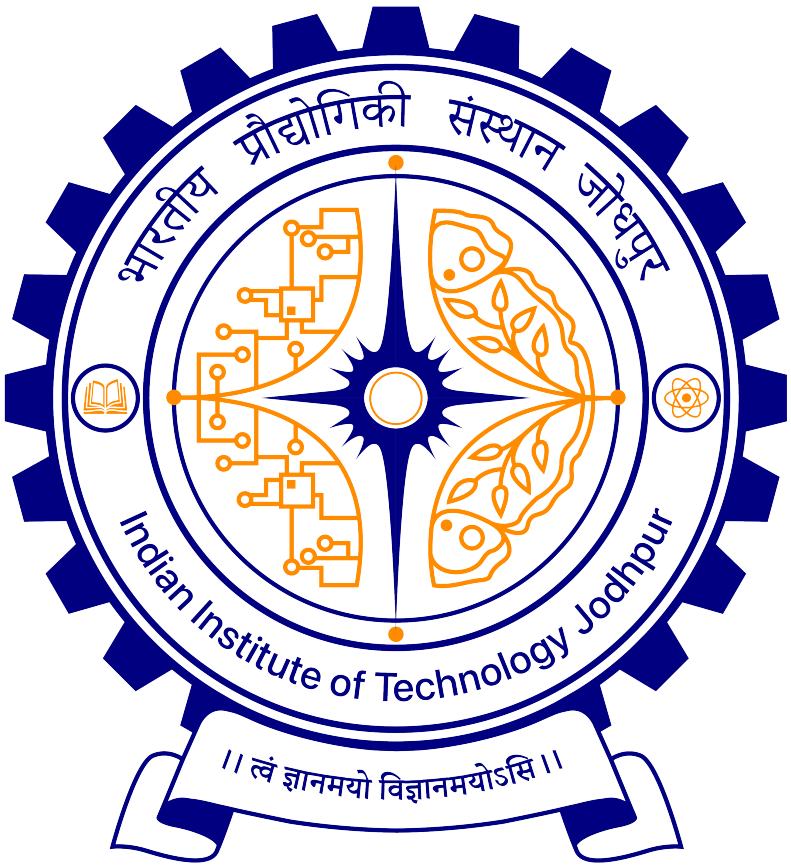

CSL7590: Deep Learning
Assignment-3

Name: Mahek Vanjani & Deepak Bhatter & Shikhar Dave
Roll No: B22EE088 & B22EE022 & B22CH032

Google Colab (Single Model Based Approach)
Google Colab (Router Based Approach)

Date: 30/03/2025

I. Objective

The objective of this assignment is to develop a sequential generative neural network that can create vector-based sketches in a step-by-step manner, replicating human drawing behavior. Given an object class name (e.g., "apple," "cat," "airplane"), the model should generate a sequence of strokes that gradually form a recognizable sketch. Instead of producing the entire image at once, the model should learn and replicate the natural progression of sketching, ensuring coherence and diversity in the generated outputs.

II. Dataset

The Quick Draw! dataset is utilized for this task. This dataset comprises millions of stroke-based drawings across hundreds of categories, collected from users playing the Quick, Draw! game. Each sketch is represented as a sequence of strokes, where each stroke consists of a series of points with associated pen states (down, up, or end of drawing). For this assignment, at least 10 distinct object classes were selected from the dataset:

- Airplane
- Alarm clock
- Apple
- Cat
- Candle

Data Format Explanation:

- Sketches are represented as a sequence of pen stroke actions, extending the format used in earlier works.
- Each sketch starts at the origin and progresses as a list of points.
- Each point in a sketch is represented by a 5-element vector: (Δx, Δy, p1, p2, p3).
- Δx and Δy specify the offset distance in x and y directions from the previous point.
- The pen's state is represented by a one-hot vector of 3 states:
  - p1: Pen touches the paper (draws a line).
  - p2: Pen lifts, stopping drawing.
  - p3: Drawing ends, and subsequent points are ignored.

