, it doesn't help training the generator. So we set the training schedule so as to keep the discriminator above 75% and below 95% accuracy.

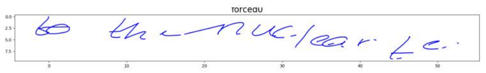
5 Experiments

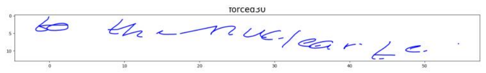
5.1 Protocol

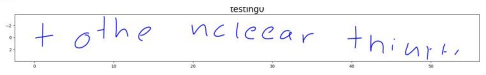
We pretrain our handwriting generation model following Graves' specifications [2]. Table 1 shows the hyperparameters of the model. After pretraining our handwriting generation model, we take the weights of the pretrained model and perform Professor Forcing as described in Section 4.2. Furthermore, to propagate the gradient of the discriminator back

Hyperparameter	Value
LSTM state size	100
First fully connected dimension	100
Second fully connected dimension	10

Table 2: Hyperparameters for the discriminator.


(a) Teacher forcing outputs at iteration 0 for the text "to the nuclear."


(b) Teacher forcing outputs at iteration 30 for the text "to the nuclear."


(c) Free running outputs at iteration 0 for the text "to the nuclear."

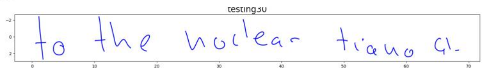
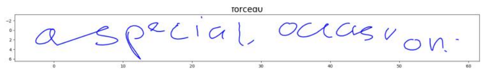
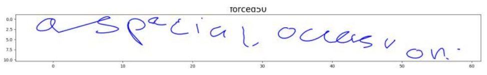
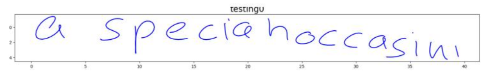

(d) Free running outputs at iteration 30 for the text "to the nuclear."

Figure 4


(a) Teacher forcing outputs at iteration 0 for the text "a special occasion."


(b) Teacher forcing outputs at iteration 50 for the text "a special occasion."


(c) Free running outputs at iteration 0 for the text "a special occasion."

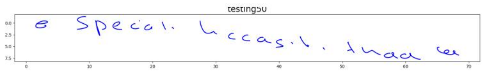

(d) Free running outputs at iteration 50 for the text "a special occasion."

Figure 5: Teacher forcing and free-running outputs at various stages in training for the input text, "a special occasion."

into the generators we use truncated backpropagation through time, with $T = 1$. Generator hyperparameters remain the same (as shown in Table 1); Table 2 shows the hyperparameters for training the discriminator. Due to limited time, we do professor forcing on a single example and plot generated handwriting samples for both teacher forced and free running generators after each iteration.

5.2 Results

Figure 4a and figure 4b show the outputs of the generator using teacher forcing mode at iterations 0 and 30 for the phrase "to the nuclear"; Figure 4c and figure 4d show the outputs of the generator in free running mode at iterations 0 and 30 for the phrase "to the nuclear".

Figure 5a and figure 5b show the outputs of the generator using teacher forcing mode at iterations 0 and 50 for the phrase "a special occasion; Figure 5c and figure 5d show the outputs of the generator in free running mode at iterations 0 and 50 for the phrase "a special occasion".

5.3 Discussion

Figures 4a, 4b, 4c, 4d and figures 5a, 5b, 5c, 5d show that professor forcing causes the outputs of free running and teacher forced modes to become gradually become closer to each other. Furthermore, with professor forcing free running mode is better able to output longer legible sentences. If we plot the discriminator loss, we see that the discriminator is quickly able to tell the difference between the outputs of the different modes of the generator.

While the results are not optimal, results show that professor forcing has a notable effect on generated outputs; specifically, results from the two generators seem to converge towards each others' styles. One interesting thing we found was that after a bit of training the generators tended to diverge and output junk; we believe that jointly training with the traditional loss function on handwriting data would improve quality.

6 Conclusion/Future Work

For future work, we would like to improve the Professor Forcing training technique to by emphasizing the reproduction of the style of the output. We propose doing this by adding a regularization parameter to the generator cost: $C_{gen} = NLL + \lambda C_{free}$. By decreasing the value of lambda, we can force our training output to more faithfully reproduce the styles in the handwriting. In general, λ will provide more control over the tradeoff between fooling the generator and reducing the original log likelihood cost of predicting the data, giving us more control over the generative model.

7 Contributions

All authors maintained a high level of collaboration and contributed equally. In the beginning, M.L. and C.C. focused on the generator and M.P. and J.M. focused on the discriminator. In later stages, all authors were closely involved with getting backpropagation hooked up between the discriminator and the generator, debugging, training, and producing deliverables for the project.

References

[1] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[2] A. Graves. Generating Sequences With Recurrent Neural Networks. *ArXiv e-prints*, August 2013.

[3] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[4] David Ha and Douglas Eck. A neural representation of sketch drawings. In *International Conference on Learning Representations*, 2018.

[5] A. Lamb, A. Goyal, Y. Zhang, S. Zhang, A. Courville, and Y. Bengio. Professor Forcing: A New Algorithm for Training Recurrent Networks. *ArXiv e-prints*, October 2016.

[6] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Networks. *ArXiv e-prints*, June 2014.

[7] M. Liwicki and H. Bunke. Iam-ondb - an on-line english sentence database acquired from handwritten text on a whiteboard. In *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*, pages 956–961 Vol. 2, Aug 2005.

[8] U. v. Marti and H. Bunke. Handwritten sentence recognition. In *In Proc. Int. Conf. on Pattern Recognition*, pages 467–470, 2000.

[9] M. Zimmermann and H. Bunke. Automatic segmentation of the iam off-line database for handwritten english text. In *Object recognition supported by user interaction for service robots*, volume 4, pages 35–39 vol.4, 2002.

[10] Sam Greydanus. Scribe: realistic handwriting with tensorflow, 2016.

[11] Rahul Dey and Fathi M. Salem. Gate-variants of gated recurrent unit (GRU) neural networks. *CoRR*, abs/1701.05923, 2017.