Handwriting Synthesis

Animesh Bohara (160050040) [1],
Eashan Gupta (160050045) [1] **, Ankit (160050046)** [1]

[1]Department of Computer Science and Engineering,
Indian Institute of Science Bombay, Mumbai, India

1. Introduction

We try to tackle the problem of Online Handwriting Generation. Online Handwriting Generation deals with *Online data* input in the form of a sequence of the x and y-coordinates of the pen locations, compared to the pixel values in Offline data. We have tried to gain some ideas from the already implemented paper by Alex Graves and add some of our own to do the computation more efficiently or using a simpler model. This is also following some of the ideas mentioned in the future research topics in the same paper

2. Background

Link to the paper:

- *Generating Sequences With Recurrent Neural Networks by Alex Graves*

According to the given paper we can generate a handwriting for a given sentence by using an RNN and modelling attention. The loss for the same is modeled in the form of mixture of bi-variate Gaussian distributions and the attention as a mixture of Gaussian distributions based on the index number of the character of the line. The model is given data from the IAM OnDb dataset(link) which is of the form of sentences and the offset based strokes.

3. Other Related Works

3.1. Handwriting synthesis using CycleGAN

Papers:

(1) *Generating Handwritten Chinese Characters using CycleGAN*

An approach for text to image handwriting synthesis is inspired from (1). We can use standard handwritten input images for each of the letters to feed the CycleGan and generate new images for each character. Then we use these to generate the complete set of handwriting. Some pros and cons of this approach are:

1. **Cons:**
   - These images may be generated but it may be difficult to maintain the same style of handwriting for all the characters.
   - It may be particularly difficult to generate images of complete words in running hand. This is a very major problem as compared to other approaches as this way deals with images directly
2. **Pros:**
   - The complete set of characters may be generated while not using the entire set during training the model. That is to say that the model will be able to learn the style of handwriting using a partial set of characters and generate a complete set later on.

3.2. Using Generative Adversarial Networks

Papers:

(1) *Handwriting Profiling Using Generative Adversarial Networks*

As GANs are most suited for any type of machine generation, we thought they would be useful for handwriting generation as well. The above paper uses a GAN to generate letters (images instead of a sequence of points), and then uses a Reinforcement Learning based approach to join the letters. We ultimately discarded the idea because the Reinforcement Learning part was not clearly described well in the paper, and we could not come up with a reasonable reward function that could work for joining the letters.

We also thought of incorporating reinforcement learning in our model to join the letters but in the end we decided against it for using RNNs.

3.3. SeqGAN

Papers:

(1) *SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient*

This was a very nice paper about a new kind of GAN which is best suited for generating sequences. The problem with using a GAN for sequence generation is that we need to evaluate the fitness of a partial sequence as well. In this paper, that is done through a Monte Carlo search. But it could not be applied to our case as our input space is continuous, while the paper assumes a discrete input space, and thus Monte Carlo search is difficult in our case .

4. Goals

The main goal of this project was to use a simpler model to generate handwriting from an input of a sentence. We aimed to directly take as input the stroke-offset data of various characters and aim on only learning how to join them using the stroke-offset data of the complete corresponding sentence. So, we divided the job for each sentence into 2 parts:

1. Separate all the letters of a word and use an RNN to generate the stroke-offset for all of these and string them together. Note that the joining of these letters may not be a smooth curve.
2. Use another RNN to train over these words to make the joining process smooth. Pass the above calculated stringed character data and create a new stroke-offset data for the word.

These steps may be more in number but each of them is much easier than processing the entire sentence in one go.

Future possibility:

Later on, we planned to use a game theoretic approach to train both RNNs, for generating character and the smooth word, simultaneously like that done in GANs but were unable to do so due to time and other constraints.

5. Our Approach

Experiments

- **Language Used:** Keras, Tensorflow, Python3
- ***Link For Code***
- **Platform to run code:** Google Colab

5.1. Task 1: Sequence of Characters to Word

Stroke-offset sequence of various characters to stroke-offset sequence of the corresponding word.

Used the code from the following sources:

(1) `https://github.com/hardmaru/write-rnn-tensorflow`
(2) `https://github.com/sjvasquez/handwriting-synthesis/`
(3) `https://github.com/datalogue/keras-attention`

5.1.1. Experiment 1

For this experiment, we used the simple attention based model to take as input the stroke-offset data for characters and the words to train the model.
We used the model for the attention based RNN from (3) to get the layer and added an LSTM encoder before this. The loss function was used by us taking inspiration from (1) and the original paper. This gives the parameters for the mixture of the bi-variate Gaussian distributions of the offsets. These parameters were then used to produce samples but this approach failed as all values remained very close to zero and did not produce good results.
The utility functions were used from (2) and others were created by us. The data processing to appropriate formats was done by us.

For this implementation, the number parameters required to be trained was 19052 while the normal implementation requires around 3 million weights.

5.1.2. Experiment 2

For this experiment we changed the above code to accommodate a Gaussian function to calculate the attention as used in the paper. We trimmed and reduced the parameters required for training according to our understanding such as W_{a} was removed. We used the sum of Gaussian function calculated over previous inputs to calculate attention. This approach also failed as the loss did not decrease.The loss function and other code were same for this.

For this implementation, the number parameters required to be trained was 18152.

5.1.3. Experiment 3

For this experiment, we got rid of the encoder, and correspondingly the attention layer as well, and only used a single GRU layer. In theory the previous models should outperform this model, but that is not the case. This might be due to the fact that this model contains very few number of trainable parameters, and as a result trains very fast so we can train this for a large number of epochs.

For this implementation, the number of parameters required to be trained was 663

Following are some of the sample results from our network:

Figure 1. Text: *a*

Figure 2. Text: *won*

Figure 3. Text: *brag*

Figure 4. Text: *try*

Figure 5. Text: *beg*

Figure 6. Text: *bag*

Figure 7. Text: *ball*

5.2. Task 2: For Individual Characters

In this task we aim to generate the stroke-offset sequence given a certain character

5.2.1. Experiment 4

For this experiment, we used an *Embedding Layer* to convert the input one-hot encoding vector to a real-D dimensional vector. Then we used a GRU layer above it inspired by the relative success of Experiment 3 to generate the parameters of the mixture of bivariate Gaussian distribution that is predicted as the probability of the next point. But unfortunately, this Experiment didn't give any reasonable results.

6. Challenges

- Training the model : Choosing the optimization algorithm, tuning the hyperparameters was very challenging as the training time was substantial.
- The keras platform was new to us so it was quite challenging to start with it and decode it.

- We wanted to make use of the recently announced TPUs on Google Colab for faster training, but that expects a tf.keras Model but the Attention we were using was a keras layer, causing many compatibility issues, which we were not able to resolve.

7. Individual Contributions

Everyone worked together and nearly equally. All the models were discussed together and then implemented, so as such there was no clear distinct division of work.