—--

CS671 : Deep Learning and Applications
Assignment 06 - Report

—--

Recurrent Neural Network

Course Instructor :- Dr. Dileep AD

Ankit Mehra (S22020)
Urvashi Goswami(S22024)
Ashish Rana (S22019)

Introduction

Recurrent Neural Network (RNN)

Recurrent Neural Network is basically a generalization of a feed-forward neural network that has an internal memory. RNNs are a special kind of neural networks that are designed to effectively deal with sequential data. This kind of data includes time series (a list of values of some parameters over a certain period of time) , text documents, which can be seen as a sequence of words, or audio, which can be seen as a sequence of sound frequencies over time.
RNN is recurrent in nature as it performs the same function for every input of data while the output of the current input depends on the past one computation. For making a decision, it considers the current input and the output that it has learned from the previous input.

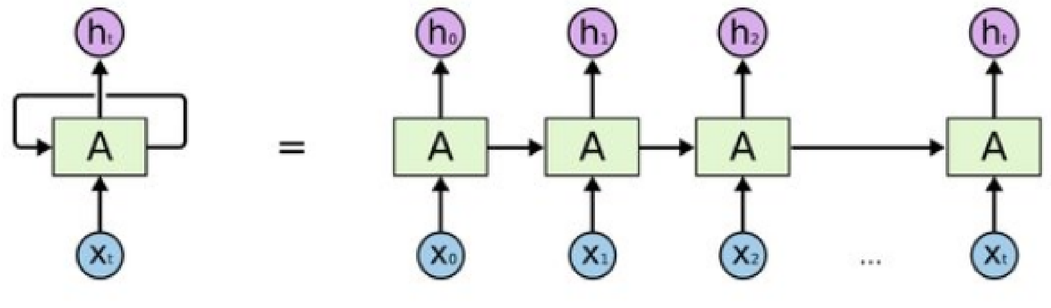


Fig- 1, A simple recurrent neural network(unfolded)
Source- ResearchGate

Unlike feed-forward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. In other neural networks, all the inputs are independent of each other. But in RNN, all the inputs are related to each other.
When we read a sentence, we read it word by word, keep the prior words / context in memory and then update our understanding based on the new words which we incrementally read to understand the whole sentence. This is the basic idea behind the RNNs — they iterate through the elements of input sequence while maintaining an internal “state”, which encodes everything which it has seen so far. The “state” of the RNN is reset when processing two different and independent sequences.
Recurrent neural networks are a special type of neural network where the outputs from previous time steps are fed as input to the current time step.
The RNNs do this by taking the output of each neuron (input nodes are fed into a hidden layer with sigmoid or tanh activations), and feeding it back to it as an input. By doing this, it does not only receive new pieces of information in every time step, but it also adds to these new pieces of information a weighted version of the previous output.

A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. This type of flow of information through time (or sequence) in a recurrent neural network is shown in the diagram below, which unrolls the sequence (loop unrolled).
First, it takes the X(0) from the sequence of input and then it outputs h(0) which together with X(1) is the input for the next step. So, the h(0) and X(1) is the input for the next step. Similarly, h(1) from the next is the input with X(2) for the next step and so on. This way, it keeps remembering the context while training.

The formula for the current state is- $h_t = f(h_{t-1}, x_t)$

Applying Activation Function: $h_t = tanh(W_{hh} h_{t-1} + W_{xh} x_t$

W is weight, h is the single hidden vector, W_{hh} is the weight at previous hidden state, W_{xh} is the weight at current input state, tanh is the activation function, that implements a Non-linearity that squashes the activations to the range[-1.1]

Output: $y_t = W_{hy} h_t$

where, Yt is the output state

Advantages of Recurrent Neural Network

- RNN can model sequence of data so that each sample can be assumed to be dependent on previous ones
- Recurrent neural networks are even used with convolutional layers to extend the effective pixel neighborhood.

Disadvantages of Recurrent Neural Network

- Gradient vanishing and exploding problems.
- Training an RNN is a very difficult task.
- It cannot process very long sequences if using *tanh* or *relu* as an activation function.

Therefore we can see the RNN doesn't learn the long-range dependencies across time steps. This makes them not very useful.
We need some sort of Long term memory, which is just what LSTMs provide.

LSTM

Long Short-Term Memory (LSTM) networks are a modified version of recurrent neural networks, which makes it easier to remember past data in memory. The vanishing gradient problem of RNN is resolved here. LSTM is well-suited to classify, process and predict time series given time lags of unknown duration. It trains the model by using back-propagation. In an LSTM network, three gates are present:

