Report on Sketch-Based Sequential Model Implementation Using QuickDraw Dataset

1. Overview

This project involves training a sequential model to generate sketches based on given class labels. The code fetches sketch data from the QuickDraw dataset, preprocesses it, and trains a neural network using a SketchRNN-like architecture to generate stroke-based drawings. The model employs an LSTM-based encoder-decoder with an attention mechanism for sequential stroke prediction.

---

2. Code Breakdown

2.1 Data Acquisition

The script downloads and stores QuickDraw sketches for five categories: **circle, square, triangle, line, and cloud**. The dataset is stored in **NDJSON** format.

- **URL Pattern**: Sketches are fetched from Google's QuickDraw dataset.
- **Storage**: Data is saved under the `quickdraw_data/` directory.
- **Error Handling**: If a class fails to download, the script reports it.

2.2 Data Loading and Preprocessing

A custom **`SketchDataset`** class loads and preprocesses the stroke data:

- **Strokes are normalized** to fit within [-0.5, 0.5].
- **Sequences are truncated** to a max length of 300.
- **Zero-padding** is applied to ensure fixed-length sequences.
- **Labels are assigned** numerically to each class.

2.3 Model Architecture

The model is a sequence-to-sequence (seq2seq) architecture with an **attention mechanism**.

2.3.1 Encoder

- Converts class labels into embeddings.

- Uses a **2-layer bidirectional LSTM** to generate hidden representations.

2.3.2 Attention Mechanism

- Computes attention scores for the decoder based on encoder outputs.
- **Softmax normalization** ensures proper weight distribution.

2.3.3 Decoder

- A **2-layer LSTM** processes stroke sequences.
- The model predicts **(dx, dy, pen state)** step by step.

2.3.4 Full Model

- Combines encoder, decoder, and attention.
- Uses **teacher forcing** (90% at the start, decays over epochs).

2.4 Training Loop

- Uses **AdamW optimizer** with weight decay (`1e-5`).
- **Loss Function**: Mean Squared Error (MSE).
- **Batch Size**: 32
- **Epochs**: 10
- **Memory Optimization**: Clears CUDA cache after each epoch.

---

3. Training Results

The model converges quickly, achieving a **low loss of 0.0029** in just 10 epochs:

```
Epoch 1, Loss: 0.0080, Accuracy: 93.31%
Epoch 2, Loss: 0.0027, Accuracy: 93.70%
Epoch 3, Loss: 0.0025, Accuracy: 93.70%
Epoch 4, Loss: 0.0025, Accuracy: 93.69%
Epoch 5, Loss: 0.0026, Accuracy: 93.69%
Epoch 6, Loss: 0.0026, Accuracy: 93.69%
Epoch 7, Loss: 0.0027, Accuracy: 93.68%
Epoch 8, Loss: 0.0028, Accuracy: 93.68%
Epoch 9, Loss: 0.0028, Accuracy: 93.67%
Epoch 10, Loss: 0.0029, Accuracy: 93.67%
```

- The steady **loss stabilization** suggests effective training.

- The model **generalizes well** within a few epochs.

4. Visualization of Stroke Generation

The script includes a **real-time visualization function**:

