14/05/2023

Deep Learning Assignment-6

RNN and LSTM

Presented by:

1. R. Ramakrishna Kashyap – T22101
2. Anuj Kumar Shukla – T22103
3. Nishant Gupta- T22221

Index

1.Introduction

Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM) are two types of neural networks used for processing sequential data.

In traditional feedforward neural networks, the input and output layers are fixed, and the information flows only in one direction. In contrast, RNNs have a feedback loop that allows information to be passed from one step to the next, enabling them to model sequences of arbitrary length. The key idea behind RNNs is to use the output from the previous step as an input to the current step, which allows them to maintain information about the previous steps. LSTM is a variant of RNN that addresses the problem of vanishing gradients that can occur during the training of RNNs. The LSTM architecture includes an internal memory cell that can maintain information over long periods of time. The cell has three gates - the input gate, output gate, and forget gate - which control the flow of information into and out of the cell. The gates enable the LSTM to selectively forget or remember information from previous steps and make it well-suited for tasks involving long-term dependencies.

Both RNN and LSTM have been widely used in various applications such as natural language processing, speech recognition, time-series analysis, and image captioning, to name a few. In this assignment, we will be building RNN and LSTM models for classification of sequential data using two provided datasets.

2. Details of the task and data

Task of this assignment is to build RNN and LSTM for classification of sequential data.

We have been given two types of datasets.

Dataset-1: Handwritten character dataset

Dataset-2: Consonant Vowel (CV) segment dataset

After building the RNN and LSTM for these two datasets best model needs to be find by tuning hyperparameters.

Hyperparameters: Number of hidden layers, number of neurons in each hidden layer of the network at each timestep, number of RNN/LSTM layers etc.

Convergence Criteria: Difference between average error of successive epochs falling below a threshold of 10^{-4}.

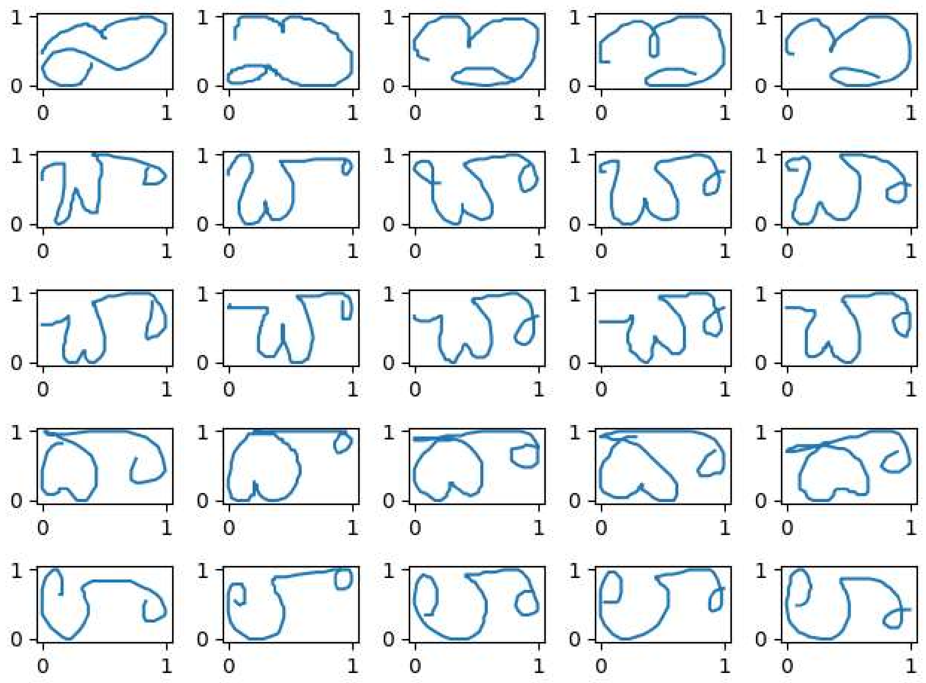


Fig.1: Normalized images from Each Handwritten character classes

3. Architectures and Parameters

Different architectures were tried & tested. Of all those four of the best architectures are discussed below. The results mentioned below are based on these four Architectures.

<u>Recurrent Neural Network</u>

Each data point was padded & reshaped to (100,39) before feeding to the input layer.

Architecture - 1

This architecture consists of an input layer of shape (200, 2), followed by masking layer, a simple recurrent layer with 5 units, flattening layer, and a dense layer with 5 units and softmax activation.

