DESIGN AND DEVELOPMENT OF AUTOMATIC INDIAN SIGN LANGUANGE TO TELUGU SPEECH CONVERSION USING LSTM

A Major Project Report submitted in a partial fulfilment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY (IDP)

In

ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted by

NARLA SANJANA (18011P0413)

TULASI ANVESH (18011P0417)

CHILAKALA TEJA HARSHA (18011U0403)

MANKALA CHAITNA (18011U0406)

Under the guidance of

Dr. K ANITHA SHEELA

Professor

Department of Electronics and Communications Engineering

JNTUH College of Engineering

Jawaharlal Nehru Technological University, Hyderabad-500 085

DEPARTMENT OF ELECTRONICS AND COMMUNICATIONS ENGINEERING

JNTUH COLLEGE OF ENGINEERING

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY

HYDERABAD-500085

DECLARATION OF THE CANDIDATES

We , Narla Sanjana(18011P0413),Tulasi Anvesh(18011P0417),Chilakala Teja Harsha(18011U0403),Mankala Chaitna(18011U0406), here by certify that the project report entitled as "**DESIGN AND DEVELOPMENT OF AUTOMATIC INDIAN SIGN LANGUANGE TO TELUGU SPEECH CONVERSION USING LSTM**" is a bonafide record work done and submitted under the esteemed guidance of **Dr.K Anitha Sheela**, Professor, Department of ECE, JNTUH, in particular fulfilment of the requirements for the award of the Degree of **Bachelor Of Technology(IDP)** in **Electronics and Communication Engineering** during the academic year 2021-2022.This is a record of bonafide work carried out by us and the results embodied in this project have not been reproduced/copied from any source and have not been submitted to any other University or Institute for the award of any other degree or diploma.

Narla Sanjana(18011P0413)

Tulasi Anvesh(18011P0417)

Chilakala Teja Harsha(18011U0403)

Mankala Chaitna (18011U0406)

Department of ECE, JNTUH College of Engineering.

DEPARTMENT OF ELECTRONICS AND COMMUNICATIONS ENGINEERING

JNTUH COLLEGE OF ENGINEERING

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY

HYDERABAD-500085

CERTIFICATE BY THE SUPERVISOR

This is to certify that this dissertation work titled " **DESIGN AND DEVELOPMENT OF AUTOMATIC INDIAN SIGN LANGUANGE TO TELUGU SPEECH CONVERSION USING LSTM"** being submitted by Narla Sanjana(18011P0413),TulasiAnvesh(18011P0417),ChilakalaTejaHarsha(18011U0403) ,Mankala Chaitna (18011U0406) in partial fulfilment of the requirement for the award of degree of **BACHELOR OF TECHNOLOGY (IDP)** in **Electronics and Communication Engineering** from JNTUH during the academic year 2021- 2022 is a record of bonafide work carried out by them, under my supervision. This dissertation has not been submitted to any other university or institution for the award of any other degree. The results are verified and found to be satisfactory.

SUPERVISOR

Dr.K Anitha Sheela,

Professor,

Department of ECE,

JNTUH College of Engineering, Hyderabad-500085.

DEPARTMENT OF ELECTRONICS AND COMMUNICATIONS ENGINEERING

JNTUH COLLEGE OF ENGINEERING

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY

HYDERABAD-500085

CERTIFICATE BY THE HEAD OF THE DEPARTMENT

This is to certify that this dissertation work titled " **DESIGN AND DEVELOPMENT OF AUTOMATIC INDIAN SIGN LANGUANGE TO TELUGU SPEECH CONVERSION USING LSTM"** being submitted by Narla Sanjana(18011P0413), TulasiAnvesh(18011P0417),ChilakalaTejaHarsha(18011U0403),Mankala Chaitna (18010U0406) in partial fulfilment of the requirements for the award of the **BACHELOR OF TECHNOLOGY (IDP)** in **Electronics and Communication Engineering** from JNTUH during the academic year 2021-2022. The results embodied in this dissertation have not been submitted to any other University or Institute for the award of any Degree.

Dr.A Rajani,
M.Tech – NIT Trichy Ph.D – IIT Delhi,
Associate Professor & Head,
Electronics & Communication Engineering,
JNTUH College of Engineering,
Hyderabad –500 085.

ACKNOWLEDMENTS

We take this opportunity to exhibit our deep gratitude to all those who have extended their help by every means possible for the successful completion of this project.

We are grateful to **Dr.A.Prabhu Kumar**, Ph.D.Ind.Engg, Professor &Principal, Mechanical Engineering, JNTUH College of Engineering, Hyderabad, for his support while pursuing this project.

We thank **Dr. A Rajani**, Associate Professor & Head, Department of Electronics and Communication Engineering for her constant support.

We would like to thank our guide **Dr. Anitha Sheela Kancharla**, for her timely advice and valuable suggestions while working on this project as well as throughout the B. Tech course. She has helped to keep us focused and pointed in the right direction.

We are grateful to the Project Review Committee members and Department of Electronics and Communication Engineering who have helped in successfully completing this project by giving their valuable suggestions and support.

We thank our friends, families and all professors of ECE Department who have supported us all the way during the project.

Narla Sanjana(18011P0413)

Tulasi Anvesh(18011P0417)

Chilakala Teja Harsha(18011U0403)

Mankala Chaitna (18011U0406)

ABSTRACT

Indian sign language is used by hearing and speech impaired people to communicate with other people. This project presents a system that converts Indian sign language to Telugu speech. Our project aims to develop a model that can help these people to communicate with those who do not understand ISL and English.

Indian Sign language uses hand gestures, and these gestures are represented as a video sequence, and they contain both spatial and temporal features. In the gesture video, information lies in the sequence as well and not just in the frame. A Reccurent Neural Network has time based functionality and it uses the sequence information as well in the recognition task. For this, we are using LSTM (Long Short Term Memory) model which is a type of RNN for the detection of hand gestures and converting it into text as it is able to learn long term dependencies.

The sign to text conversion stage involves the creation of a dataset containing recorded videos of Indian Sign Language hand gestures. The pre-processing of the video gesture can be done using the Media pipe python library that detects the landmarks of the hands. MediaPipe library provides detection solutions for hands, face, pose etc. We are using MediaPipe Holistic Solution for obtaining the Key Points. MediaPipe Holistic utilizes the pose, face and hand landmark models in MediaPipe Pose, MediaPipe Face Mesh and MediaPipe Hands respectively to generate a total of 543 landmarks (33 pose landmarks, 468 face landmarks, and 21 hand landmarks per hand).

The extracted landmarks or Key points are used to decode the sign language. After preprocessing the frames of the video, the sign language gesture has to be converted to text by training and testing/classifying using Neural Networks. We are planning to record 50 Video Sequences per gesture and the data from all the sequences is combines into an array and given to the LSTM neural network model for training and testing/classifying of our system.

For the translation of text from English to Telugu, googletrans API can be used. The final stage is the text to speech conversion and the festival module in python can be used for text to speech synthesis.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

1.1 Aim of the project

To Design and Development of Automatic Indian Sign Language to Telugu Speech conversion using LSTM.

1.2 Objectives

- To create dataset.
- To perform preprocessing i.e., extract landmarks from the frames in videos.
- Conversion of ISL Gesture signs to Text using LSTM.
- Conversion of Text to Telugu speech.

1.3 Motivation for the project

- Indian Sign Language acts as a medium of communication by visually impaired, deaf and dumb people who constitute a significant portion of Indian population.
- Most people find it difficult to apprehend ISL gestures. This has created a communication gap between the hearing and speech impaired and those who do not understand ISL.
- This project aims to bridge this gap of communication by developing a model that converts Indian Sign Language to Speech.

