Generating Quick Drawings with LSTM Recurrent Neural Networks

Michelle McGhee and James Ortiz
Department of Computer Science
Stanford University
mmcghee@stanford.edu, jameso2@stanford.edu

Abstract

To better understand how the tasks of handwriting generation and drawing generation are related, we adapted an LSTM recurrent neural network used for handwriting generation to generate drawings of cars. Specifically, the model takes in the object class as well as the beginning of a car drawing and attempts to predict the remainder of the drawing given these inputs. Overall, we found that the model had difficulty making accurate predictions; however, the network seemed to have learned some information about the shape of a car from the first 50 points fed as input.

1 Introduction

Both handwriting and drawing are complex tasks that can be modeled as sequence generation. In this project, we explored whether the connection between generating handwriting and generating drawings allows us to adapt a handwriting generation model for drawing generation instead. The input to our model is an object class (ex: apple, bat, car) and the first 50 points of a drawing of that object. The output of our model is a sequence of points that completes the drawing. The end goal is to create a tool that can draw with a user: if the user starts to draw an object, then the model can complete the drawing.

2 Related work

Our model architecture is adapted from an implementation of Alex Graves' paper on handwriting generation [1, 2]. Graves trains a deep recurrent neural network to take in a string of text (ex: "Hello") and output the sequence of points in the handwritten version of the string. We were inspired to adapt this neural network to drawing generation by Google's "Quick, Draw!" game [3], which uses a neural network that can classify images drawn by a user. The drawings that Google gathered from this game have been compiled into a large dataset [4], and many interesting projects have arisen from this dataset. For example, Magenta created a tool called "Draw Together," [5] in which the user starts drawing an image and the tool completes the drawing. Inspired by this project, we decided to build our own version of "Draw Together" using a modified version of the neural network architecture implemented in Graves' paper.

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

3 Dataset and Features

3.1 Quick Draw dataset

The data in Google's "Quick, Draw!" dataset is stored as a set of x, y, t points representing the various strokes of the drawing. x, y represent the x-, y- coordinates of a point, and t is an indicator variable indicating whether this point is the last point in the stroke. For the purposes of this project, we focused on a single object class: cars. We chose cars because drawing a car is not trivially simple, but this task is also not so difficult that everyone draws a car in vastly different ways. We took 100,000 car drawings from the Quick Draw dataset and placed 100 of them into the test set, leaving the remainder for the training set. After eliminating the drawings that contained fewer than 50 points, we ended up with 99,740 drawings in our training set and 100 drawings in our test set.

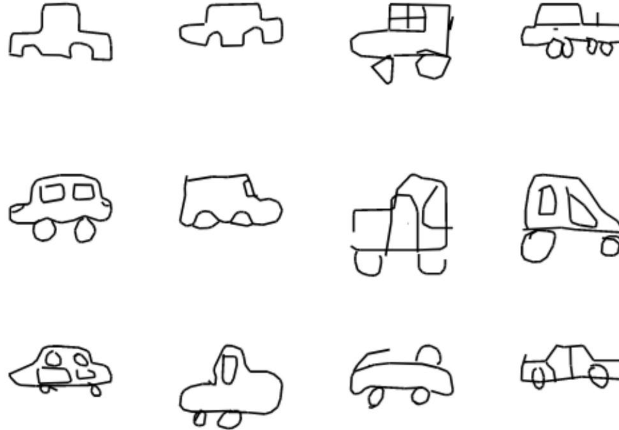

Figure 1: Car drawings from Google's "Quick, Draw!" dataset.

3.2 Encoding mechanism

The input to the model was an object class and the first 50 points of a drawing. We used a GloVe model trained on a Wikipedia corpus of 6 billion tokens to compute a 50-dimensional word embedding for the object class. Then, we extracted the first 50 points from each of the drawings and flattened these arrays into vectors. Finally, we concatenated the word embedding to each of the flattened vectors and fed the resulting vectors as inputs to the model.

4 Methods

We used a modified version of the deep recurrent neural network developed by Alex Graves for handwriting generation. The original architecture implemented by Graves is shown in the following figure.

As seen in the figure above, the neural network consists of a window layer, two hidden layers, and a mixture density output layer. The window layer was used in handwriting generation as an attention mechanism that determined which parts of the input string were most relevant to the handwritten points being generated. Since this mechanism was not relevant for our task, we removed the window layer from the neural network.

