CS671: Deep Learning and Applications
Programming Assignment 6

RAUNAV GHOSH(M.Tech-CSP), T22104

PRASHANT DHANANJAY KULKARNI(M.Tech-CSE), T22058

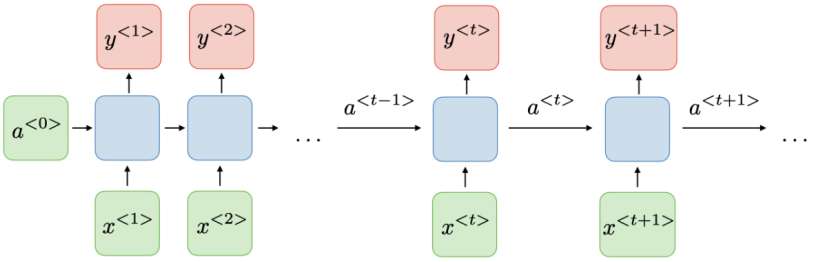
SACHIN BAHULEYAN(M.Tech-CSE), T22060

Contents

1 Introduction

Recurrent Neural Networks (RNNs) are a class of artificial neural networks that are designed to process sequential data by capturing the temporal dependencies among the input elements. Unlike traditional feedforward neural networks, RNNs have feedback connections that allow information to be passed from one time step to the next. This unique architecture makes RNNs particularly useful for a wide range of tasks such as natural language processing, speech recognition, and time series analysis. The key advantage of RNNs is their ability to process variable-length sequences, making them highly adaptable to real-world applications. In this way, RNNs have become a powerful tool in the field of deep learning, and have opened up new avenues for research and innovation in the field of machine learning.

Long Short-Term Memory (LSTM) is a specialized type of Recurrent Neural Network (RNN) that was developed to address the vanishing and exploding gradient problems that commonly occur in traditional RNNs. Like RNNs, LSTMs are designed to process sequential data and have feedback connections that allow information to be passed from one time step to the next. However, LSTMs have a unique architecture that allows them to selectively remember or forget information over long periods of time. This is achieved through the use of specialized memory cells, which are connected by gates that control the flow of information into and out of the cells. The result is a network that is capable of processing sequential data with greater accuracy and efficiency than traditional RNNs.



For each timestep t, the activation $a^{<t>}$ and the output $y^{<t>}$ are expressed as follows:

$$\boxed{a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)} \quad \text{and} \quad \boxed{y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)}$$

Figure 1: A typical schematic of an RNN, Source: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

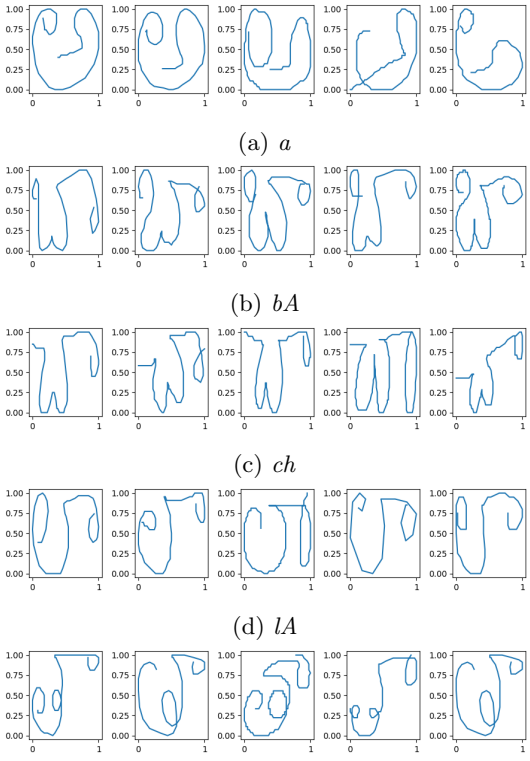
2 Problem statement

The objective of this programming assignment was to deepen our understanding of the working of recurrent neural network (RNN) and its variant, long short-term memory (LSTM). The major task includes building a RNN for classification of sequential data.
Two of the dataset given for the taskes are as follows:

1. ***Handwritten character dataset***: This dataset consists of subset of handwritten characters from Kannada/Telugu script.
2. ***Consonant Vowel (CV) segment dataset***: This dataset consists of subset of CV segments from a conversational speech data spoken in Hindi language.