1.4 Overview of the system

Indian Sign language uses hand gestures, and these gestures are represented as a video sequence, and they contain both spatial and temporal features. In the gesture video, information lies in the sequence as well and not just in the frame. A Recurrent Neural Network has time based functionality and it uses the sequence information as well in the recognition task.

For this, we are using LSTM (Long Short Term Memory) model which is a type of RNN for the detection of hand gestures and converting it into text as it is able to learn long term dependencies.

The project has three stages:

1.Video pre-processing stage

2.ISL to text conversion stage

3.Text to speech conversion stage

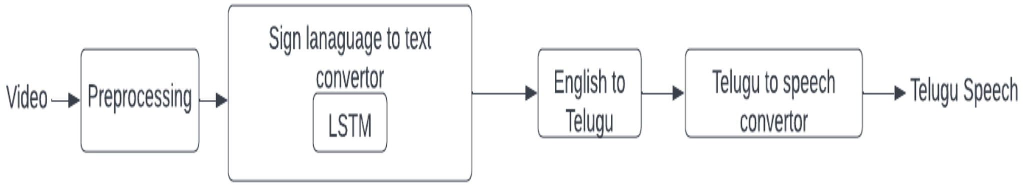


Figure 1.1 Block Diagram

1.5 Literature Survey

According to reference paper [1] they have presented a sign language gesture recognition using Argentinean sign language as their dataset. Region of hands is identified using colour based segmentation. Coloured gloves were used. According to reference paper [2] they have presented a recognition of 12 ISL gestures. Region of hands is found using skin colour based segmentation. Gesture recognition is done k-NN and HMM. The research presented in this paper pertains to ISL as defined in the Talking Hands website [3]. A Microsoft Kinect sensor is used in [4] for recognising sign languages. The sensor creates depth frames; a gesture is viewed as a sequence of these depth frames. T. Pryor et al [5] designed a pair of gloves, called SignAloud which uses embedded sensors in gloves to track the position and movement of hands, thus converting gestures to speech.

CHAPTER 2: RNN

2.1 What is RNN?

Recurrent neural networks (RNN) are a class of neural networks that are helpful in 13odelling sequence data. Derived from feedforward networks, RNNs exhibit similar behaviour to how human brains function. Simply put: recurrent neural networks produce predictive results in sequential data those other algorithms can't.

Recurrent neural networks (RNN) are the state of the art algorithm for sequential data and are used by Apple's Siri and Google's voice search. It is the first algorithm that remembers its input, due to an internal memory, which makes it perfectly suited for machine learning problems that involve sequential data. It is one of the algorithms behind the scenes of the amazing achievements seen in deep learning over the past few years.

RNNs are a powerful and robust type of neural network, and belong to the most promising algorithms in use because it is the only one with an internal memory.

Like many other deep learning algorithms, recurrent neural networks are relatively old. They were initially created in the 1980's, but only in recent years have we seen their true potential. An increase in computational power along with the massive amounts of data that we now have to work with, and the invention of long short-term memory (LSTM) in the 1990s, has really brought RNNs to the foreground.

2.2 Why RNN?

Recurrent Neural Networks (RNN) are a type of Neural Network where the output from the previous step is fed as input to the current step. RNN's are mainly used for, Sequence Classification-Sentiment Classification & Video Classification. Sequence Labelling-Part of speech tagging & Named entity recognition.

Because of their internal memory, RNN's can remember important things about the input they received, which allows them to be very precise in predicting what's coming next.

This is why they're the preferred algorithm for sequential data like time series, speech, text, financial data, audio, video, weather and much more. Recurrent neural networks can form a much deeper understanding of a sequence and its context compared to other algorithms.

2.3 How it works: RNN vs Feed-forward neural network

To understand RNNs properly, you'll need a working knowledge of "normal “feed-forward neural networks and sequential data.

Sequential data is basically just ordered data in which related things follow each other. Examples are financial data or the DNA sequence. The most popular type of sequential data is perhaps time series data, which is just a series of data points that are listed in time order.

RNN vs Feed-forward neural network

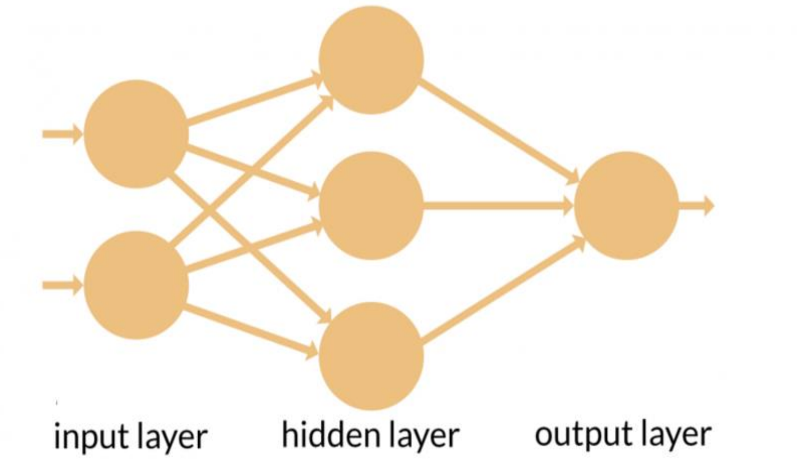


Figure 2.1 Feed-forward neural network

RNN’s and feed-forward neural networks get their names from the way they channel information.

In a feed-forward neural network, the information only moves in one direction - from the input layer, through the hidden layers, to the output layer. The information moves straight through the network and never touches a node twice. Feed-forward neural networks have no memory of the input they receive and are bad at predicting what’s coming next.

Because a feed-forward network only considers the current input, it has no notion of order in time. It simply can’t remember anything about what happened in the past except its training.

In a RNN the information cycles through a loop. When it makes a decision, it considers the current input and also what it has learned from the inputs it received previously.

The two images below illustrate the difference in information flow between a RNN and a feed-forward neural network.

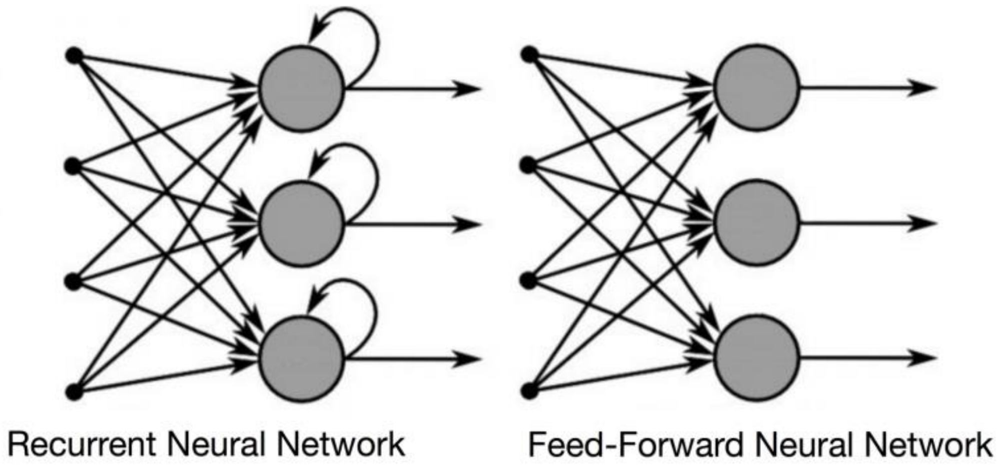


Figure 2.2 RNN vs Feed-forward neural network

A usual RNN has a short-term memory. In combination with a LSTM, they also have a long-term memory (more on that later).

Another good way to illustrate the concept of a recurrent neural network's memory is to explain it with an example: Imagine you have a normal feed-forward neural network and give it the word "neuron" as an input and it processes the word character by character. By the time it reaches the character "r," it has already forgotten about "n," "e" and "u," which makes it almost impossible for this type of neural network to predict which character would come next.