The two hidden layers consist of Long Short Term Memory (LSTM) cells and are governed by the following equations:

$$h_t^1 = H(W_{h^1x}x_t + W_{h^1h^1}h_{t-1}^1 + b_h^1)$$
$$h_t^2 = H(W_{h^2x}x_t + W_{h^2h^1}h_t^1 + W_{h^2h^2}h_{t-1}^2 + b_h^2),$$

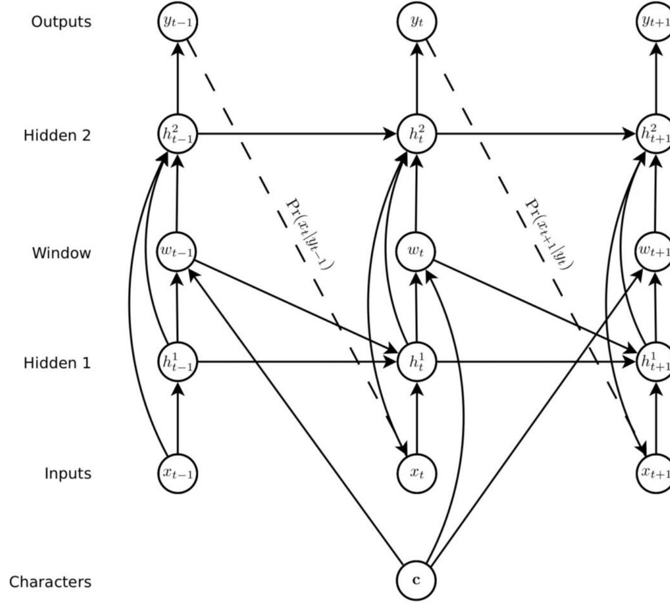

Figure 2: Architecture of deep recurrent neural network used for handwriting generation. Graphic adapted from Graves, 2014.

where h_t^n is the activation at time step t in hidden layer n, H is the hidden layer function (in this case, the equations governing an LSTM cell), $W_{h^n x}$ is the weight matrix connecting the input to the nth hidden layer, $W_{h^n h^m}$ is the weight matrix connecting the activation in the mth hidden layer to the activation in the nth hidden layer, and b_h^n is the bias vector in the nth hidden layer. Rather than initialize the first activation of the first hidden layer, h_0^1, as a vector of zeros, we initialized it as a vector that encodes the first 50 points of the drawing and the object class (refer to section 3.2 for more details on the encoding mechanism).

The mixture density output layer takes in as input h_t^1 and h_t^2, the activations from the first and second hidden layers, and outputs $\{\pi_t^j, \mu_t^j, \sigma_t^j, \rho_t^j\}_{j=1}^M$, the parameters for a mixture of M bivariate Gaussians, and e_t, the parameter for a Bernoulli distribution.

$$\hat{y}_t = b_y + \sum_{n=1}^{2} W_{yh^n} h_t^n = \left(\hat{e}_t, \{\hat{\pi}_t^j, \hat{\mu}_t^j, \hat{\sigma}_t^j, \hat{\rho}_t^j\}_{j=1}^M\right)$$
$$e_t = \frac{1}{1+\exp(\hat{e}_t)} \implies e_t \in (0,1)$$
$$\pi_t^j = \frac{\exp(\hat{\pi}_t^j)}{\sum_{j'=1}^M \exp(\hat{\pi}_t^{j'})} \implies \pi_t^j \in (0,1), \sum_j \pi_t^j = 1$$
$$\mu_t^j = \hat{\mu}_t^j \implies \mu_t^j \in \mathbb{R}$$
$$\sigma_t^j = \exp(\hat{\sigma}_t^j) \implies \sigma_t^j > 0$$
$$\rho_t^j = \tanh(\hat{\rho}_t^j) \implies \rho_t^j \in (-1,1)$$

The bivariate Gaussians determine $P(x_{t+1}|y_t)$, the probability distribution of the next point in the drawing x_{t+1} given the outputs y_t of the mixture layer:

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

and the Bernoulli distribution, $Ber(e_t)$, determines whether or not x_{t+1} is the end of a stroke.

To train the network, we used the Adam optimization algorithm, an extension of the traditional gradient descent algorithm that incorporates an exponential moving average of the gradients and the squared gradients to update the parameters. We decay the learning rate exponentially and use negative log likelihood as our loss function. More specifically, given a training example $\mathbf{x}$, which in this case is a sequence of points in a drawing, our goal is to minimize the following loss function:

$$\mathcal{L}(\mathbf{x}) = -\sum_{t=1}^{T} \log P(x_{t+1}|y_t) =$$

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

By minimizing this loss function, we are essentially choosing parameters for our neural network that maximize the likelihood of $\mathbf{x}$, the sequence of points in the drawing.

5 Experiments/Results/Discussion

We left most of the hyperparameters set in Grzegorz Opoka's implementation of Alex Grave's neural network unchanged. For example, we used the same learning decay rate of 0.5 and the same number of epochs of 30. However, we changed the batch size from 64 to 1 in order to speed up training, as we found that training initially ran very slowly.