III. Model Architecture

1. Single Model Architecture

The chosen architecture is a Conditional Recurrent Neural Network (RNN) Decoder. Unlike the original SketchRNN VAE which encodes the entire sequence into a latent vector, this model focuses directly on sequential generation conditioned on the class label.
Core Components:

- **Class Embedding Layer:** An nn.Embedding layer maps each discrete class index to a dense, learnable vector representation (`embedding_size`).
- **Conditional LSTM Decoder:** A single-layer unidirectional LSTM (`hidden_size`) serves as the core sequence generator.
- **Initialization Layer:** A fully connected layer (`fc_init_hc`) takes the class embedding as input and generates the initial hidden and cell states (h_0, c_0) for the LSTM, thereby conditioning the starting point of the generation on the desired class.
- **Output Head (GMM + Pen States):** A final fully connected layer (`fc_params`) takes the LSTM hidden state at each time step and predicts the parameters for the next stroke's distribution. This includes:
- Parameters for a Gaussian Mixture Model (GMM) with M = [Your hp.M] components to model the (Δx, Δy) distribution: mixture weights (π), means (μ_x, μ_y), standard deviations (σ_x, σ_y), and correlation (ρ_{xy}).
- Probabilities (q) for the three pen states (p1, p2, p3) via a softmax layer.

2. Router Based Model Architecture

The implemented model is inspired by the Sketch-RNN architecture , a sequence-to-sequence Variational Autoencoder (VAE) designed for generating vector sketches. The model comprises two primary components:

1. **Encoder:**Processes the input sketch sequence and encodes it into a latent space representation.
2. **Decoder:**Takes the latent representation and generates a sequence of strokes to reconstruct the sketch.

The encoder and decoder are both implemented using Recurrent Neural Networks (RNNs), specifically Long Short-Term Memory (LSTM) networks, to effectively capture the temporal dependencies in the stroke sequences.
Encoder: The encoder processes the input sequence of strokes and outputs the parameters of a latent space distribution: mean (μ) and standard deviation (σ). These parameters are used to sample a latent vector z, which captures the essential features of the input sketch.
Decoder: The decoder generates the sketch sequentially, stroke by stroke, conditioned on

the latent vector z. At each time step, the decoder predicts the parameters of a mixture of bivariate Gaussian distributions for the next point's coordinates and the probabilities for the pen states.

IV. Approach

1. Single Model Architecture

The approach involves the following steps:

1. **Data Preprocessing:**
    - **Loading:** `.npz` files for the selected classes were loaded. Data from train, valid, and test splits within files were combined initially.
    - **Purification:** Sequences shorter than [Your hp.min_seq_length] or longer than [Your `hp.max_seq_length`] were discarded to remove noise and outliers. Large $\frac{\Delta x}{\Delta y}$ values were clamped to [-1000, 1000].
    - **Normalization:** The Δx and Δy values across the entire dataset were normalized by dividing by their collective standard deviation (`data_scale_factor` = [Calculated Value]). This stabilizes training.
    - **Splitting:** The normalized dataset (pairs of (sequence, `class_index`)) was split into training (70%), validation (15%), and test (15%) sets using stratified sampling to maintain class proportions.
    - **Batching:**
        - Sequences within a batch were padded to the maximum length found in the processed datasets (Nmax = [Calculated Value]).
        - Each point was converted to the 5-element format [Δx, Δy, p1, p2, p3].
        - The `make_batch` function generated batches containing the padded stroke sequences, original sequence lengths (before padding), and corresponding class indices, all moved to the appropriate compute device ([cpu or cuda]).
2. **Model Architecture Design:**
    - The Conditional RNN Decoder architecture was chosen for its relative simplicity compared to the VAE, focusing computation directly on the sequential generation task.
    - **Conditioning Mechanism:** Class conditioning is achieved in two ways:
        - **Initial State:** The initial hidden and cell states of the LSTM are generated directly from the learned embedding of the target class via the `fc_init_hc` layer. This sets the "context" for the drawing.
        - **Per-Step Input:** At every time step during both training (teacher forcing) and generation (autoregressive), the class embedding is concatenated with the current stroke input and fed into the LSTM. This provides continuous guidance about the target object throughout the sequence generation.

- **LSTM Role:** The LSTM maintains the hidden state, capturing the temporal dependencies and the current state of the drawing process.
- **Output Layer:** The GMM output head allows the model to predict a flexible, multi-modal distribution for the next stroke's (Δx, Δy), capturing the inherent variability in drawing. The softmax output for pen states models the discrete decisions of lifting the pen or ending the drawing.