A recurrent neural network, however, is able to remember those characters because of its internal memory. It produces output, copies that output and loops it back into the network.

Simply put: recurrent neural networks add the immediate past to the present. Therefore, a RNN has two inputs: the present and the recent past. This is important because the sequence of data contains crucial information about what is coming next, which is why a RNN can do things other algorithms can't.

A feed-forward neural network assigns, like all other deep learning algorithms, a weight matrix to its inputs and then produces the output. Note that RNNs apply weights to the current and also to the previous input. Furthermore, a recurrent neural network will also tweak the weights for both through gradient descent and backpropagation through time (BPTT).

2.3.1 Types of RNNs

- One to One.
- One to Many.
- Many to One.
- Many to Many.

Also note that while feed-forward neural networks map one input to one output, RNNs can map one to many, many to many (translation) and many to one (classifying a voice).

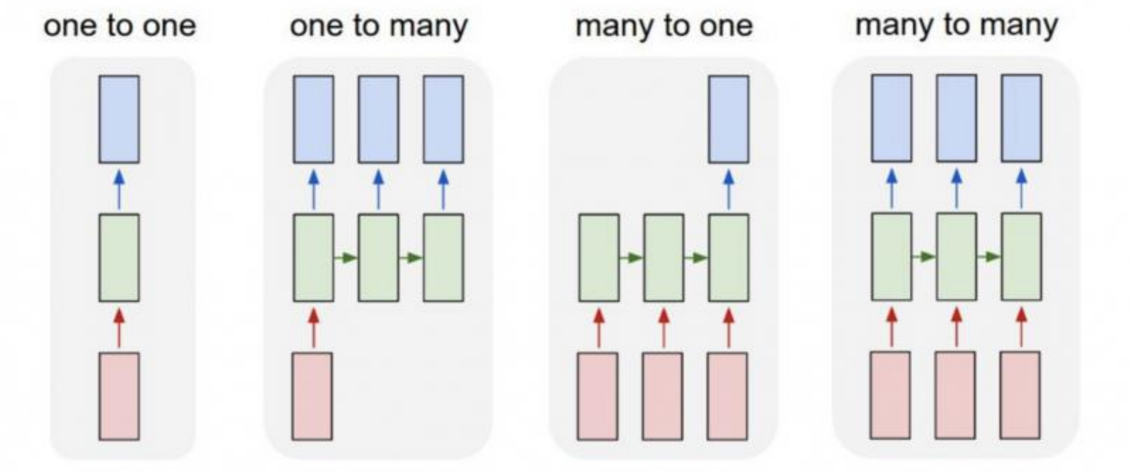


Figure 2.3 Types of RNNs

2.3.2 Backpropagation Through Time

Backpropagation (BP or backup, for short) is known as a workhouse algorithm in machine learning. Backpropagation is used for calculating the gradient of an error function with respect to a neural network's weights. The algorithm works its way backwards through the various layers of gradients to find the partial derivative of the errors with respect to the weights. Backprop then uses these weights to decrease error margins when training.

In neural networks, you basically do forward-propagation to get the output of your model and check if this output is correct or incorrect, to get the error. Backpropagation is nothing but going backwards through your neural network to find the partial derivatives of the error with respect to the weights, which enables you to subtract this value from the weights.

Those derivatives are then used by gradient descent, an algorithm that can iteratively minimize a given function. Then it adjusts the weights up or down, depending on which decreases the error. That is exactly how a neural network learns during the training process.

So, with backpropagation you basically try to tweak the weights of your model while training.

The image below illustrates the concept of forward propagation and backpropagation in a feed-forward neural network:

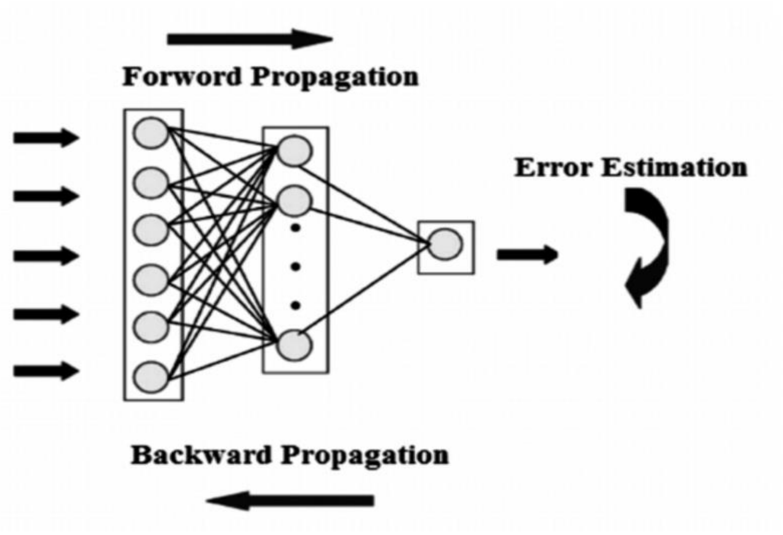


Figure 2.4 RNN Propagation

Recurrent Neural Networks are those networks that deal with sequential data. They predict outputs using not only the current inputs but also by taking into consideration those that occurred before it. In other words, the current output depends on current output as well as a memory element (which takes into account the past inputs). For training such networks, we use good old backpropagation but with a slight twist. We don't independently train the system at a specific time *'t'* We train it at a specific time *'t'* as well as all that has happened before time *'t'* like t-1, t-2, t-3.

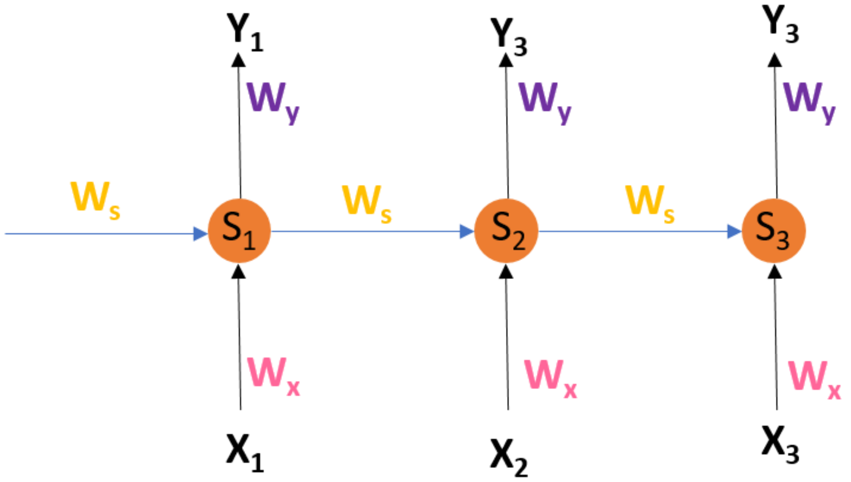


Figure 2.5 RNN Architecture

S1, S2, S3 are the hidden states or memory units at time *t1*, *t2*, *t3* respectively, and *Ws* is the weight matrix associated with it.
X1, *X2*, *X3* are the inputs at time *t1*, *t2*, *t3* respectively, and *Wx* is the weight matrix associated with it.
Y1, *Y2*, *Y*3 are the outputs at time *t1*, *t2*, *t3* respectively, and *Wy* is the weight matrix associated with it.

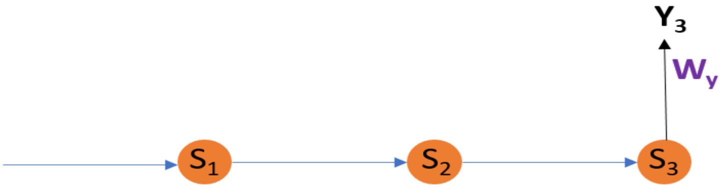
$$S_t = g_1(W_x x_t + W_s S_{t-1})$$
$$Y_t = g_2(W_Y S_t)$$

g1 and g2 are activation functions.