The following figure shows an example of the neural network's attempt to complete a car drawing given the first 50 points of the drawing.

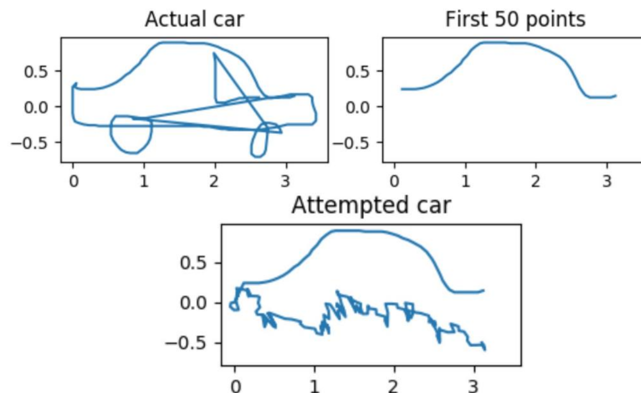

Figure 3: An example drawing of a car from our test data set (top left), the first 50 points of the drawing (top right), and the drawing predicted by our trained neural network (bottom).

The network appears to learn some information about the shape of the car: in this example, it seems to recognize that it received the top of the car as input and must draw the bottom of the car in order to complete the drawing. However, the network is unable to accurately complete the drawing. We attribute the network's ability to extract some spatial information from the first 50 points to our encoding mechanism; however, because our mechanism was simplistic, the network may not have learned sufficient information to complete the drawing.

As a measure of our model's performance, we computed the mean squared error between the sequences of points predicted by the neural network, $\hat{\mathbf{y}}^{(i)}$, and the sequences of actual points, $\mathbf{y}^{(i)}$, across the 100 test images using the following formula:

$$\text{MSE} = \frac{1}{100}\sum_{i=1}^{100}(\hat{\mathbf{y}}^{(i)} - \mathbf{y}^{(i)})^2$$

The mean squared error for our model came out to **17.68**. This result indicates that the neural network had difficulty completing the car drawings accurately. There are several reasons as to why this result may have occurred. One is that car drawings involve a fair amount of complexity and variability. Since the outline of a car is a non-standard shape, people tend to draw cars differently. As seen in Figure 1, there is a high degree of variability in the car drawings. Moreover, since the drawings were made in 20 seconds or less, the data is relatively noisy. The relatively high degree of variability and noise in the data may have made it difficult for the network to learn the overall shape of a car.

Another potential reason for the quality of our results was our choice of training hyperparameters, in particular the batch size. Since we used a batch size of 1, our gradient descent algorithm was

stochastic; therefore, the gradient update steps were efficient but not high-quality. This choice of batch size caused training to run efficiently but reduced the quality of our model's performance.

6 Conclusion/Future Work

In this work, we trained a deep recurrent neural network to predict the points in a car drawing given the object class and the first fifty points of the drawing. Overall, we found that the model had difficulty making accurate predictions; however, the network seemed to have learned some information about the shape of a car from the first 50 points because of our encoding mechanism.

Given more time and resources, we would have implemented a more sophisticated encoding mechanism to allow the network to learn more information about the first 50 points. Specifically, we would have used a convolutional neural network to extract more spatial information about the first 50 points.

We would have also trained our network on less difficult object classes, such as doors, books, etc. If the neural network successfully learned to complete drawings for these less complex classes, then we would have expanded the number of object classes to make the program more versatile.

Finally, we would have more carefully tuned the training hyperparameters, particularly the batch size and the number of epochs, to improve the quality of our model's performance.

7 Contributions

We worked together and in person for almost every step of the process. At one point, James took the lead on analyzing the architecture described in the Alex Graves' paper while Michelle took the lead on compiling and formatting the data from the "Quick, Draw!" dataset. Aside from that, we did the programming and project writeups together.

8 Appendix

The code we wrote for this project, adapted from an implementation of the handwriting generation model [2], can be found here: https://github.com/mmcghee18/handwriting-generation

References

[1] Graves, Alex. (2013). Generating sequences with recurrent neural networks. arXiv preprint arXiv:1308.0850.

[2] Grzegorz Opoka, Handwriting Generation, (2017), GitHub repository, https://github.com/Grzego/handwriting-generation

[3] Google Creative Lab (2016). Can a neural network learn to recognize doodling? Retrieved from https://quickdraw.withgoogle.com

[4] Google Creative Lab, The Quick, Draw! Dataset, (2017), GitHub repository, https://github.com/googlecreativelab/quickdraw-dataset

[5] Ha, David; Jongejan, Jonas; Johnson, Ian (2017, June 26). Draw Together with a Neural Network. Retrieved from https://magenta.tensorflow.org/sketch-rnn-demo