Architecture – 2

This architecture consists of an input layer of shape (200, 2), followed by masking, two layers of simple recurrent networks with 5 units each, where the second RNN layer returns only the final output unlike the 1st RNN layer where it returns the entire sequence. After this the output is flattened and passed through a dense layer with 5 units and softmax activation.

Architecture – 3

This architecture consists of an input layer of shape (200, 2), followed by masking layer, a simple recurrent layer with 32 units that returns sequences, another simple recurrent layer with 16 units, flattening layer, and a dense layer with 5 units and softmax activation.

Architecture – 4

This architecture consists of an input layer of shape (200, 2), followed by masking layer, a simple recurrent layer with 32 units that returns sequences, a dropout layer with a rate of 0.2, another simple recurrent layer with 16 units, flattening layer, and a dense layer with 5 units and softmax activation.

Long Short Term Memory

Each data point was padded & reshaped to (100,39) before feeding to the input layer.

Architecture – 1

This architecture consists of an input layer of shape (200, 2), followed by masking layer, an LSTM layer with 5 units, flattening layer, and a dense layer with 5 units and softmax activation.

Architecture – 2

This architecture consists of an input layer of shape (200, 2), followed by masking layer, an LSTM layer with 5 units that returns sequences, another LSTM layer with 5 units, flattening layer, and a dense layer with 5 units and softmax activation**.**

Architecture - 3

This architecture consists of an input layer of shape (200, 2), followed by masking layer, an LSTM layer with 8 units that returns sequences, another LSTM layer with 16 units, flattening layer, a dense layer with 100 units, and a final dense layer with 5 units and softmax activation.

Architecture – 4

This architecture consists of an input layer of shape (200, 2), followed by masking, an LSTM layer with 32 units that returns sequences, a dropout layer with a rate of 0.2, another LSTM layer with 16 units, flattening, and a dense layer with 5 units and softmax activation.

All inputs in each architecture of both RNN and LSTM were padded with "2" to make the input sequence length same. Padding allows the network to process the sequences in batches efficiently. It ensures that all sequences have the same shape, which is required for matrix operations and memory allocation in the underlying computations. Hence masking was used to handle these padded input sequences.

All LSTM/RNN layers were used with Tan hyperbolic activation function.

4. Structure of RNN

Here are the key components of a basic RNN:

Input layer: Accepts sequential data, which is often represented as a sequence of vectors.

Hidden layer: Computes a set of hidden states based on the current input and the previous hidden state.

Output layer: Produces an output based on the current hidden state.

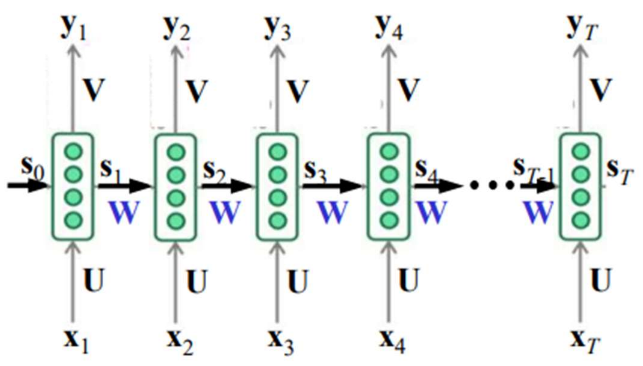
Feedback loop: Connects the hidden layer to itself, allowing information to be passed from one step to the next.

Activation function: Applies a non-linear transformation to the output of the hidden layer.

Weight matrices: Used to calculate the hidden state and output based on the input and previous hidden state.

Bias terms: Added to the weighted inputs before applying the activation function.

During training, the weights, and biases of the RNN are adjusted to minimize the error between the predicted output and the target output. The backpropagation algorithm is used to compute the gradients of the error with respect to the weights and biases, which are then used to update the parameters of the network.



$$\mathbf{s}_t = \Omega(\mathbf{U}\mathbf{x}_t + \mathbf{W}\mathbf{s}_{t-1} + \mathbf{b})$$

$$\mathbf{y}_t = \Phi(\mathbf{V}\mathbf{s}_t + \mathbf{c})$$

Fig.2: Basic structure of a RNN model

Image source: Lecture slides (CS671: Deep learning and Applications, Dr. Dileep A.D.)