The error function be:

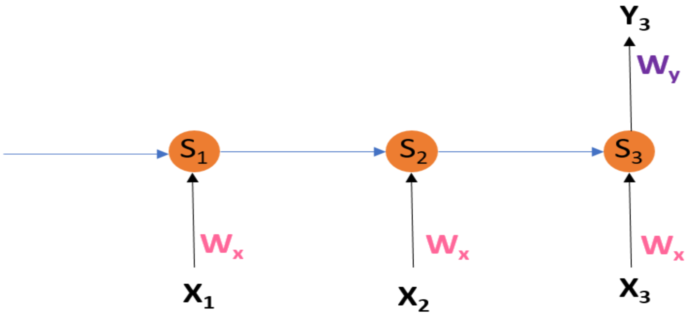
$$E_t = (d_t - Y_t)^2$$

Adjusting Wy



$$\frac{\partial E_N}{\partial W_S} = \sum_{i=1}^{N} \frac{\partial E_N}{\partial Y_N} . \frac{\partial Y_N}{\partial S_i} . \frac{\partial S_i}{\partial W_S}$$

Adjusting Wx:



$$\frac{\partial E_N}{\partial W_S} = \sum_{i=1}^{N} \frac{\partial E_N}{\partial Y_N} . \frac{\partial Y_N}{\partial S_i} . \frac{\partial S_i}{\partial W_X}$$

This method of Back Propagation through time (BPTT) can be used up to a limited number of time steps like 8 or 10. If we back propagate further, the gradient becomes too small. This problem is called the “Vanishing gradient” problem. The problem is that the contribution of information decays geometrically over time. So, if the number of time steps is >10 (Let’s say), that information will effectively be discarded.

BPTT is basically just a fancy buzz word for doing backpropagation on an unrolled RNN. Unrolling is a visualization and conceptual tool, which helps you understand what's going on within the network. Most of the time when implementing a recurrent neural network in the common programming frameworks, backpropagation is automatically taken care of, but you need to understand how it works to troubleshoot problems that may arise during the development process.

You can view a RNN as a sequence of neural networks that you train one after another with backpropagation.

The image below illustrates an unrolled RNN. On the left, the RNN is unrolled after the equal sign. Note there is no cycle after the equal sign since the different time steps are visualized and information is passed from one time step to the next. This illustration also shows why a RNN can be seen as a sequence of neural networks.

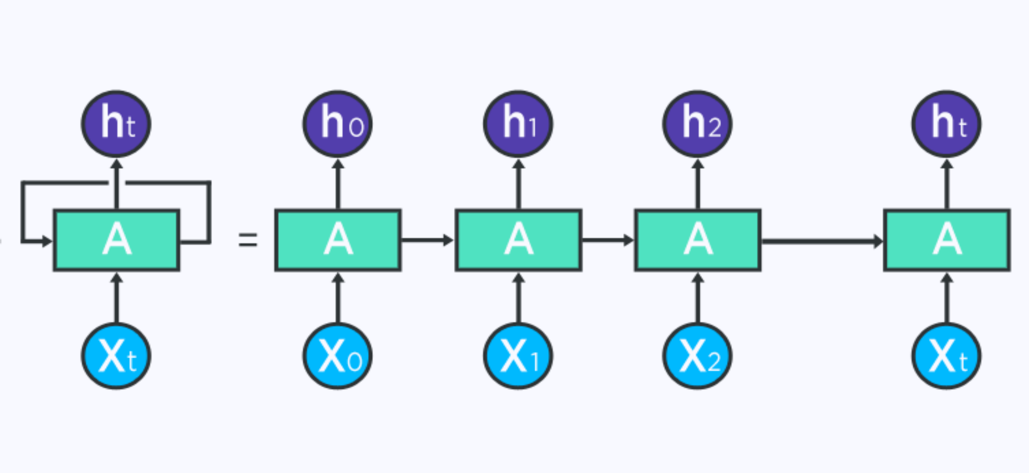


Figure 2.6 Unrolled RNN

If you do BPTT, the conceptualization of unrolling is required since the error of a given timestep depends on the previous time step.

Within BPTT the error is backpropagated from the last to the first timestep, while unrolling all the timesteps. This allows calculating the error for each timestep, which allows updating the weights. Note that BPTT can be computationally expensive when you have a high number of timesteps.

2.4 Issues with standard RNN’s

There are two major obstacles RNN’s have had to deal with, but to understand them, you first need to know what a gradient is.

A gradient is a partial derivative with respect to its inputs. If you don’t know what that means, just think of it like this: a gradient measure how much the output of a function changes if you change the inputs a little bit.

You can also think of a gradient as the slope of a function. The higher the gradient, the steeper the slope and the faster a model can learn. But if the slope is zero, the model stops learning. A gradient simply measures the change in all weights with regard to the change in error.

2.4.1 Vanishing Gradients

Vanishing Gradient occurs when the derivative or slope will get smaller and smaller as we go backward with every layer during backpropagation. When weights update is very small or exponential small, the training time takes too much longer, and in the worst case, this may completely stop the neural network training.

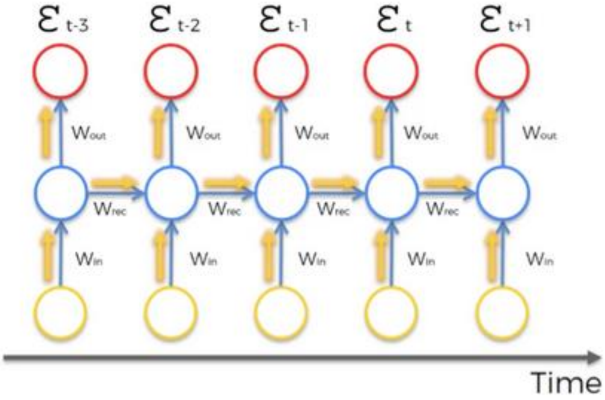


Figure 2.7 Vanishing Gradient Problem

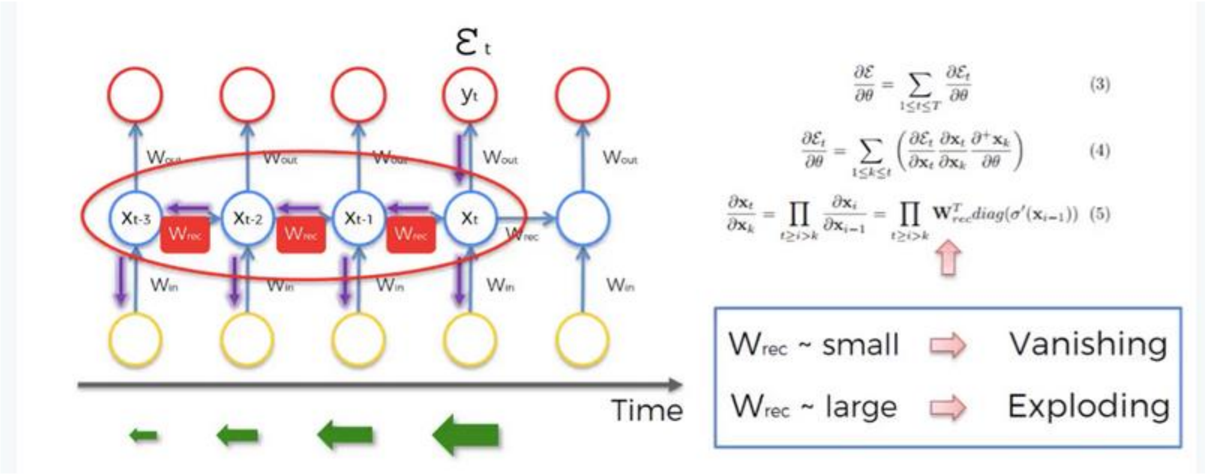
A vanishing Gradient problem occurs with the sigmoid and tanh activation function because the derivatives of the sigmoid and tanh activation functions are between 0 to 0.25 and 0–1. Therefore, the updated weight values are small, and the new weight values are very similar to the old weight values.