3 Methodology

We start by preprocessing the datasets for the given taskes. *Handwritten charecter dataset* sequence of consists of coordinates of 2-dimension that are generated while writing this charectors on a graphics tablet. We are required to feed these sequences to an **RNN** to classify the charecter to one of the classes. The sequences are min-max normalised in the range of $[0, 1]$, such that all the sequences are normalised and belong to the same input space. Figure 2 shows some samples from each classes after normalisation.



(a) *a*

(b) *bA*

(c) *ch*

(d) *lA*

(e) *tA*

Figure 2: Normalised samples from the *Handwritten charecter dataset*

For the *CV dataset*, each sequence consists of vectors of 39-dimension. These vectors are MFCC(Mel frequency cepstral coefficient) features vector for every frame. We take the feature vectors as it is without any normalisation.

We considered a set of 6 architectures with different hyperparameters for each dataset, 3 of which are using RNNs and the rest using LSTMs. The summary of the architectures are given in Table 1 and 2.

Layer	Hidden unit	Activation function
RNN	32	*Hyperbolic Tangent*
Dense	5	*Softmax*

Layer	Hidden unit	Activation function
RNN	10	*Hyperbolic Tangent*
RNN	5	*Hyperbolic Tangent*
Dense	5	*Softmax*

Layer	Hidden unit	Activation function
RNN	64	*Hyperbolic Tangent*
RNN	32	*Hyperbolic Tangent*
Dense	200	*Rectified Linear*
Dense	5	*Softmax*

Layer	Hidden unit	Activation function
LSTM	32	*Hyperbolic Tangent*
Dense	5	*Softmax*

Layer	Hidden unit	Activation function
LSTM	10	*Hyperbolic Tangent*
RNN	5	*Hyperbolic Tangent*
Dense	5	*Softmax*

Layer	Hidden unit	Activation function
LSTM	64	*Hyperbolic Tangent*
LSTM	32	*Hyperbolic Tangent*
Dense	200	*Rectified Linear*
Dense	5	*Softmax*

Table 1: Architectures used on Handwritten dataset

Layer	Hidden unit	Activation function
RNN	64	*Hyperbolic Tangent*
Dense	5	*Softmax*

Layer	Hidden unit	Activation function
RNN	32	*Hyperbolic Tangent*
RNN	16	*Hyperbolic Tangent*
Dense	5	*Softmax*

Layer	Hidden unit	Activation function
LSTM	64	*Hyperbolic Tangent*
LSTM	32	*Hyperbolic Tangent*
Dense	128	*Rectified Linear*
Dense	5	*Softmax*

Layer	Hidden unit	Activation function
LSTM	64	*Hyperbolic Tangent*
Dense	5	*Softmax*

Layer	Hidden unit	Activation function
LSTM	32	*Hyperbolic Tangent*
RNN	16	*Hyperbolic Tangent*
Dense	5	*Softmax*

Layer	Hidden unit	Activation function
LSTM	64	*Hyperbolic Tangent*
LSTM	32	*Hyperbolic Tangent*
Dense	128	*Rectified Linear*
Dense	5	*Softmax*

Table 2: Architectures used on CV dataset

We use **Adam optimizers** for backpropagation with a learning rate of 0.0001. This was a decision we had to take due the loss oscillating more than we would like for the default larger learning rate. Stopping criteria for the training was kept as when the successive average epoch error difference falls below 10^{-4}.

For the last RNN and LSTM architectures for CV datasets, we used dropouts in the RNN layers to reduce overfitting of the models.