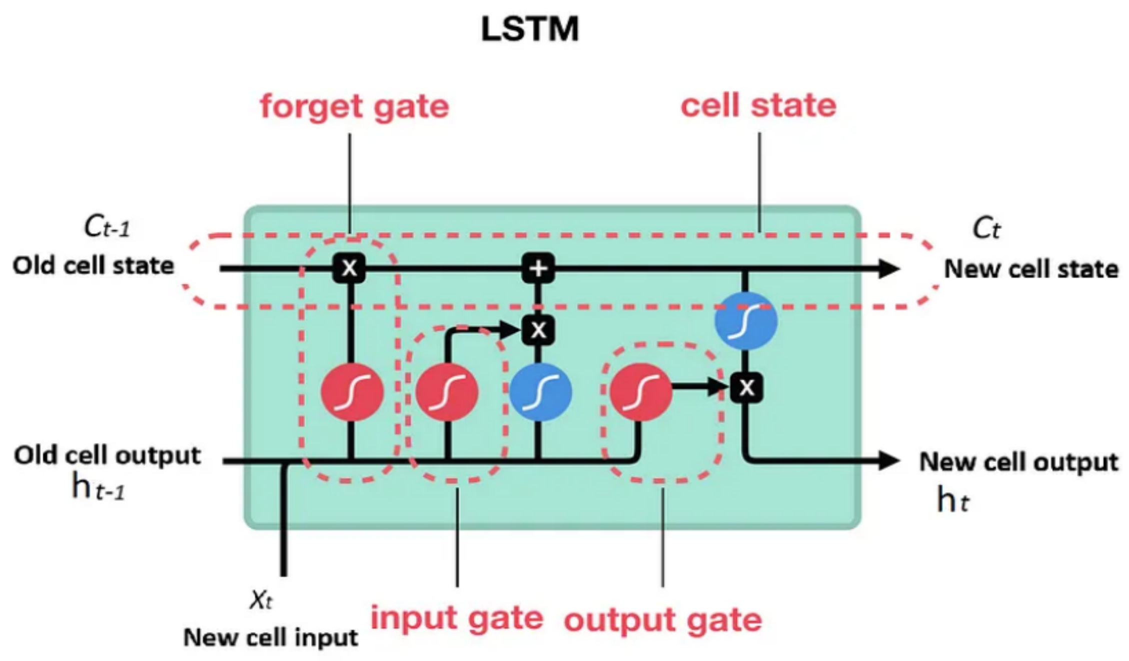


Fig 2 - LSTM cell representation
Source- Medium.com

Input gate — discover which value from input should be used to modify the memory. The Sigmoid function decides which values to let through 0,1. and tanh function gives weightage to the values which are passed, deciding their level of importance ranging from-1 to 1.

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$

$$\varsigma_t = tanh(W_c x_t + U_c h_{t-1} + b_c)$$

Forget gate — discover what details to be discarded from the block. It is decided by the sigmoid function. it looks at the previous state(ht-1) and the content input(Xt) and outputs a number between 0(*omit this*)and 1(*keep this*)for each number in the cell state Ct−1.

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \varsigma_i)$$

Output gate — the input and the memory of the block is used to decide the output. The Sigmoid function decides which values to let through 0,1. and tanh function gives weightage to the values which are passed, deciding their level of importance ranging from-1 to 1 and multiplied with the output of Sigmoid.

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$$

$$h_t = o_t \odot tanh(c_t)$$

Kaiming initialization :- Kaiming initialization, also known as He initialization, is a popular weight initialization technique .Kaiming initialization adjusts the variance of the random initialization based on the number of input and output channels of each layer. This adjustment helps in preventing the vanishing and exploding gradients problem, which can occur during the training process.

Normalized images of Handwritten characters.

Min–max normalization : Min-max normalization performs a linear transformation on the original data. This technique gets all the scaled data in the range (0, 1). The formula to achieve this is the following:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

- **Character “a”** “అ”

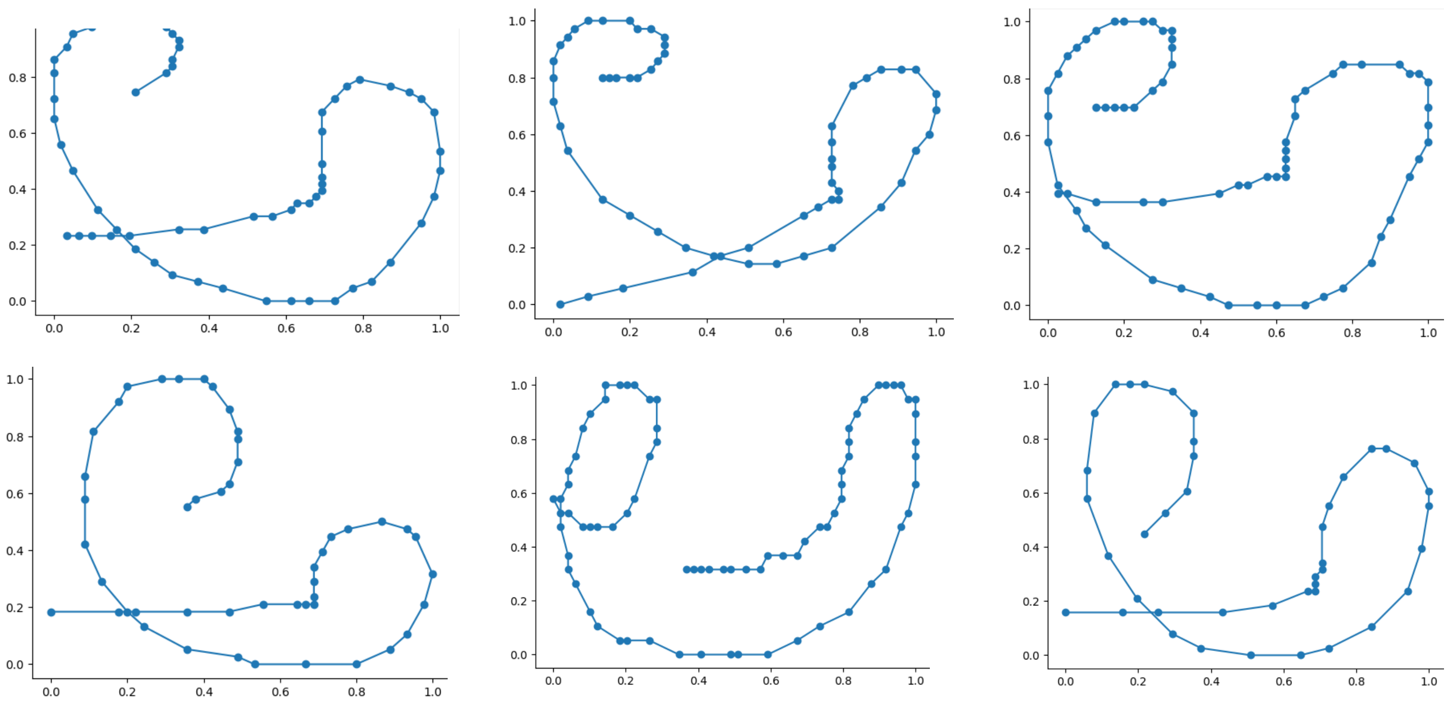
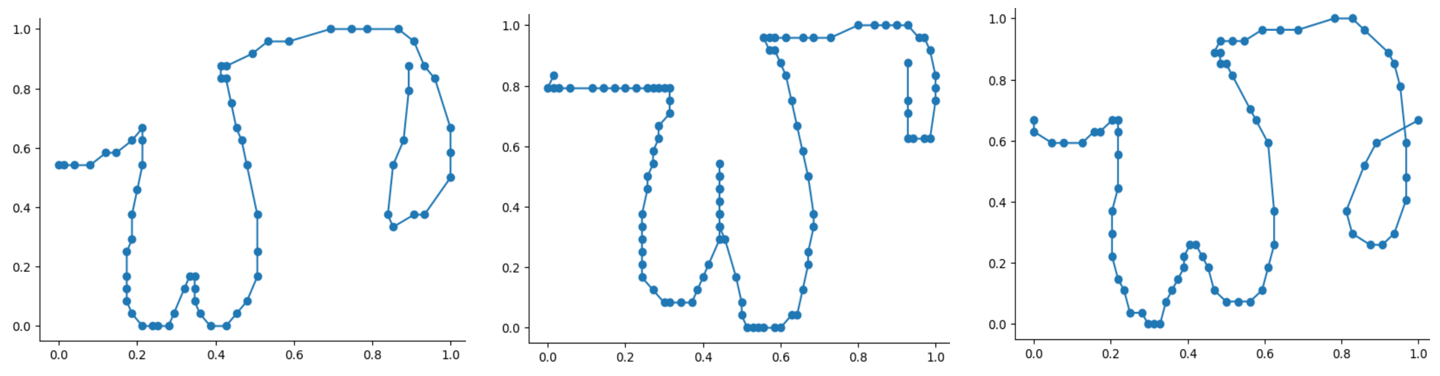