This leads to Vanishing Gradient problem.We can avoid this problem using the ReLU activation function because the gradient is 0 for negatives and zero input, and 1 for positive input.

2.4.2 Exploding gradients

Exploding gradient occurs when the derivatives or slope will get larger and larger as we go backward with every layer during backpropagation. This situation is the exact opposite of the vanishing gradients.

This problem happens because of weights, not because of the activation function. Due to high weight values, the derivatives will also higher so that the new weight varies a lot to the older weight, and the gradient will never converge.



$\mathcal{E}_t$
y_t
W_{out}
W_{out}
W_{out}
W_{out}
W_{out}
X_{t-3}
X_{t-2}
X_{t-1}
X_t
W_{rec}
W_{rec}
W_{rec}
W_{rec}
W_{in}
W_{in}
W_{in}
W_{in}
W_{in}
Time
$\frac{\partial \mathcal{E}}{\partial \theta} = \sum_{1 \le t \le T} \frac{\partial \mathcal{E}_t}{\partial \theta}$ (3)
$\frac{\partial \mathcal{E}_t}{\partial \theta} = \sum_{1 \le k \le t} \left(\frac{\partial \mathcal{E}_t}{\partial \mathbf{x}_t} \frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} \frac{\partial^+ \mathbf{x}_k}{\partial \theta} \right)$ (4)
$\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} = \prod_{t \ge i > k} \frac{\partial \mathbf{x}_i}{\partial \mathbf{x}_{i-1}} = \prod_{t \ge i > k} \mathbf{W}_{rec}^T diag(\sigma'(\mathbf{x}_{i-1}))$ (5)
W_{rec} ~ small ⇨ Vanishing
W_{rec} ~ large ⇨ Exploding

Figure 2.8 Exploding Gradients

So, it may result in oscillating around minima and never come to global minima point. Exploding gradients are when the algorithm, without much reason, assigns a stupidly high importance to the weights. Fortunately, this problem can be easily solved by truncating or squashing the gradients.

During the backpropagation in the deep neural networks, the Vanishing gradient problem occurs due to the sigmoid and tan activation function and the exploding gradient problem occurs due to large weights.

CHAPTER 3: LSTM

3.1 Introduction

Long Short Term Memory networks – usually just called "LSTMs"-are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hoch Reiter & Schmid Huber (1997), and were refined and popularized by many people in following work. They work tremendously well on a large variety of problems, and are now widely used.

LSTM networks were designed specifically to overcome the long-term dependency problem faced by recurrent neural networks RNNs (due to the vanishing gradient problem). LSTMs have feedback connections which make them different to more traditional feedforward neural networks. This property enables LSTMs to process entire sequences of data (e.g., time series) without treating each point in the sequence independently, but rather, retaining useful information about previous data in the sequence to help with the processing of new data points. As a result, LSTMs are particularly good at processing sequences of data such as text, speech and general time-series.

An example-Consider we are trying to predict monthly ice cream sales. As one might expect, these vary highly depending on the month of the year, being lowest in December and highest in June.

An LSTM network can learn this pattern that exists every 12 periods in time. It doesn't just use the previous prediction but rather retains a longer-term context which helps it overcome the long-term dependency problem faced by other models. It is worth noting that this is a very simplistic example, but when the pattern is separated by much longer periods of time (in long passages of text, for example), LSTMs become increasingly useful.

3.2 Architecture

Firstly, at a basic level, the output of an LSTM at a particular point in time is dependent on three things:

- ▹ The current long-term memory of the network-known as the cell state.
- ▹ The output at the previous point in time-known as the previous hidden state.
- ▹ The input data at the current time step.

LSTMs use a series of 'gates' which control how the information in a sequence of data comes into, is stored in and leaves the network. There are three gates in a typical LSTM; forget gate, input gate and output gate. These gates can be thought of as filters and are each their own neural network. We will explore them all in detail during the course of this article. In the following explanation, we consider an LSTM cell as visualised in the following diagram. When looking at the diagrams in this article, imagine moving from left to right.

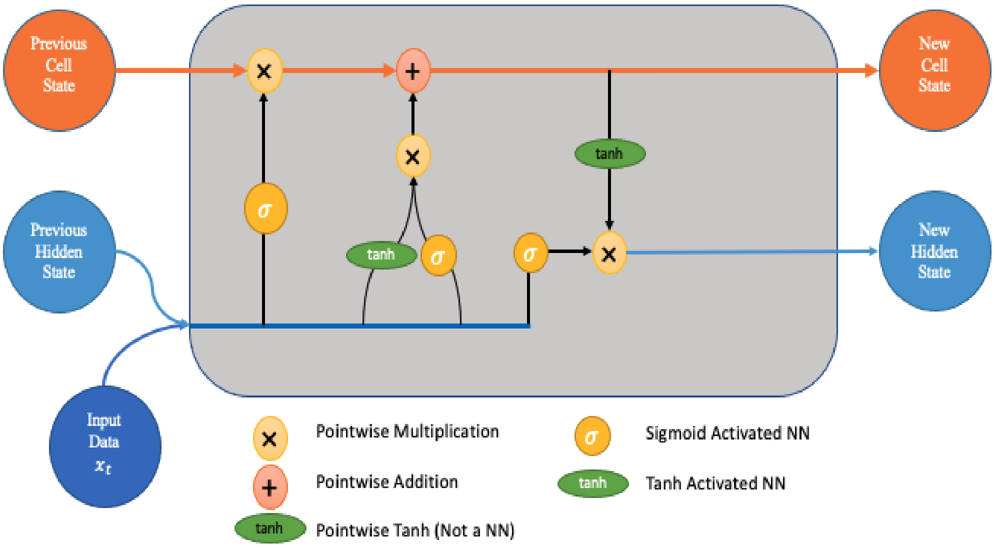


Figure 3.1 LSTM Diagram

Step 1:

The first step in the process is the **forget gate**. Here we will decide which bits of the cell state (long term memory of the network) are useful given both the previous hidden state and new input data.

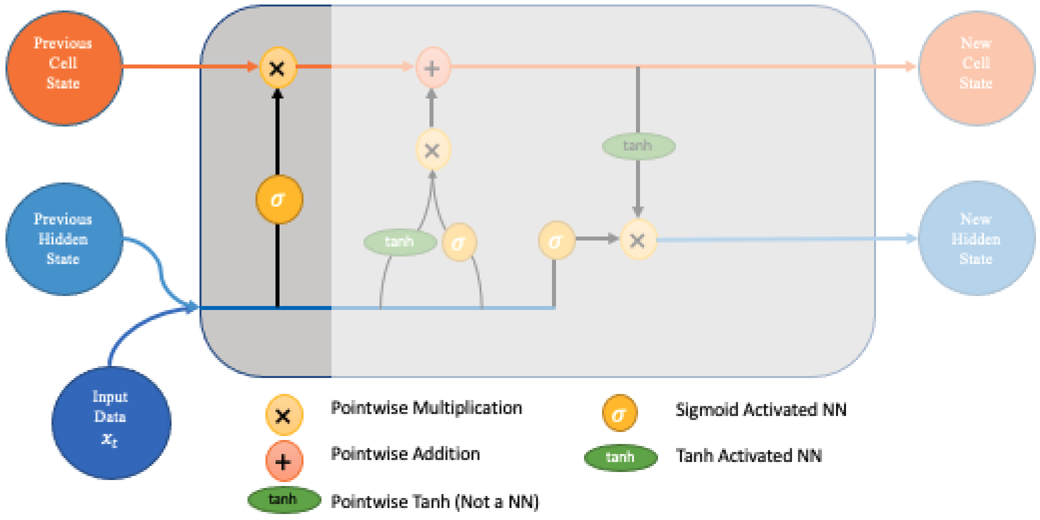


Figure 3.2 Forget Gate

To do this, the previous hidden state and the new input data are fed into a neural network. This network generates a vector where each element is in the interval [0,1] (ensured by using the sigmoid activation). This network (within the forget gate) is trained so that it outputs close to 0 when a component of the input is deemed irrelevant and closer to 1 when relevant. It is useful to think of each element of this vector as a sort of filter/sieve which allows more information through as the value gets closer to 1.