4 Results

4.1 RNN on Handwritten dataset

4.1.1 Architecture 1

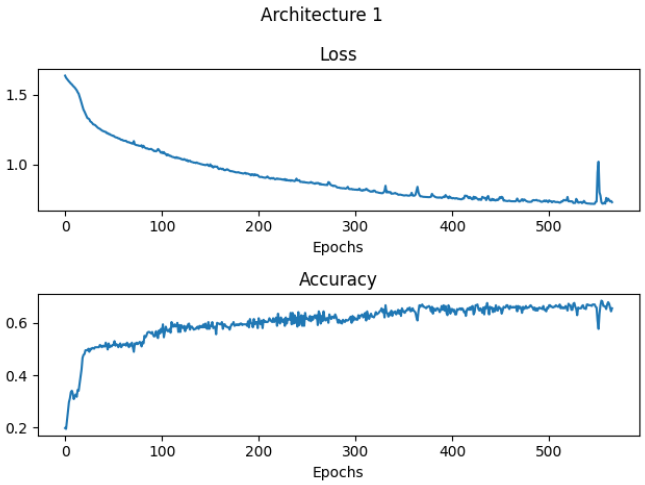


Figure 3: Architecture-1:Loss and accuracy over epochs

Dataset	**Accuracy**
Training	0.6588
Testing	0.6399

Table 3: Metrics for Architecture-1

	Pred				
Actual	a	bA	chA	lA	tA
a	20	0	0	0	0
bA	0	4	15	0	1
chA	0	4	16	0	0
lA	0	1	0	5	14
tA	0	0	0	1	19

Table 4: Confusion matrix on test dataset for architecture-1

4.1.2 Architecture 2

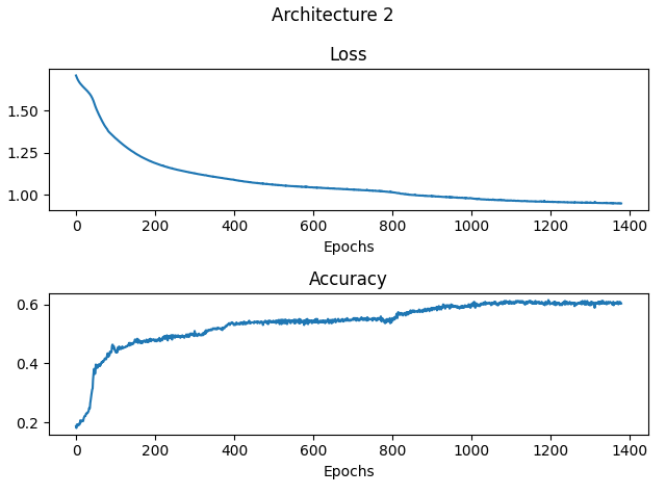


Figure 4: Architecture-2:Loss and accuracy over epochs

Dataset	**Accuracy**
Training	0.6034
Testing	0.5400

Table 5: Metrics for Architecture-2

	Pred				
Actual	a	bA	chA	lA	tA
a	20	0	0	0	0
bA	0	13	2	0	5
chA	0	7	6	0	7
lA	0	8	2	0	10
tA	0	4	1	0	15

Table 6: Confusion matrix on test dataset for architecture-2

4.1.3 Architecture 3

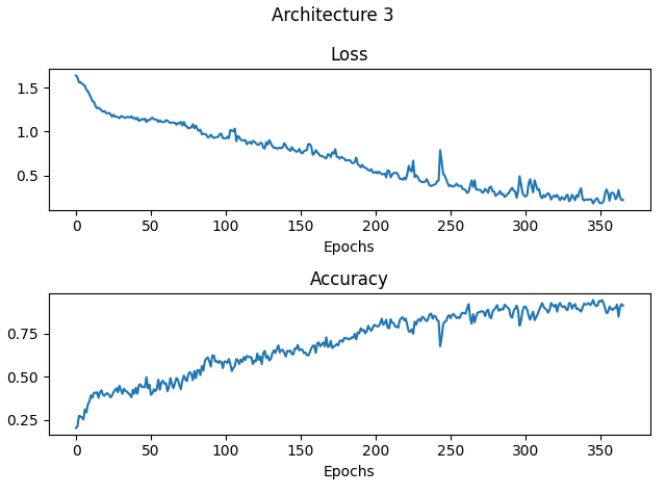


Figure 5: Architecture-3:Loss and accuracy over epochs

Dataset	**Accuracy**
Training	0.9446
Testing	0.8700

Table 7: Metrics for Architecture-3

	Pred				
Actual	a	bA	chA	lA	tA
a	20	0	0	0	0
bA	0	13	5	1	1
chA	0	3	17	0	0
lA	0	0	0	19	1
tA	0	1	0	1	18