Fig- 3- Normalised image of character “a”.

- **Character “cha”** “చ”



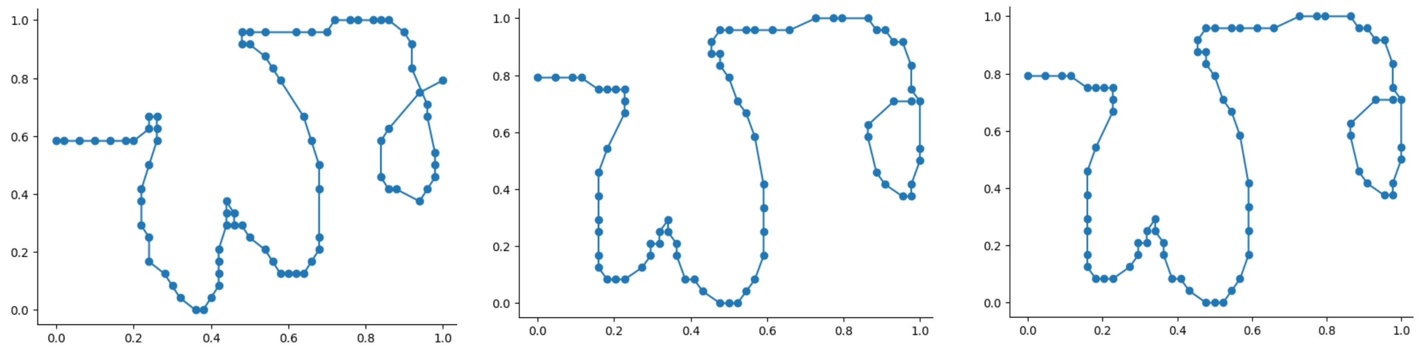


Fig- 4- Normalised image of character “chA”.