5. Working of RNN

The working of an RNN architecture involves processing sequential data by recursively applying the same set of operations at each time step. Here is a step-by-step explanation of how it works:

- At each time step t, the RNN accepts an input vector x_t, which represents the current data point in the sequence.
- The RNN also receives a hidden state vector s_{t-1}, which represents the state of the network at the previous time step.
- The input vector and the previous hidden state are combined to produce a new hidden state h_t using a set of weights and biases. This is done using a non-linear activation function such as the hyperbolic tangent or ReLU.
- The new hidden state is then used to produce an output vector y_t at the current time step using another set of weights and biases. This output can be used for classification, regression, or other tasks.
- The output vector y_t can also be fed back into the network as the input for the next time step, along with the new hidden state h_t.

This process is repeated for every time step in the sequence, with the hidden state and output vector being updated at each step. The feedback loop allows the network to maintain a memory of the previous inputs, which is useful for processing sequential data such as time series or natural language. The weights and biases of the network are updated during training using the backpropagation algorithm to minimize the difference between the predicted and actual output.

6. Structure of LSTM

A Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN) that is designed to better handle the vanishing gradient problem that can occur in traditional RNNs. The LSTM architecture consists of several components that work together to allow the network to learn and remember information over long time periods. Here are the main components of an LSTM:

Input gate: This gate determines which information from the current input will be used to update the cell state. It takes the input vector and the previous hidden state as inputs and produces an output vector that represents which information should be allowed into the cell state.

Forget gate: This gate determines which information from the previous cell state should be forgotten. It takes the input vector and the previous hidden state as inputs and produces an output vector that represents which information should be removed from the cell state.

Cell state: This is the memory of the network. It maintains a long-term memory of the input sequence and can be modified by the input and forget gates.

Output gate: This gate determines which information from the current cell state should be used to produce the output. It takes the input vector and the previous hidden state as inputs and produces an output vector that represents which information should be used in the final output.

Hidden state: This is the output of the LSTM cell. It is computed from the current cell state and the output gate, and is used as the input for the next LSTM cell in the sequence.

Each of these components is a neural network layer with its own set of weights and biases. During training, the weights and biases of the LSTM are updated using the backpropagation algorithm to minimize the difference between the predicted and actual output. By selectively remembering or forgetting information from the input sequence, the LSTM can effectively handle long-term dependencies and improve the performance of the network on tasks such as natural language processing, speech recognition, and time series prediction.

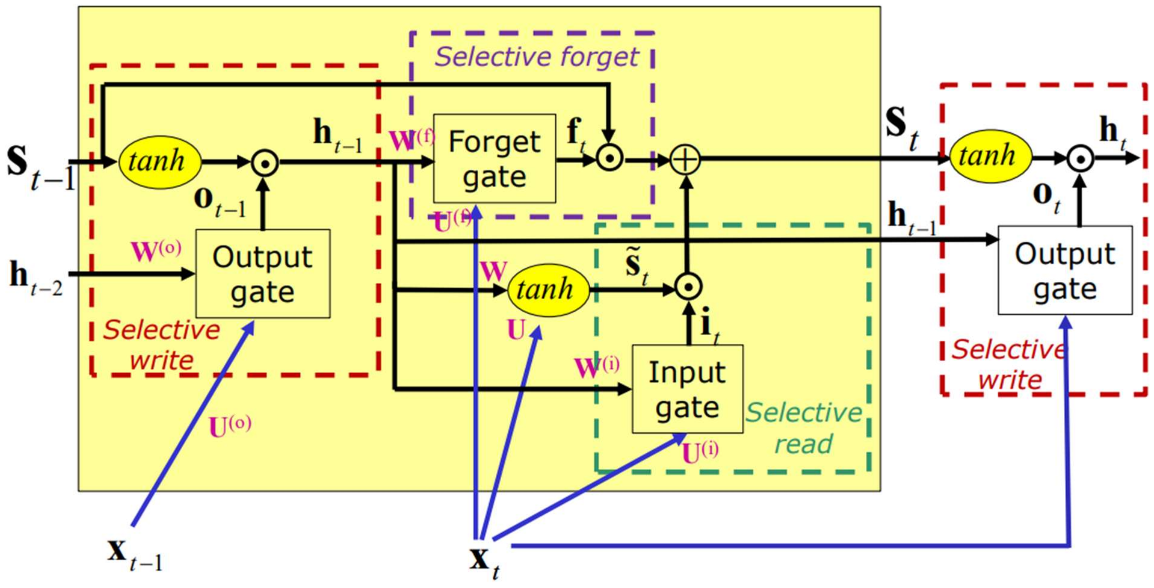


Fig.3: Basic structure of a LSTM model

Image source: Lecture slides (CS671: Deep learning and Applications, Dr. Dileep A.D.)