Table 8: Confusion matrix on test dataset for architecture-3

4.2 LSTM on Handwritten dataset

4.2.1 Architecture 1

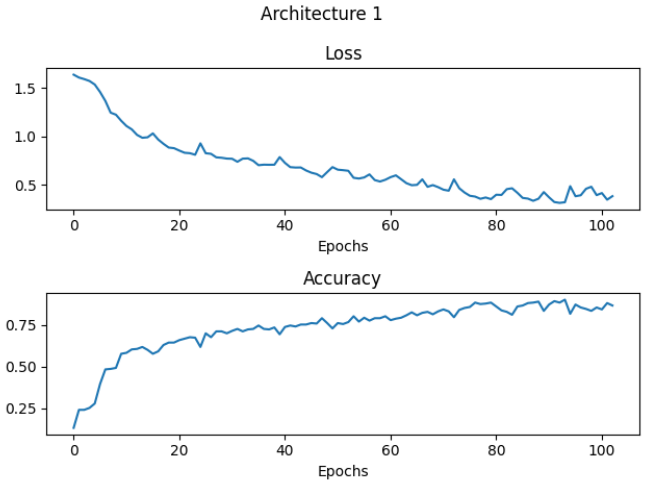


Figure 6: Architecture-1:Loss and accuracy over epochs

Dataset	**Accuracy**
Training	0.8746
Testing	0.8500

Table 9: Metrics for Architecture-1

	Pred				
Actual	a	bA	chA	lA	tA
a	20	0	0	0	0
bA	0	17	3	0	0
chA	0	4	16	0	0
lA	0	0	0	12	8
tA	0	0	0	0	20

Table 10: Confusion matrix on test dataset for architecture-1

4.2.2 Architecture 2

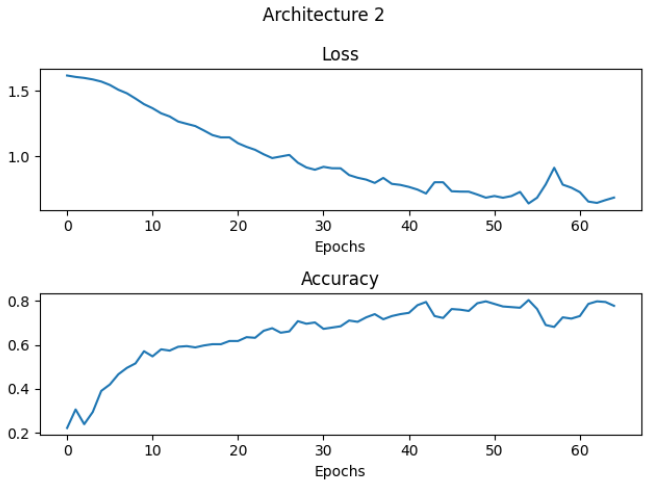


Figure 7: Architecture-2:Loss and accuracy over epochs

Dataset	**Accuracy**
Training	0.7755
Testing	0.8299

Table 11: Metrics for Architecture-2

	Pred				
Actual	a	bA	chA	lA	tA
a	19	0	0	1	0
bA	0	17	1	2	0
chA	0	6	10	0	4
lA	0	0	0	20	0
tA	0	0	3	0	17