3. **Training Strategy:**

   - **Teacher Forcing:** During training, the decoder receives the ground-truth stroke from the previous time step as input (along with the class embedding) to predict the parameters for the current ground-truth stroke. This stabilizes initial training.
   - **Optimization:** The Adam optimizer was used with an initial learning rate of [Your hp.lr].
   - **Learning Rate Scheduling:** A ReduceLROnPlateau scheduler monitored the validation loss and reduced the learning rate by a factor of [Your hp.scheduler_factor] if no improvement was seen for [Your hp.scheduler_patience] validation steps.
   - **Gradient Clipping:** Gradients were clipped to a maximum norm of [Your hp.grad_clip] to prevent exploding gradients during backpropagation.
   - **Validation:** The model's performance was periodically evaluated on the validation set using the same reconstruction loss metric.
   - **Checkpointing:** Model checkpoints (including decoder state, optimizer state, scheduler state, and best validation loss) were saved periodically and whenever a new best validation loss was achieved. The best model checkpoint was used for final generation.

4. **Loss:** The model was trained solely using a Reconstruction Loss, specifically the Negative Log-Likelihood (NLL) of the target sequence given the parameters predicted by the decoder. This loss consists of two main parts, averaged over the valid points in the batch sequences (as determined by the mask):

   - **Stroke Loss (LS):** The negative log-likelihood of the target (Δx, Δy) under the predicted Gaussian Mixture Model distribution. Calculated using the log-sum-exp trick for numerical stability over the M mixture components.
   - **Pen State Loss (LP):** The negative log-likelihood (equivalent to categorical cross-entropy) of the target pen states (p1, p2, p3) given the predicted probabilities (q).

   Note: Unlike the SketchRNN VAE, this approach does not include a Kullback-Leibler (KL) divergence loss term, as there is no latent variable z being explicitly modeled or regularized against a prior.

5. **Sequential Generation:** For generating new sketches:

- **Initialization:** Select a target `class_index`. Obtain its embedding and use `fc_init_hc` to generate the initial LSTM hidden state (h_0, c_0).
- **Start Token:** Provide a Start-of-Sequence (SOS) token (e.g., [0, 0, 1, 0, 0]) as the first input stroke s_0.
- **Autoregressive Loop:** For each subsequent step t up to Nmax:
  - Concatenate the previous generated/sampled stroke s_t-1 with the target class embedding.
  - Feed this concatenated vector into the LSTM decoder (using the hidden state from step t-1).
  - Obtain the predicted distribution parameters (π, μ, σ, ρ, q) for the next stroke s_t from the output layer.
  - Sample the next stroke (Δx, Δy, p1, p2, p3) from these distributions. Sampling involves:
    - Choosing a mixture component based on π (adjusted by temperature).
    - Sampling (Δx, Δy) from the selected bivariate Gaussian distribution (scaled by temperature).
    - Choosing a pen state based on q (adjusted by temperature).
    - Convert the sampled values back into the 5-element vector format s_t.
    - Use s_t as the input for step t+1.
  - Stop if the sampled pen state indicates End-of-Drawing (p3=1) or Nmax is reached.
- **Post-processing:** Denormalize the sampled (Δx, Δy) values using the `data_scale_factor`. Calculate cumulative positions for plotting/animation.

2. Router Based Model Architecture

The approach involves the following steps:

1. **Data Preprocessing:** Load and preprocess the stroke data from the Quick, Draw! dataset. Normalize the stroke coordinates and convert the pen states into a suitable format.

2. **Model Architecture Design:** Design the encoder and decoder networks for single classes using LSTM layers. In total 10 different encoder and decoder models are generated for five different classes and they are used as required (input given by the user). The encoder encodes the input stroke sequence into a latent vector z, while the decoder generates the stroke sequence conditioned on z.

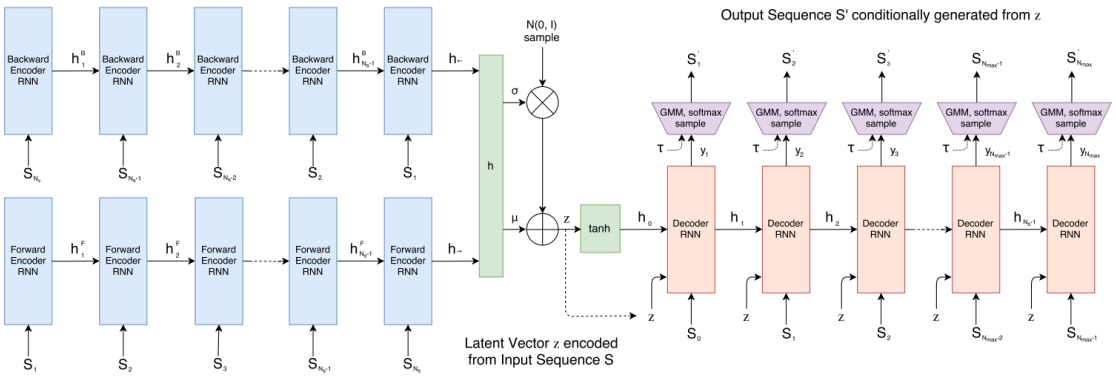

Figure 1: Model Architecture

3. **Training Strategy:** Train the model using the Adam optimizer. The loss function comprises two components:

   - **Reconstruction Loss (LR):** Measures the difference between the generated and target stroke sequences.It comprises two parts:

     (a) **Stroke Loss (LS):** Calculated using the negative log-likelihood of the bivariate Gaussian distributions modeling the pen's movements.

     $$LS = -\sum_{i=1}^{N} \log\left(\sum_{j=1}^{M} \pi_{ij} \cdot \mathcal{N}(\Delta x_i, \Delta y_i \mid \mu_{x_{ij}}, \mu_{y_{ij}}, \sigma_{x_{ij}}, \sigma_{y_{ij}}, \rho_{xy_{ij}})\right)$$

     where π_{ij} are the mixture weights, N denotes the bivariate normal distribution, and M is the number of mixtures.

     (b) **Pen State Loss (LP):** Computed using the cross-entropy between the predicted and actual pen states.

     $$LP = -\sum_{i=1}^{N} \sum_{k=1}^{3} p_{ik} \log q_{ik}$$

     where p_{ik} and q_{ik} are the actual and predicted probabilities of the pen states, respectively.

   - **Kullback-Leibler Divergence Loss (LKL):** Regularizes the latent space to follow a standard normal distribution.

     $$LKL = -\frac{1}{2} \sum_{j=1}^{z_{size}} \left(1 + \log(\sigma_j^2) - \mu_j^2 - \sigma_j^2\right)$$

4. The total loss is a weighted sum of LR and LKL.

   $$\text{Loss} = \text{LR} + \eta.\text{LKL}$$

where η is a weighting factor that balances the two loss components.

5. **Sequential Generation:** During inference, provide the class name as input to the model to generate sketches in a step-by-step manner, mimicking human drawing behavior.

V. Training & Hyperparameter

1. `classes`
   - Name of the sketch classes.
   - Value: Cat, Apple, Airplane, Candle, Alarm Clock
2. `data_location`
   - Location of the data for Sketch Classes.
   - Value: "/content/sketch_data"
3. `enc_hidden_size`
   - Number of hidden units in the encoder's LSTM.
   - Value: 256
4. `dec_hidden_size`
   - Number of hidden units in the decoder's LSTM.
   - Value: 512
5. `Nz`
   - Size of the latent vector generated by the encoder.
   - Value: 128
6. `M`
   - Number of Gaussian mixtures in the decoder's Mixture Density Network (MDN).
   - Value: 20
7. `dropout`
   - Dropout rate used in encoder and decoder to prevent overfitting.
   - Value: 0.4
8. `batch_size`
   - Number of samples processed per training batch.
   - Value: 100

9. `max_seq_length`
    - Maximum length of the input sketch sequence.
    - Value: 200
10. `eta_min`
    - Minimum learning rate for cyclical learning rate scheduling.
    - Value: 0.01
11. `R`
    - Annealing rate for KL divergence during training.
    - Value: 0.99995
12. `KL_min`
    - Minimum threshold for KL divergence.
    - Value: 0.2
13. `wKL`
    - Weight applied to the KL divergence term in the loss function.
    - Value: 0.5
14. `lr`
    - Initial learning rate for the optimizer.
    - Value: 0.9999
15. `lr_decay`
    - Learning rate decay factor after each epoch.
    - Value: 0.00001
16. `min_lr`
    - Minimum allowable learning rate.
    - Value: 0.00001
17. `grad_clip`
    - Maximum value for gradient clipping to avoid exploding gradients.
    - Value: 1.0
18. `temperature`
    - Temperature for controlling randomness during sampling from the model.

- Value: 0.4

19. `epochs`
    - Number of Epochs for training of the model.
    - Value: 50,000 for Router Based model & 20 for Single Model Architecture.