7. Working of LSTM

The working of a Long Short-Term Memory (LSTM) network involves the flow of information through several components or gates that operate together to enable the network to capture long-term dependencies in sequential data. Here is a step-by-step description of how an LSTM processes input sequences:

- The LSTM receives a sequence of inputs, one at a time, and processes each input in turn. Each input is typically a vector of features, such as the embedding of a word in natural language processing or a frame of audio in speech recognition.
- At each time step, the input is first passed through the input gate, which decides which information to store in the cell state. The input gate takes the input vector and the previous hidden state as input and produces an output vector that represents which information should be allowed into the cell state.
- Next, the forget gate determines which information from the previous cell state should be retained or forgotten. The forget gate takes the input vector and the previous hidden state as input and produces an output vector that represents which information should be removed from the cell state.
- The cell state, which acts as the memory of the LSTM, is updated by selectively adding or removing information from the input and forget gates. The cell state is also known as the "long-term memory" of the network.
- The output gate decides which information from the current cell state should be used to produce the output. The output gate takes the input vector and the previous hidden state as input and produces an output vector that represents which information should be used in the final output.
- Finally, the hidden state, which is the output of the LSTM cell, is computed from the current cell state and the output gate. The hidden state is also known as the "short-term memory" of the

network and is used as the input for the next LSTM cell in the sequence.

- During training, the weights and biases of the LSTM are updated using the backpropagation algorithm to minimize the difference between the predicted and actual output.
- By selectively remembering or forgetting information from the input sequence, the LSTM can effectively handle long-term dependencies and improve the performance of the network on tasks such as natural language processing, speech recognition, and time series prediction.

8. Results and Inferences

Handwritten Characters Dataset

Recurrent Neural Network

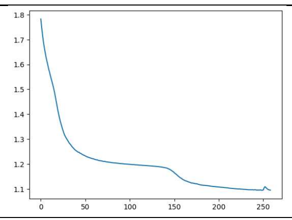
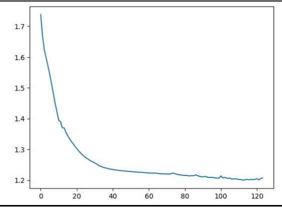
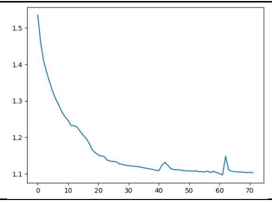
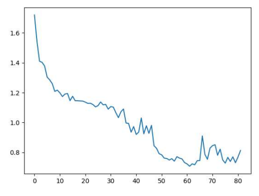
Architectures	Training Accuracy	Testing Accuracy	Error vs Epochs
I	46.8%	41%	
II	35.46%	42%	
III	44.76%	40%	
IV	60.46%	60%	

Table.1: Accuracy and Error v/s epochs plot for various architectures

According to test accuracy it can be concluded that fourth architecture is best among all the architectures.

		Actual Class				
Predicted Class		**ai**	**bA**	**chA**	**dA**	**IA**
	ai	20	0	0	0	0
	bA	0	11	7	0	2
	chA	0	10	10	0	0
	dA	0	3	1	15	1
	IA	0	9	6	0	5

Long Short Term Memory

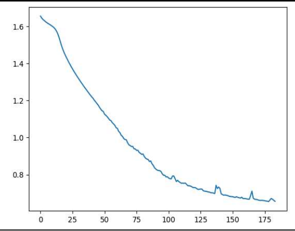
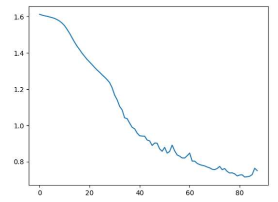
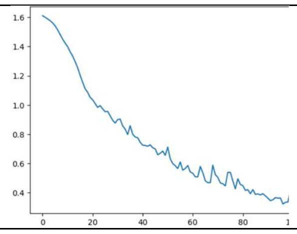
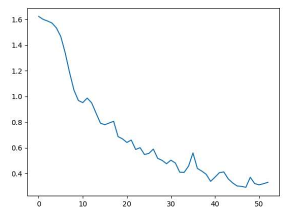
Architecture	Training Accuracy	Testing Accuracy	Error vs Epoch
I	67.15%	67%	
II	61.62%	61%	
III	71.8%	85%	
IV	92.73%	93%	

Table.2: Accuracy and Error v/s epochs plot for various architectures

According to test accuracy it can be concluded that fourth architecture is best among all the architectures.