Table 12: Confusion matrix on test dataset for architecture-2

4.2.3 Architecture 3

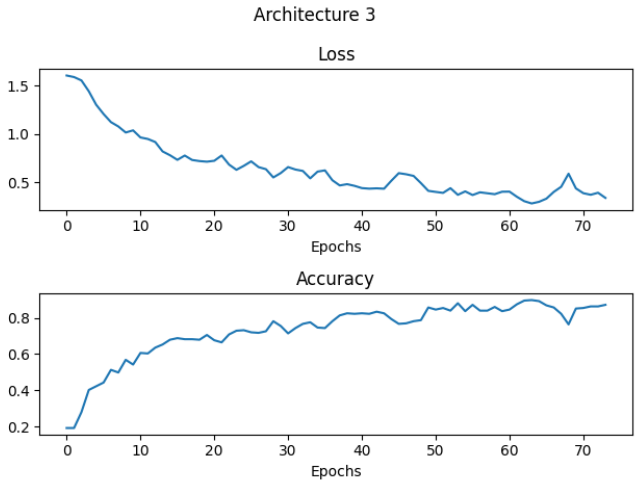


Figure 8: Architecture-3:Loss and accuracy over epochs

Dataset	**Accuracy**
Training	0.8979
Testing	0.9300

Table 13: Metrics for Architecture-3

	Pred				
Actual	a	bA	chA	lA	tA
a	20	0	0	0	0
bA	0	18	1	0	1
chA	0	4	16	0	0
lA	0	1	0	19	0
tA	0	0	0	0	20

Table 14: Confusion matrix on test dataset for architecture-3

4.3 RNN on CV dataset

4.3.1 Architecture 1

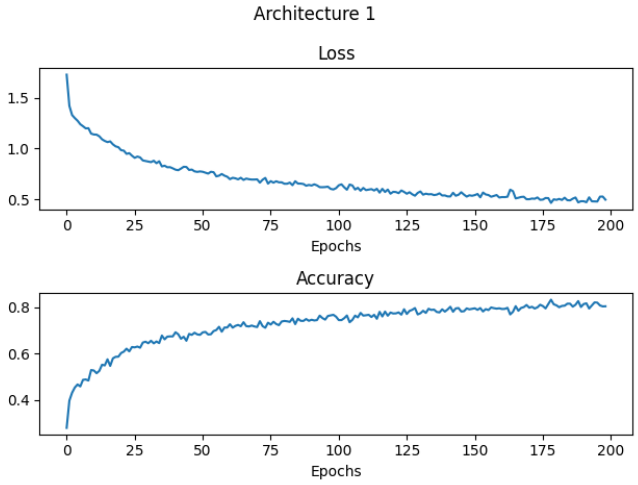


Figure 9: Architecture-1:Loss and accuracy over epochs

Dataset	**Accuracy**
Training	0.8664
Testing	0.6028

Table 15: Metrics for Architecture-1

	Pred				
Actual	a	bA	chA	lA	tA
a	53	26	8	15	0
bA	51	47	14	11	4
chA	7	8	54	26	1
lA	4	2	9	67	4
tA	0	0	1	4	75

Table 16: Confusion matrix on test dataset for architecture-1

4.3.2 Architecture 2

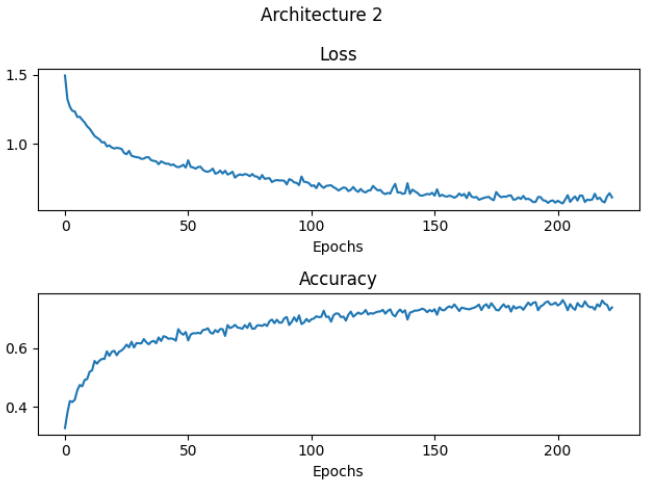


Figure 10: Architecture-2:Loss and accuracy over epochs

Dataset	**Accuracy**
Training	0.7884
Testing	0.6700

Table 17: Metrics for Architecture-2

	Pred				
Actual	a	bA	chA	lA	tA
a	41	51	4	6	0
bA	32	80	9	3	3
chA	8	17	61	7	3
lA	0	7	5	72	2
tA	0	0	3	2	75