These outputted values are then sent up and pointwise multiplied with the previous cell state. This pointwise multiplication means that components of the cell state which have been deemed irrelevant by the forget gate network will be multiplied by a number close to 0 and thus will have less influence on the following steps. In summary, the forget gate decides which pieces of the long-term memory should now be forgotten (have less weight) given the previous hidden state and the new data point in the sequence.

Step 2:

The next step involves the **new memory network** and the **input gate**. The goal of this step is to determine what new information should be added to the networks long-term memory (cell state), given the previous hidden state and new input data.

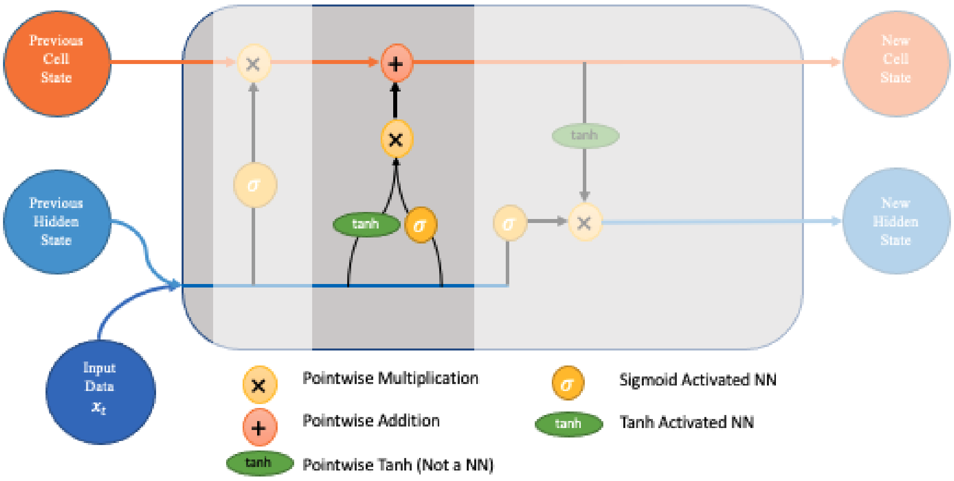


Figure 3.3 Input Gate

Both the new memory network and the input gate are neural networks in themselves, and both take the same inputs, the previous hidden state and the new input data. It is worth noting that the inputs here are actually the same as the inputs to the forget gate!

1) The **new memory network** is a tanh activated neural network which has learned how to combine the previous hidden state and new input data to generate a ‘new memory update vector’. This vector essentially contains information from the new input data given the context from the previous hidden state. This vector tells us how much to update each component of the long-term memory (cell state) of the network given the new data.

   Note that we use a tanh here because its values lie in [-1,1] and so can be negative. The possibility of negative values here is necessary if we wish to reduce the impact of a component in the cell state.

2) However, in part 1 above, where we generate the new memory vector, there is a big problem, it doesn't actually check if the new input data is even worth remembering. This is where the **input gate** comes in. The input gate is a sigmoid activated network which acts as a filter, identifying which components of the 'new memory vector' are worth retaining. This network will output a vector of values in [0,1] (due to the sigmoid activation), allowing it to act as a filter through pointwise multiplication. Similar to what we saw in the forget gate, an output near zero is telling us we don't want to update that element of the cell state.

3) The output of parts 1 and 2 are pointwise multiplied. This causes the magnitude of new information we decided on in part 2 to be regulated and set to 0 if need be. The resulting combined vector is then added to the cell state, resulting in the long-term memory of the network being updated.

Step 3:

Now that our updates to the long-term memory of the network are complete, we can move to the final step, **the output gate**, deciding the new hidden state. To decide this, we will use three things; the newly updated cell state, the previous hidden state and the new input data.

One might think that we could just output the updated cell state; however, this would be comparable to someone unloading everything they had ever learned about the stock market when only asked if they think it will go up or down tomorrow.

To prevent this from happening we create a filter, the output gate, exactly as we did in the forget gate network. The inputs are the same (previous hidden state and new data), and the activation is also sigmoid (since we want the filter property gained from outputs in [0,1]).

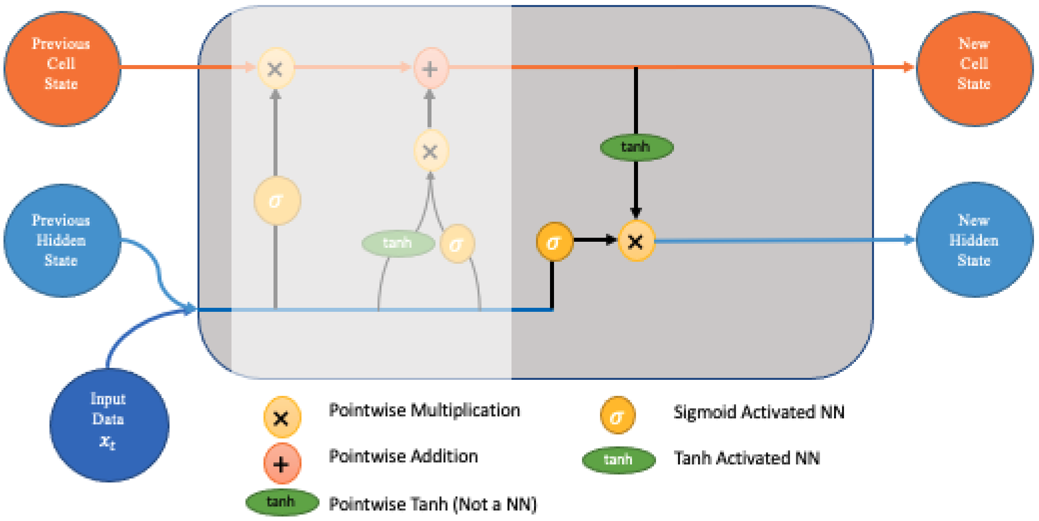


Figure 3.4 Output Gate

As mentioned, we want to apply this filter to the newly updated cell state. This ensures that only necessary information is output (saved to the new hidden state). However, before applying the filter, we pass the cell state through a tanh to force the values into the interval [-1,1].

The step-by-step process for this final step is as follows:

- Apply the tanh function to the current cell state pointwise to obtain the squished cell state, which now lies in [-1,1].
- Pass the previous hidden state and current input data through the sigmoid activated neural network to obtain the filter vector.
- Apply this filter vector to the squished cell state by pointwise multiplication.
- Output the new hidden state.

3.3 How the LSTM improves the RNN

The advantage of the Long Short-Term Memory (LSTM) network over other recurrent networks back in 1997 came from an improved method of back propagating the error. Hoch Reiter and Schmid Huber called it "constant error back propagation"

But what does it mean to be “constant”? We’ll go through the architecture of the LSTM and understand how it forward and back propagates to answer the question. We will make some comparisons to the Recurrent Neural Network (RNN) along the way. If you are not familiar with the RNN, you may want to read about it here. However, we should first understand what is the issue of the RNN that demanded for the solution the LSTM presents. That issue is the exploding and vanishing of the gradients that comes from the backward propagation step.