- **<u>Character “da”</u> “ದ”**

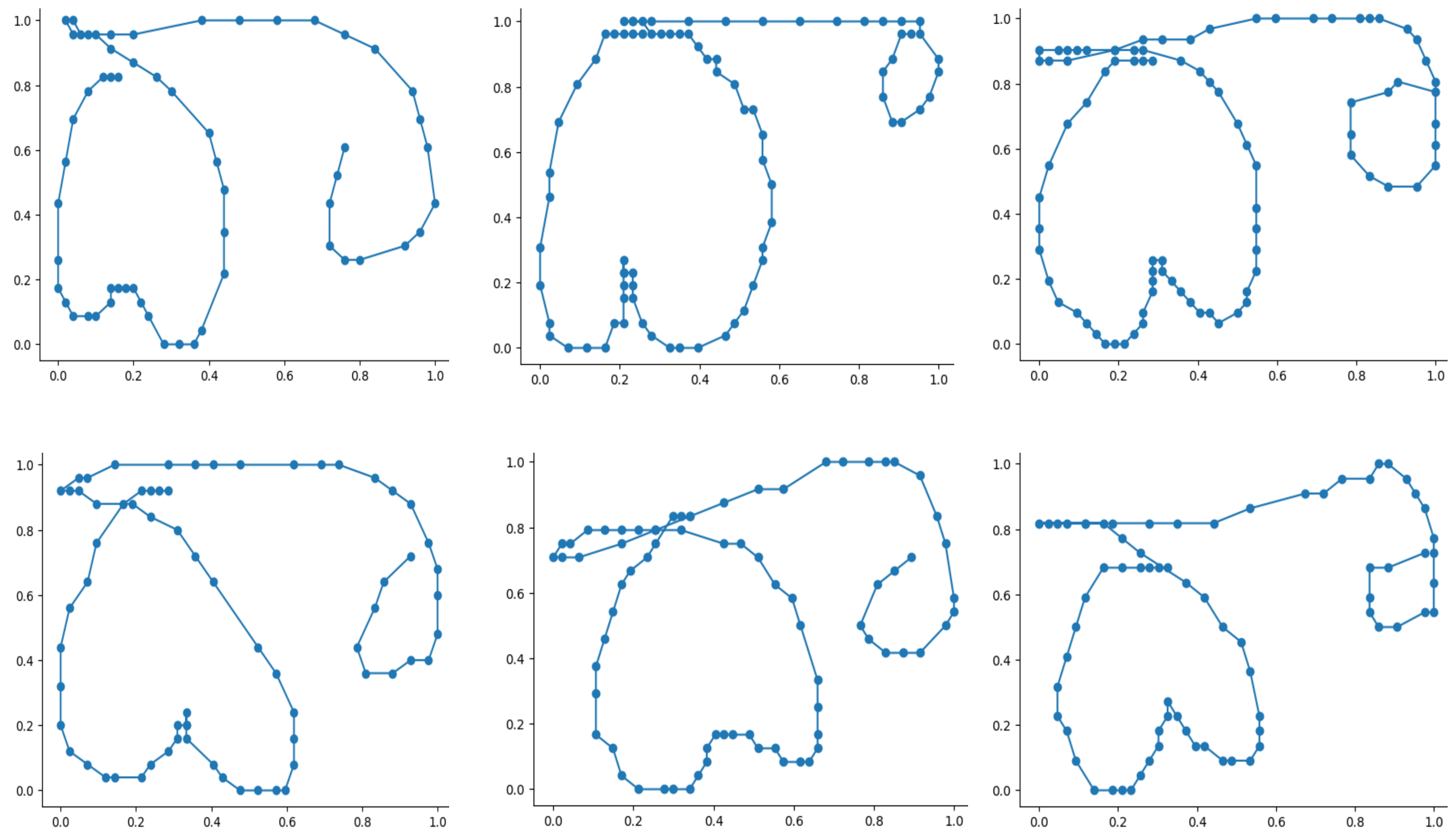


Fig- 5- Normalised image of character “da”.

- **<u>Character “la”</u> “ల”**

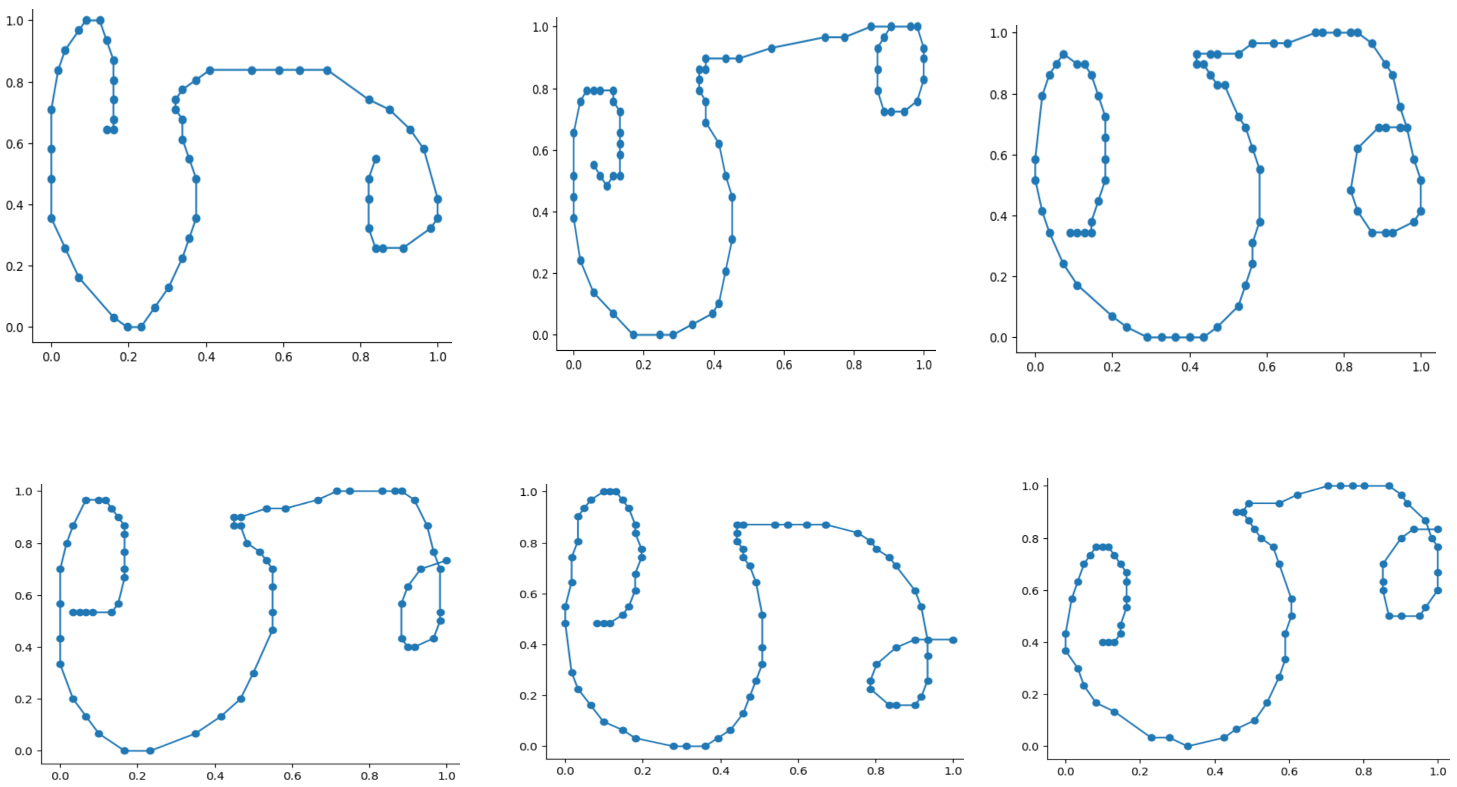
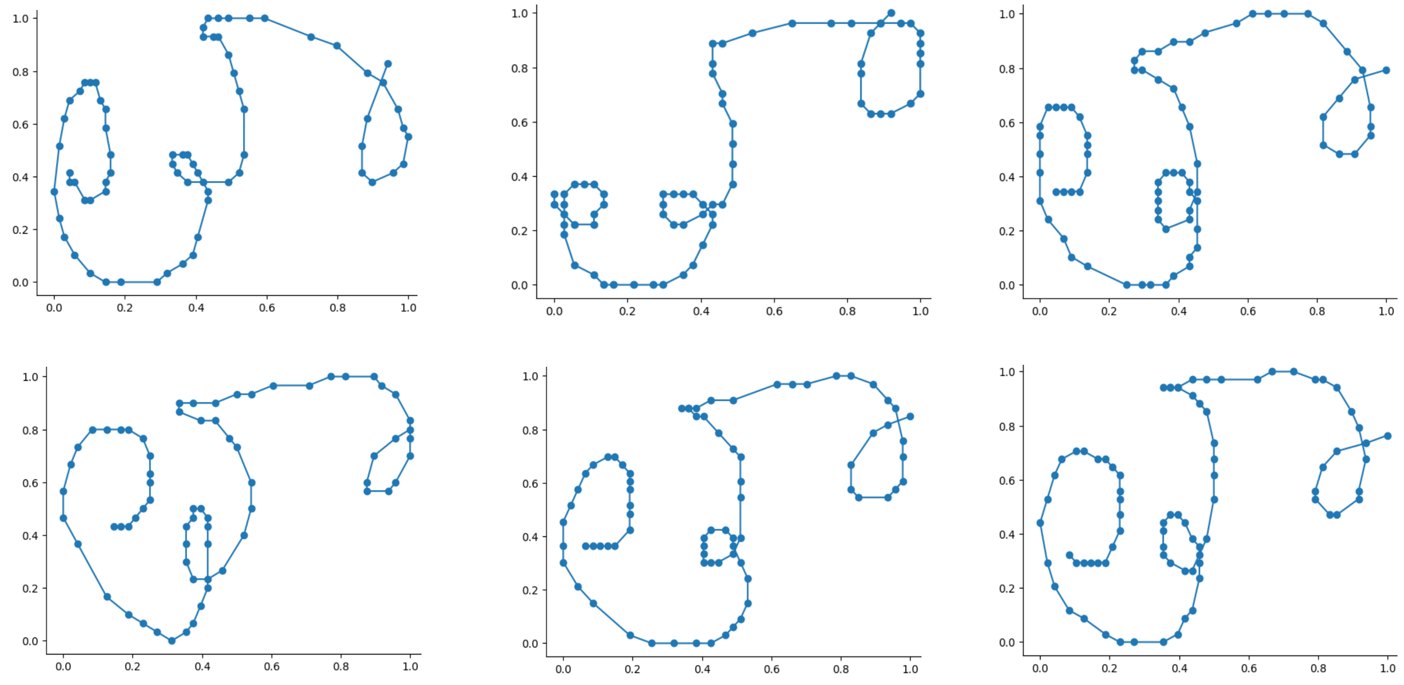