Table 18: Confusion matrix on test dataset for architecture-2

4.3.3 Architecture 3

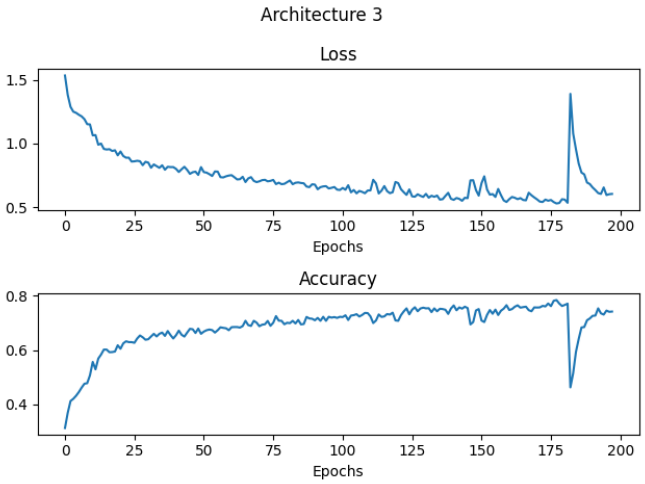


Figure 11: Architecture-3:Loss and accuracy over epochs

Dataset	**Accuracy**
Training	0.7884
Testing	0.6639

Table 19: Metrics for Architecture-3

	Pred				
Actual	a	bA	chA	lA	tA
a	33	53	4	12	0
bA	26	86	7	5	3
chA	12	14	62	6	2
lA	0	7	4	73	2
tA	0	3	3	2	72

Table 20: Confusion matrix on test dataset for architecture-3

4.4 LSTM on CV dataset

4.4.1 Architecture 1

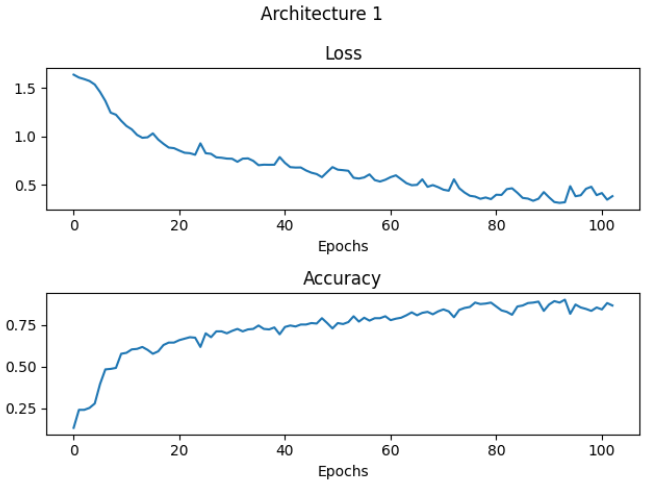


Figure 12: Architecture-1:Loss and accuracy over epochs

Dataset	**Accuracy**
Training	1.0
Testing	0.8146

Table 21: Metrics for Architecture-1

	Pred				
Actual	a	bA	chA	lA	tA
a	89	9	0	4	0
bA	16	89	19	1	2
chA	2	13	76	3	2
lA	5	0	8	72	1
tA	0	0	4	2	74