VI. Evaluation

- **Primary Metric:** Qualitative visual inspection of the generated sketches and animations for different classes. Key aspects assessed were:
  - **Recognizability:** Does the generated sketch resemble the target class?
  - **Coherence:** Are the strokes connected logically? Does the drawing process appear somewhat natural?
  - **Class Specificity:** Does the model generate distinct styles or features for different classes?
  - **Diversity:** Does generating multiple times for the same class with the same temperature yield slightly different (but still recognizable) results?
- **Secondary Metric:** Validation loss (NLL) was used to monitor training progress, guide learning rate scheduling, and select the best model checkpoint. Lower validation loss generally indicates better generalization.
- **Analysis:** Generated outputs were compared visually against examples from the Quick, Draw! dataset. Sampling temperature was varied during generation to observe its effect on randomness vs. determinism.

VII. Parameters & Model Summary

The implemented model is a Conditional RNN Decoder. During generation, the process is:

- Input: Class Index.
- Class Index → Class Embedding.
- Class Embedding → Initial LSTM State (h_0, c_0).
- Loop (t=0 to Nmax or End):
  - Input: Previous stroke s_t-1 + Class Embedding.
  - Process: LSTM step (h_t-1 → h_t).
  - Output Layer: h_t → Distribution Parameters (π, μ, σ, ρ, q).
  - Sample: Parameters → Next Stroke s_t.
  - Store/Plot: s_t.

  - Stop: If s_t indicates End-of-Drawing.

Some Example outputs:

- Cat
- Candle
- Apple
- Alarm Clock
- Airplane

VIII. Bonus Task

1. Methodology-1

- Pre-computation of Full Sequences: Unlike the sequential scene generation, this approach first generates the complete stroke sequence (x, y coordinates, and pen states) independently for each requested class using the standard conditional sampling method. All sequences are stored in memory before animation begins.
- Parallel Visualization via Subplots: The core visualization strategy relies on Matplotlib subplots. A grid layout (e.g., 2x2, 3x1) is automatically determined based on the number of classes requested, with each class assigned its own dedicated subplot within a single figure.
- Synchronized Frame Generation: The animation progresses frame-by-frame based on a shared time index, typically determined by the length of the longest generated sequence. In each frame:
- The function iterates through all subplots.
- For each subplot, it plots the corresponding pre-generated sketch up to the current time index.
- If a sketch's sequence is shorter than the current time index, it remains fully drawn (static) in subsequent frames.
- Individual Plot Scaling: Each subplot maintains its own axis limits, automatically calculated based on the extent of the specific sketch it displays. This ensures each sketch is well-framed within its panel, regardless of the size differences between sketches.
- Unified Frame Capture and Flip: After all subplots are updated for a given time index, the entire figure (containing all panels) is captured as a single image. This composite image then undergoes a vertical flip (ImageOps.flip) before being added as one frame to the final GIF animation.

Final Multiple Sketch Output

2. Methodology-2

- The core approach involved invoking the trained conditional decoder model iteratively, once for each specified object class in the desired scene sequence.
- Crucially, a coordinate management system was implemented. While each object's stroke sequence (dx, dy) is generated relative to its own start point, these were converted to absolute coordinates on a shared scene canvas.
- An offset tracker was used to determine the starting position for each subsequent object, ensuring they didn't overlap unintentionally (e.g., placing the next object horizontally adjacent to the bounding box of the previous one, plus spacing).

Final Multiple Sketch Output

IX. Conclusion

This project successfully implemented a Conditional RNN Decoder capable of generating sequential sketches conditioned on class labels. The model learned to produce recognizable, step-by-step drawings for the target classes [Airplane, Apple, Alarm Clock, Candle, Cat]. The use of class embeddings for initializing the decoder state and conditioning the per-step input proved effective in guiding the generation process. Visual evaluation confirmed the model's ability to capture class-specific features and generate coherent stroke sequences. While simpler than the SketchRNN VAE, this architecture directly addresses the objective of class-conditional sequential generation.

Potential areas for future work include:

- Training on a larger number of classes and more data per class.
- Extensive hyperparameter tuning (hidden sizes, embedding size, dropout, learning rate).
- Exploring attention mechanisms within the decoder.
- Investigating alternative RNN architectures (e.g., GRU).
- Developing more quantitative evaluation metrics beyond NLL, potentially incorporating sketch recognition models.

X. Resources

- A Neural Representation of Sketch Drawings

XI. Files

- Single Model Architecture
- Router based Architecture