Fig- 6- Normalised image of character “la”.

- **<u>Character “ta”</u> “ట”**



DATASET - I (Handwriting Data)

Handwritten character dataset: This dataset consists of a subset of handwritten characters from Kannada/Telugu script. Each character is seen as a sequence of 2-dimensional points (x and y coordinates) of one stroke of character (pen down to pen up). There are five character classes namely: a, chA, dA, lA, tA .Each file is of .txt extension and preprocessing is needed.

Number of samples.

Table- 1 Samples

Class	Training Data	Test Data
a (అ)	**69**	**20**
chA (చ)	**70**	**20**
dA (డ)	**69**	**20**
lA (ల)	**68**	**20**
tA (త)	**69**	**20**

Values of parameters taken over all architectures for dataset 1.

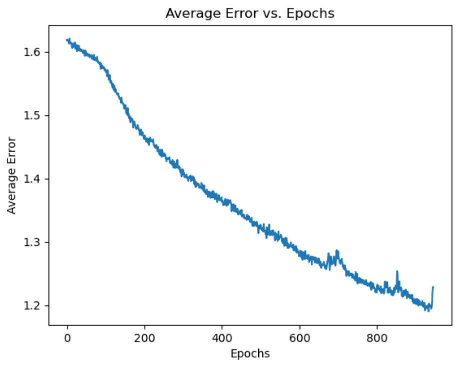
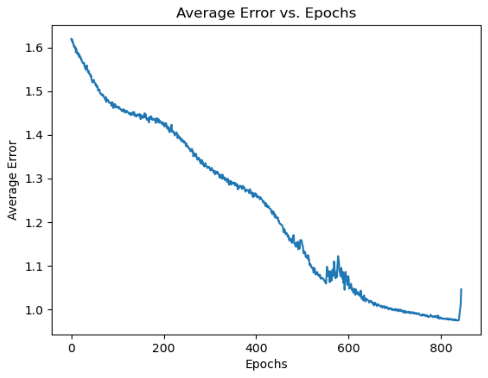
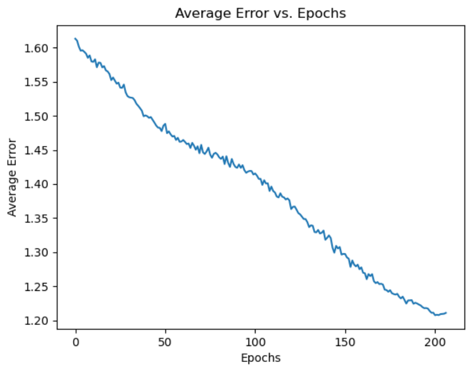
Table- 2 Values of parameters

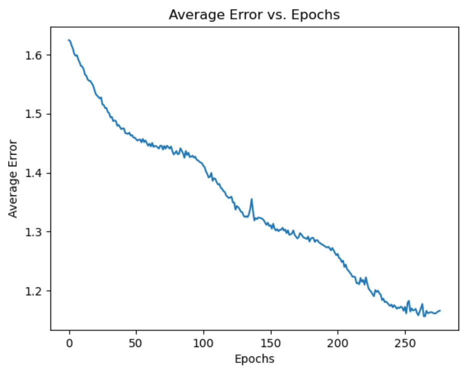
Parameters	Values
Learning Rate	0.0001
Activation (RNN)	tanh
Activation (FCNN)	Softmax
Loss Function	Cross Entropy
Stopping Criterion	Early Stopping

'tanh' is a popular activation function for RNNs because it outputs values between -1 and 1, which can help in reducing the vanishing gradient problem that is often associated with RNNs. By using the tanh activation function, the gradients can be kept within a more reasonable range, which can help in improving the training process of the RNN model.