Table 22: Confusion matrix on test dataset for architecture-1

4.4.2 Architecture 2

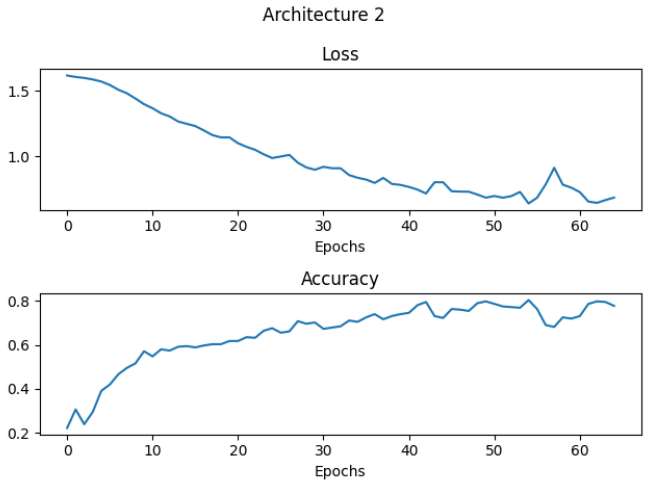


Figure 13: Architecture-2:Loss and accuracy over epochs

Dataset	**Accuracy**
Training	0.9989
Testing	0.8370

Table 23: Metrics for Architecture-2

	Pred				
Actual	a	bA	chA	lA	tA
a	96	3	0	3	0
bA	12	92	18	2	3
chA	5	18	69	2	2
lA	3	0	3	79	1
tA	0	1	2	2	75

Table 24: Confusion matrix on test dataset for architecture-2

4.4.3 Architecture 3

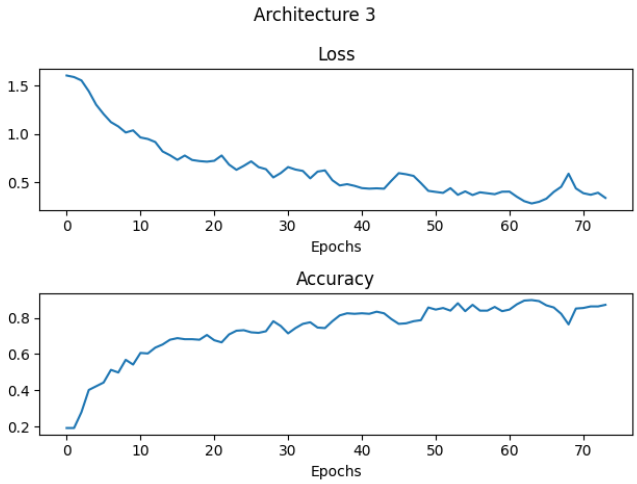


Figure 14: Architecture-3:Loss and accuracy over epochs

Dataset	**Accuracy**
Training	0.9979
Testing	0.8553

Table 25: Metrics for Architecture-3

	Pred				
Actual	a	bA	chA	lA	tA
a	91	7	0	2	2
bA	9	92	21	2	3
chA	2	5	85	3	1
lA	4	0	6	74	2
tA	0	0	2	0	78

Table 26: Confusion matrix on test dataset for architecture-3

5 Observations

1. Sequence learning can be more difficult than learning from regular data due to challenges such as long-term dependencies, variable sequence length, time complexity, noisy or missing data, and interpreting model predictions.

2. When we work with data that has a time-based structure, such as text, speech or time-series data, it can be more difficult to understand the patterns and connections between events. Sequence models are important for tasks like understanding language, recognizing speech, and predicting future values in time series data.

3. RNNs and LSTMs can be computationally expensive to train, especially for complex architectures.

4. Using APIs for modeling RNNs are little tricky.

5. The loss function for RNN can saturate quickly due to the vanishing gradient problem, which occurs when the gradients become very small as they propagate backward through time. Techniques such as gating mechanisms help gradients propagate more effectively and avoid saturation.

6. LSTMs can be more effective than simple RNNs at capturing long-term dependencies in the sequence data, but they may require more computational resources.

7. Depending on the task, different LSTM architectures (such as stacked LSTMs) may perform better than others.

8. Overfitting can be a challenge when working with sequence datasets. Regularization techniques such as dropout or weight decay can help prevent overfitting.