		Actual Class				
		ai	**bA**	**chA**	**dA**	**IA**
Predicted Class	**ai**	20	0	0	0	0
	bA	0	15	5	0	2
	chA	0	4	15	0	1
	dA	0	0	3	16	1
	IA	0	1	4	2	13

Consonant Vowel Dataset

Recurrent Neural Network

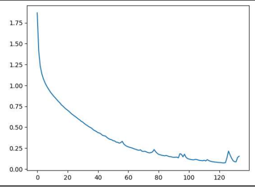
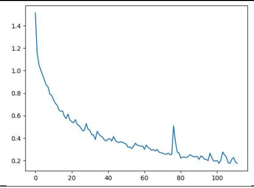
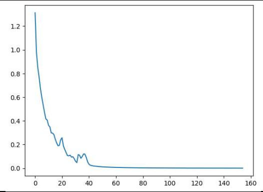
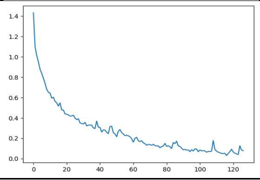
Architectures	Training Accuracy	Testing Accuracy	Error vs Epochs
I	95.8%	70.5%	
II	96.68%	74.1%	
III	100%	76.61%	
IV	98.38%	70.86%	

Table.3: Accuracy and Error v/s epochs plot for various architectures

Based on test accuracy it can be concluded that third architecture is best among all the architectures considered.

Long Short-Term Memory

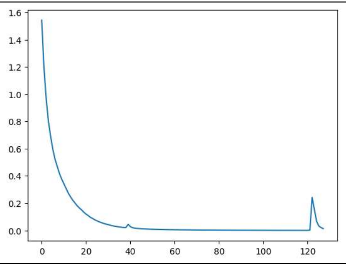
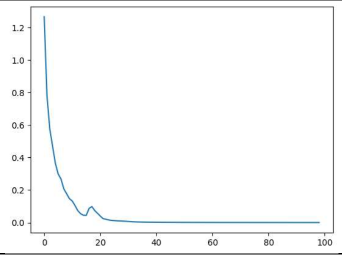
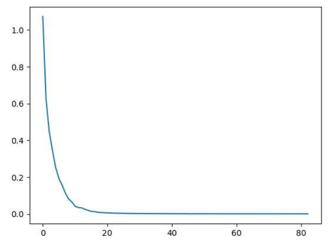
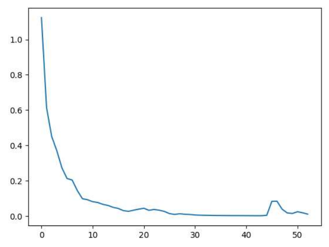
Architectures	Training Accuracy	Testing Accuracy	Error vs Epochs
I	100%	85.61%	
II	100%	84.17%	
III	100%	86.7%	
IV	99.64%	86.33%	

Table.4: Accuracy and Error v/s epochs plot for various architectures

Based on test accuracy it can be concluded that third architecture is best among all the architectures considered.

Confusion Matrix for Best architecture of RNN

	Actual Class					
		ba	**bha**	**hl**	**pa**	**re**
Predicted Class	**ba**	44	2	1	22	3
	bha	3	46	0	0	0
	hl	0	1	44	0	7
	pa	25	1	0	27	0
	re	0	0	2	5	45

Confusion Matrix for Best architecture of LSTM

	Actual Class					
		ba	**bha**	**hl**	**pa**	**re**
Predicted Class	**ba**	56	1	1	12	3
	bha	2	47	0	0	0
	hl	0	1	50	0	1
	pa	1	1	0	50	1
	re	0	0	1	4	47

Above confusion matrices were built using test dataset on 3rd Architecture, which was found to be best performing among other architectures considered

- It can be observed the RNN was giving below 50% accuracy. This may be because it is struggling to capture long-term dependencies in sequential data. Here the patterns span over many time steps, the inherent limitations of standard RNN architectures may hinder their ability to capture and retain relevant information over longer sequences.
- Also RNNs are prone to the vanishing or exploding gradient problem, where gradients can diminish exponentially or grow extremely large during training. This can hinder the model's ability to effectively learn and update the parameters, leading to suboptimal performance and lower accuracy.
- LSTM performed much better compared to RNN. This show that it was able to capture long-term dependencies in sequential data by introducing a memory cell and gates. The memory cell allows LSTM to retain information over longer time intervals, making it more effective in capturing dependencies that span across many time steps.
- Also LSTM overcomes the problem of Vanishing gradient issue through the use of the forget gate. The forget gate enables the LSTM to control the flow of information through the memory cell, allowing it to retain important information and discard irrelevant or redundant information.