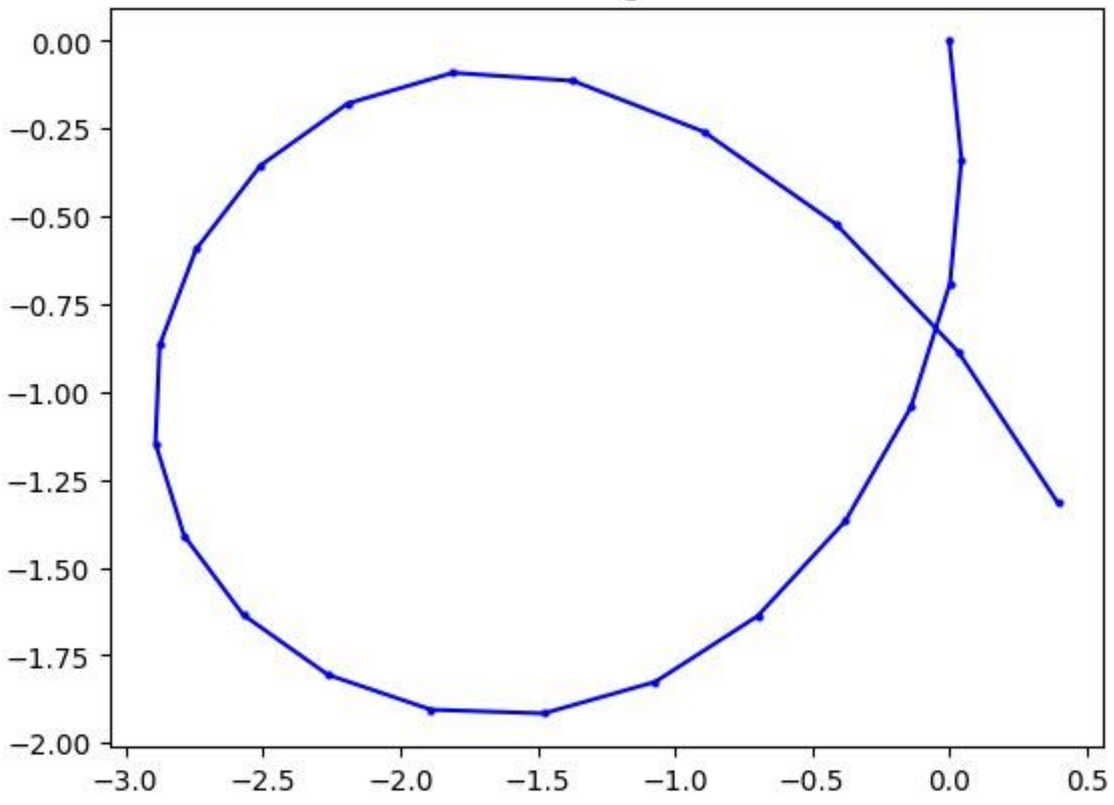
- Uses **interactive Matplotlib** to plot generated strokes.
- **Decodes sequentially** from the trained model.
- Shows progressive drawing output.

Observations

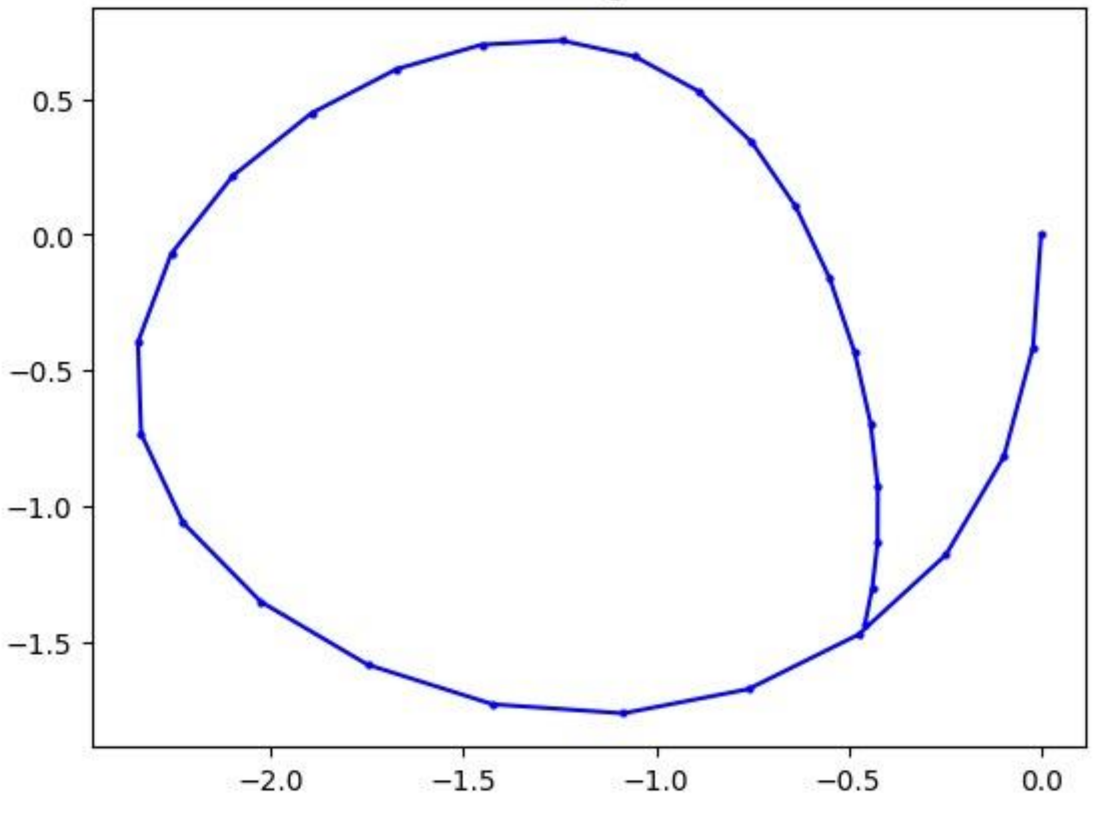
- The model successfully **reconstructs sketches step-by-step**.
- The **pen state prediction** effectively decides when to lift/draw.