Vanishing and Exploding Gradients

Back propagation is the propagation of the error from its prediction up until the weights and biases. In recurrent networks like the RNN and the LSTM this term was also coined **Back Propagation Through Time** (BPTT) since it propagates through all time steps even though the weight and bias matrices are always the same.

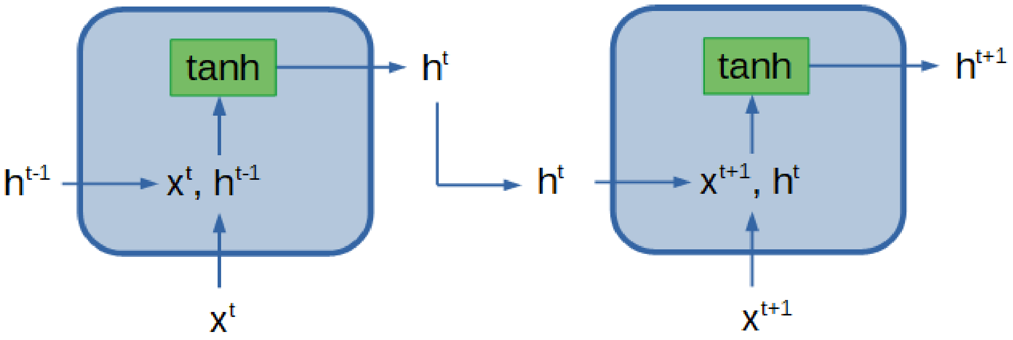


Figure 3.5 Portion of a typical RNN with two time steps

The figure above depicts a portion of a typical RNN with two inputs. The green rectangle represents the feed forward calculation of net inputs and their hidden state activations using a hyperbolic tangent (tanh) function.

The feed forward calculations use the same set of parameters (weight and bias) in all time steps.

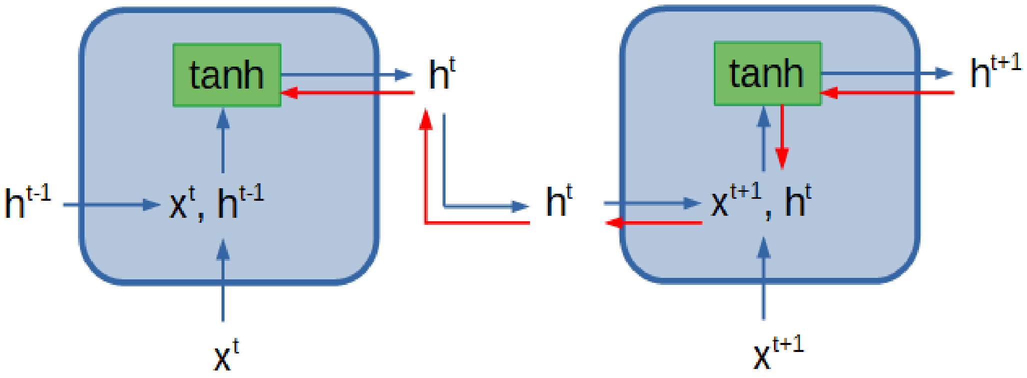


Figure 3.6 Forward propagation path (blue) and back propagation path (red) of a portion of a typical RNN

In red we see the BPTT path. For large sequences one can see that the calculations stack. This is important because it creates an exponential factor that depends greatly on the values of our weights. Everytime we go back a time step, we need to make an inner product between our current gradient and the weight matrix.

We can imagine our weight matrix to be a scalar and let's say that the absolute scalar is either around 0.9 or 1.1. Also, we have a sequence as big as 100 time steps. The exponential factor created by multiplying these values one hundred times would raise a **vanishing gradient** issue for 0.9:

$\mathbf{0.9^{100} = 0.000017(\ldots)}$

and an **exploding gradient** issue for 1.1: $\mathbf{1.1^{100} = 13780.61(\ldots)}$

Essencially, the BPTT calculation at the last time step would be similar to the following:

$$\frac{dL}{dW^1}=\frac{dL}{d\hat{y}}\frac{d\hat{y}}{dh^n}\frac{dh^n}{dh^{n-1}}(\cdots)\frac{dh^3}{dh^2}\frac{dh^2}{dh^1}\frac{dh^1}{dW^1}$$

$$\frac{dL}{dW^1}=\frac{dL}{d\hat{y}}(W_h)^T(W_n)^T(\cdots)(W_3)^T(W_2)^T\frac{dh^1}{dW^1}$$

Note that although the representation is not completely accurate, it gives a good idea of the exponential stacking of the weight matrices in the BPTT of an RNN with n inputs. W_h is the weight matrix of the last linear layer of the RNN.

$$W_{new}=W_{old}-\eta\frac{dL}{dW}$$

Next, we would be adding a portion of these values to the weight and bias matrices. You can see that we either barely improve the parameters, or try to improve so much that it backfires.

Now that we understand these concepts of vanishing and exploding gradients, we can move on to learn the LSTM. Let’s start by its forward pass.

LSTM Forward Propagation

Despite the differences that make the LSTM a more powerful network than RNN, there are still some similarities. It maintains the input and output configurations of one-to-one, many-to-one, one-to-many and many-to many. Also, one may choose to use a stacked configuration.

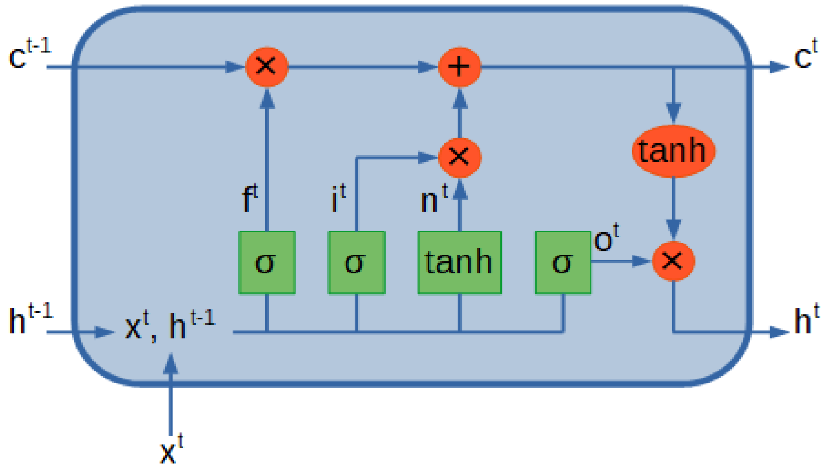


Figure 3.7 Representation of an LSTM Cell

Above we can see the forward propagation inside an LSTM cell. It is considerably more complicated than the simple RNN. It contains four networks activated by either the sigmoid function (σ) or the tanh function, all with their own different set of parameters. Each of these networks, also refered to as gates, have a different purpose. They will transform the cell state for time step t (*c^t*) with the relevant information that should be passed to the next time step. The orange circles/elipse are element-wise transformations of the matrices that preceed them. Here's what the gates do:

Forget gate layer (f): Decides which information to forget from the cell state using a σ function that modulates the information between 0 and 1. It forgets everything that is 0, remembers all that is 1 and everything in the middle are possible candidates.

Input gate layer (i): This could also be a remember gate. It decides which of the new candidates are relevant for this time step also with the help of a σ function.

New candidate gate layer (n): Creates a new set of candidates to be stored in the cell state. The relevancy of these new candidates will be modulated by the element-wise multiplication with the input gate layer.