RNN Architectures

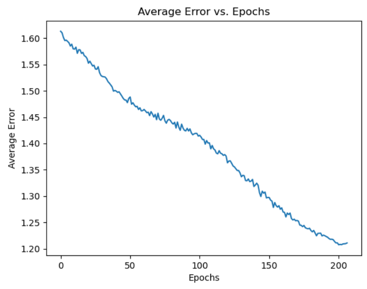
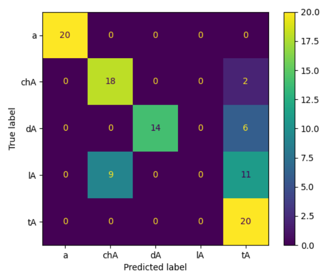
Table- 3 RNN architectures and their accuracy

Architectures	Hidden Nodes		Error V/s Epochs	Accuracy	
	L1	L2		Train	Test
Arch 1	64	-		75.36%	68.00%
Arch 2	128	-		76.30%	69.90%
Arch 3	128	128		79.20%	72.00%

Arch 4	**256**	**-**		**78.12%**	**71.00%**

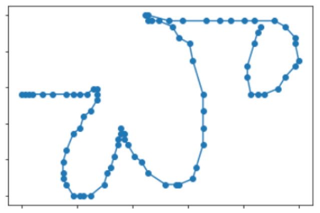
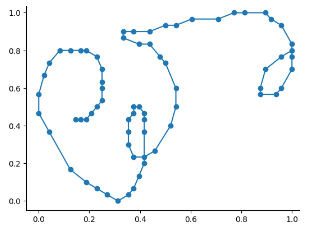
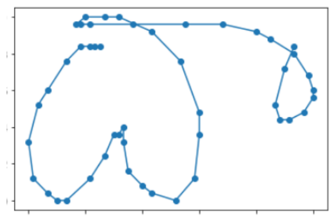
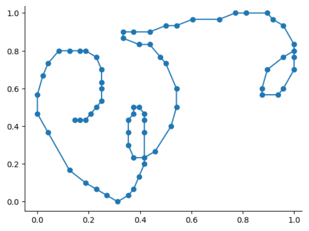
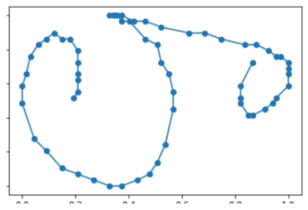
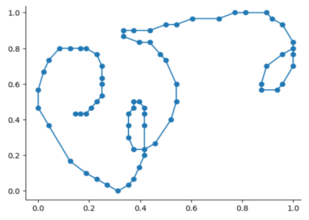
<u>Best Architecture based on test accuracy.</u>

Table- 4 Best architecture.

Architecture		**Error v/s Epoch Graph**	**Confusion Matrix**	**Accuracy**	
				Train	**Test**
128	**128**			**79.2%**	**72.0%**

Visualization of misclassified character sequences.

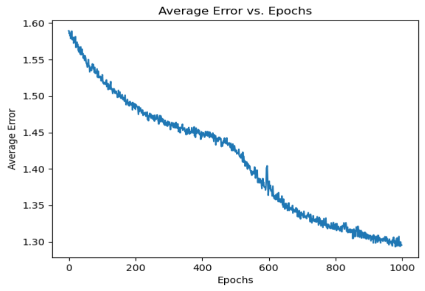
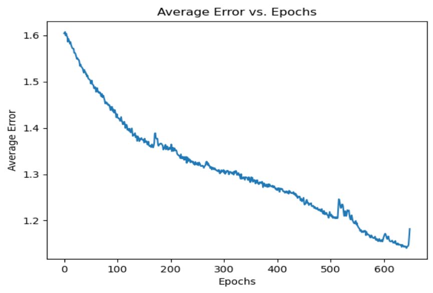
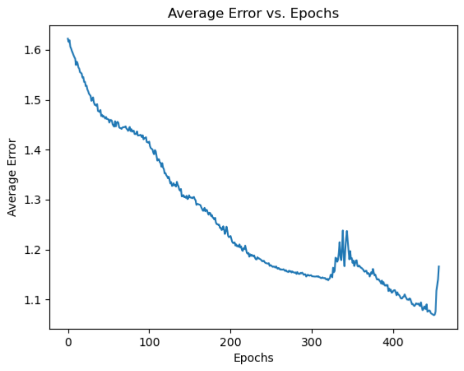
Table- 5 Misclassified characters.

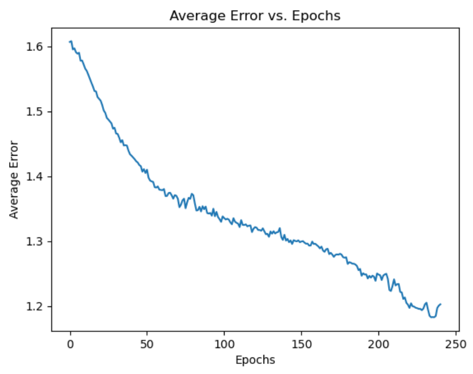
S.no.	Correct Class	Misclassified Images	Misclassified to	Misclassified class
1	a (అ)	-	-	-
2	chA (చ)			tA (త)
3	dA (ద)			tA (త)
4	lA (ల)			tA (త)
5	tA (త)	-	-	-

From the above results we can infer that some of the characters are misclassified into another class. This has happened because our data set is too small and not diverse enough, the model does not have enough information to learn all the patterns and make accurate predictions.

LSTM Architectures

Table- 6- LSTM architectures and their accuracy.