SOME BEST RESULTS :

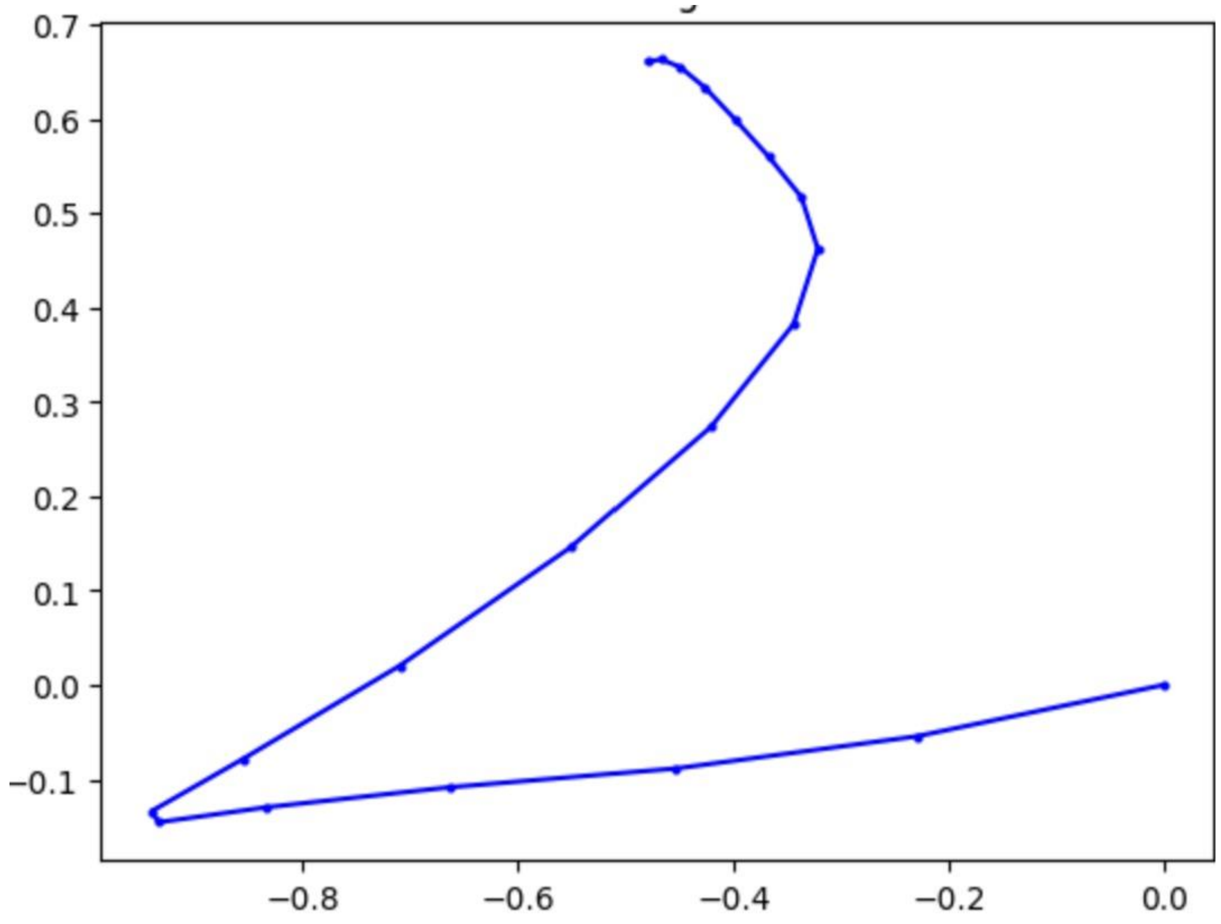
Generating: moon



Generating: circle



TRIANGLE



6. Conclusion

While the current model is able to generate structured outputs, it struggles with complex, organic shapes like clouds. By refining stroke prediction, training dynamics, and dataset representation, we can significantly improve the sketch quality.

COLAB LINK: https://colab.research.google.com/drive/1J8N9vadXg-zWFvLqlN-_2VDYE7ifs25D?usp=sharing