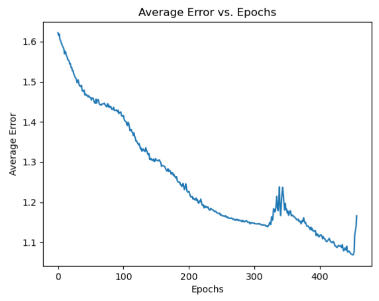
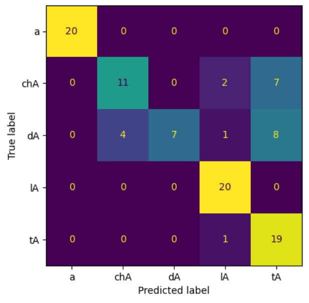
Architectures	Hidden Nodes		Error V/s Epochs	Accuracy	
	L1	L2		Train	Test
Arch 1	64	-		69.32%	61.00%
Arch 2	128	-		75.94%	70.40%
Arch 3	128	128		80.19%	77.00%

Arch 4	**256**	**-**		**77.19%**	**72.00%**

From the above results we can observe that as the model becomes more complex it converges earlier in comparison to the simpler models, this is because when a complex architecture is used, the model may be able to learn more quickly from the training data and converge to a good solution in fewer epochs. This is because a complex architecture can more effectively capture the underlying relationships and patterns in the data, allowing it to make better predictions. In contrast, a simpler architecture may require more epochs to converge to a good solution because it has fewer parameters and may not be able to capture the same level of complexity in the data.

Best Architecture based on test accuracy.

Table- 7 Best architecture.

Architecture		**Error v/s Epoch Graph**	**Confusion Matrix**	**Accuracy**	
				Train	**Test**
128	**128**			**80.2%**	**77.0%**

DATASET - II (CV_DATA)

This dataset consists of CV segments from speech data spoken in Hindi language. There is data for 5 CV segments (5-class data). The 39-dimensional Mel frequency cepstral coefficient (MFCC) features extracted frame-wise from utterances of a particular CV segment uttered by multiple people are provided in separate files corresponding to each class. Each data file is considered as one sample. Each row in a data file indicates one 39-dimensional MFCC feature vector. The number of such feature vectors (rows) depend on the duration of the speech segment.

Number of samples

Table- 8 Samples

Class	Training Data	Test Data
ka (क)	383	96
kaa (का)	510	127
ne (ने)	970	243
nii (नी)	194	48
Pa (प)	212	53

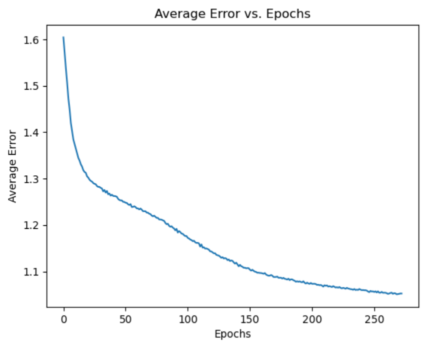
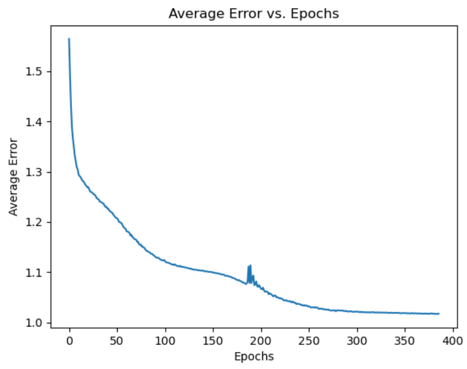
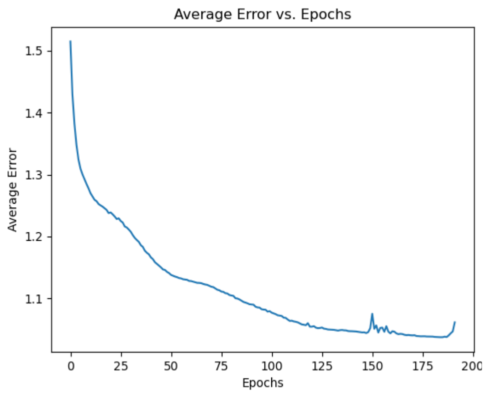
Values of parameters taken over all architectures for dataset 2.

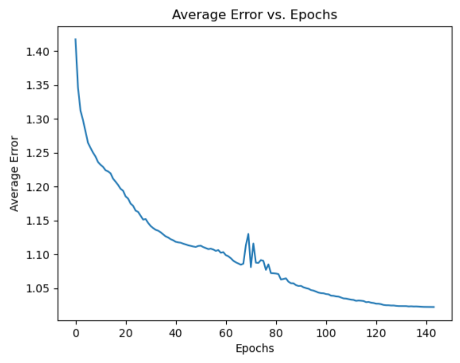
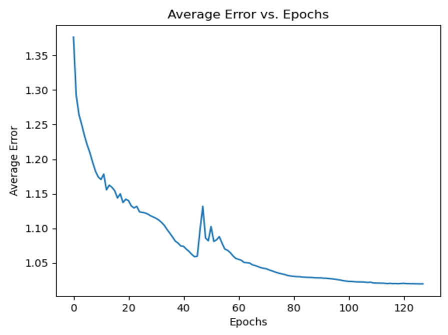
Table- 9 Value of parameters

Parameters	Values
Learning Rate	0.001
Activation (RNN)	tanh
Activation (FCNN)	Softmax
Loss Function	Cross Entropy
Stopping Criterion	Early Stopping

RNN Architectures

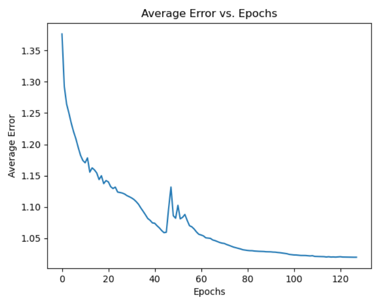
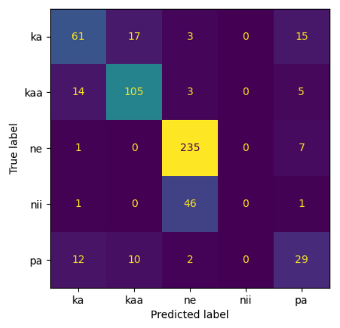
Table- 10-RNN architectures and their accuracy

Architectures	Hidden Nodes			Error V/s Epochs	Accuracy	
	L1	L2	L3		Train	Test
Arch 1	256	-	-		85.68%	71.60%
Arch 2	512	-	-		88.85%	72.84%
Arch 3	256	256	-		86.03%	74.96%

Arch 4	**512**	**512**	**-**		**88.28%**	**72.66%**
Arch 5	**512**	**512**	**512**		**89.07%**	**75.84%**

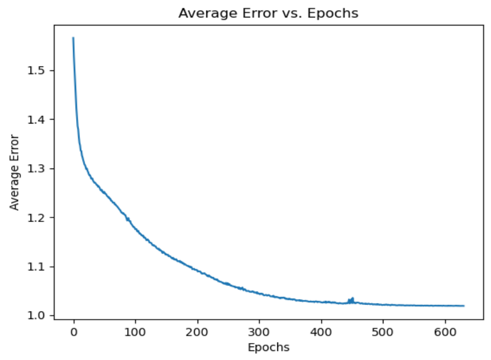
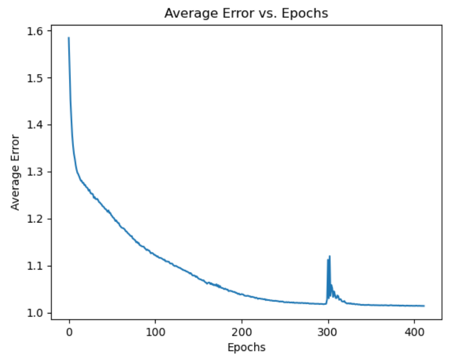
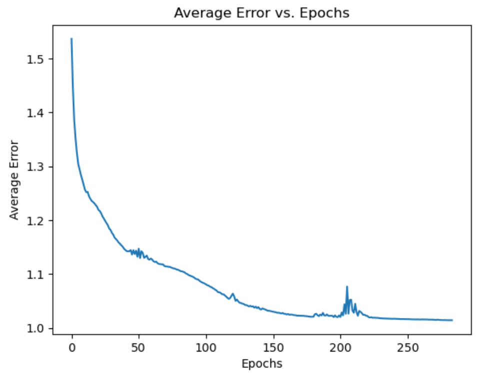
Best Architecture based on test accuracy.

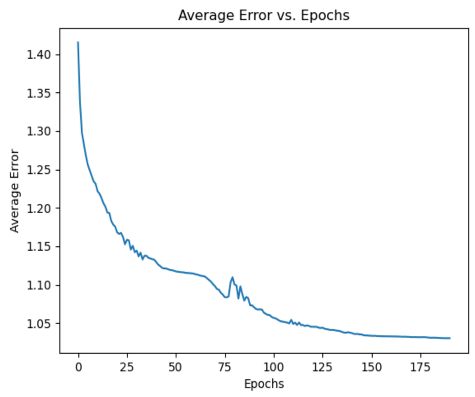
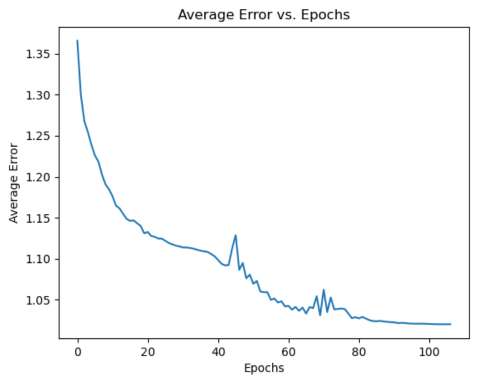
Table- 11- Best architecture.

Architecture			**Error v/s Epoch Graph**	**Confusion Matrix**	**Accuracy**	
					Train	**Test**
512	**512**	**512**			**89.1%**	**75.8%**

LSTM Architectures

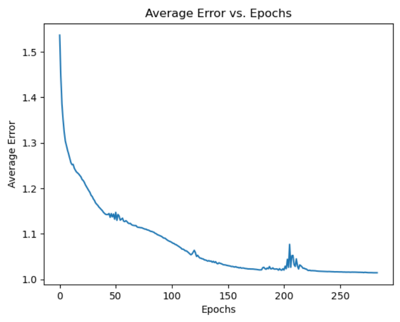
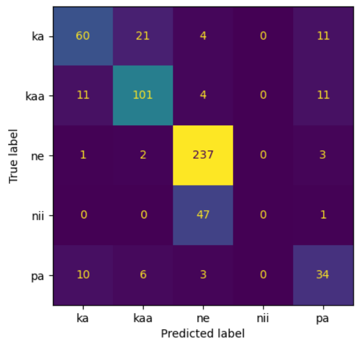
Table- 12- LSTM architectures and their accuracy.

Architectures	Hidden Nodes			Error V/s Epochs	Accuracy	
	L1	L2	L3		Train	Test
Arch 1	256	-	-		78.63%	73.02%
Arch 2	512	-	-		79.07%	74.25%
Arch 3	256	256	-		81.03%	76.19%

Arch 4	**512**	**512**	**-**		**80.40%**	**75.31%**
Arch 5	**512**	**512**	**512**		**79.50%**	**74.60%**

Best Architecture based on test accuracy.

Table- 13-Best architecture.

Architecture		**Error v/s Epoch Graph**	**Confusion Matrix**	**Accuracy**	
				Train	**Test**
256	**256**			**81.1%**	**76.2%**

From the above results we can see that many points of “nii” classes are misclassified to “ne” class , From our observation , it has happened because of **data imbalance** , ie. 970 samples of “ne” and only 194 samples of “nii”, one class has significantly more sample than the other which leads to misclassification , the model may have become biased towards the majority class and have difficulty distinguishing between the minority classes.