Output gate layer (o): Determines which parts of the cell state are output. The cell state is normalized through a tanh function and is multiplied element-wise by the output gate that decides which relevant new candidate should be output by the hidden state.

$$f^t = \sigma\left(\left(X^t\right)^T \cdot W_f + B_f\right)$$
$$i^t = \sigma\left(\left(X^t\right)^T \cdot W_i + B_i\right)$$
$$n^t = \tanh\left(\left(X^t\right)^T \cdot W_n + B_n\right)$$
$$o^t = \sigma\left(\left(X^t\right)^T \cdot W_o + B_o\right)$$
$$c^t = \left(f^t \times c^{t-1}\right) + \left(i^t \times n^t\right)$$
$$h^t = o^t \times \tanh\left(c^t\right)$$
$$\hat{y} = h^t \cdot W_h + B_h$$
$$L = Loss\left(\hat{y}, y\right)$$

On the left you can see the calculations performed inside an LSTM cell. The last two calculations are an external feed forward layer to obtain a prediction and some loss function that takes the prediction and the true value.

The entire LSTM network’s architecture is built to deal with a three time step input sequence and to forecast a time step into the future like shown in the following figure:

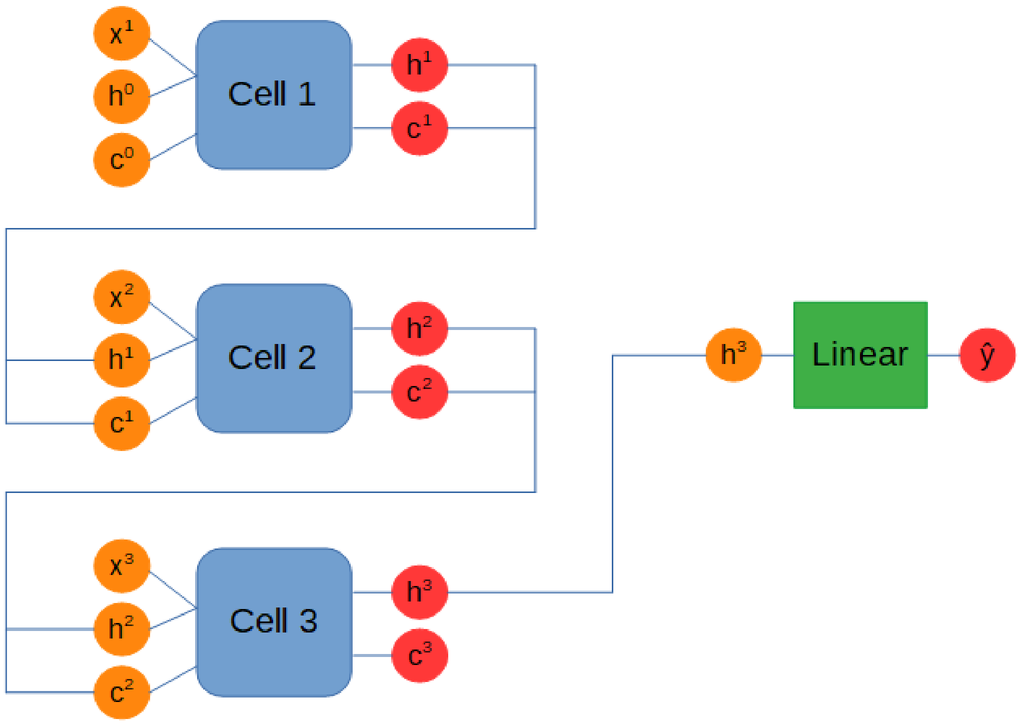


Figure 3.8 Representation of an LSTM with three inputs and one output

Putting the inputs and parameters into vector and matrix form may help understand the dimansionality of the calculations. Note that we are using four weight and bias matrices with their own values.

$$X^t = \begin{bmatrix} x^t \\ h_1^{t-1} \\ h_2^{t-1} \end{bmatrix} W_{(o,n,i,f)} = \begin{bmatrix} w_{11}^x & w_{12}^x \\ w_{11}^h & w_{12}^h \\ w_{21}^h & w_{22}^h \end{bmatrix} B_{(o,n,i,f)} = \begin{bmatrix} b_{11} & b_{12} \end{bmatrix}$$

This is the forward propagation of the LSTM. Now it is time to understand how the network back propagates and how it shines compared to the RNN.

LSTM Back Propagation

The improved learning of the LSTM allows the user to train models using sequences with several hundreds of time steps, something the RNN struggles to do.

Something that wasn’t mentioned when explaining the gates is that it is their job to decide the relevancy of information that is stored in the cell and hidden states so that, when back propagating from cell to cell, the passed error is as close to 1 as possible. This ensures that there is no vanishing or exploding of gradients.

Another simpler way of understanding the process is that the cell state connects the layers inside the cell with information that stabilizes the propagation of error somewhat like a ResNet does.

Let’s see how the error is kept constant by going through the back propagation calculations. We’ll start with the linear output layer.

$$\frac{dL}{d\hat{y}} = y - \hat{y}$$

$$\frac{dL}{dW_{\hat{y}}} = \frac{dL}{d\hat{y}}\frac{d\hat{y}}{dW_{\hat{y}}} = \left(\frac{dL}{d\hat{y}} \cdot (h^3)^T\right)^T$$

$$\frac{dL}{dB_{\hat{y}}} = \frac{dL}{d\hat{y}}\frac{d\hat{y}}{dB_{\hat{y}}} = \frac{dL}{d\hat{y}} \cdot 1$$

Now we’ll go about the LSTM cell’s back propagation. But first let’s get a visual of the path we must take within a cell.

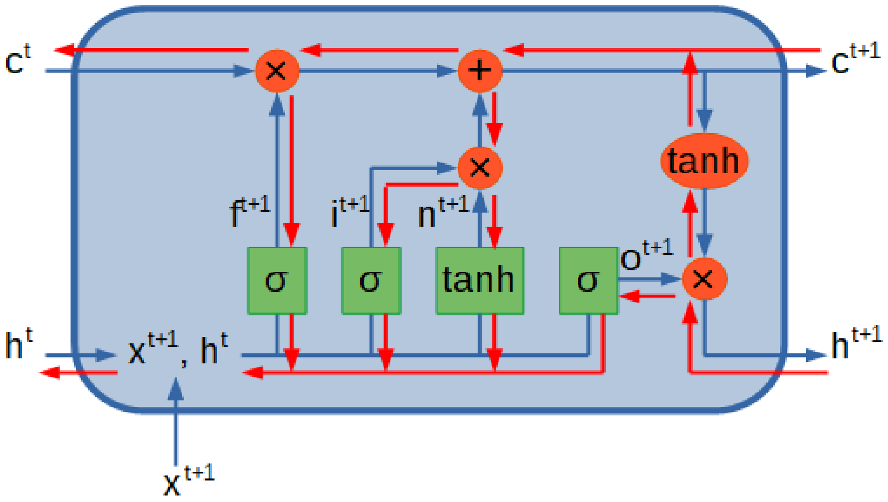


Figure 3.9 Full back propagation (red arrows) inside an LSTM cell

As you can see, the path is quite complicated, which makes for computationally heavier operations than the RNN.

Bellow you can see the back propagation of both outputs of an LSTM cell, the cell state and the hidden state. You can refer to the equations I showed above for the forward pass to get a better understanding of which equations we are going through.

$$\frac{dL}{dh^3}=\frac{dL}{d\hat{y}}\frac{d\hat{y}}{dh^3}=\left(\frac{dL}{d\hat{y}}\cdot(W_{\hat{y}})^T\right)^T$$

$$\frac{dL}{dc^3}=\frac{dL}{dh^3}\frac{dh^3}{dc^3}=\frac{dL}{dh^3}\times o^3\times \tanh'(c^3)$$

You can see that the information that travelled forward through the cell state is now going backwards modulated by the tanh’. Note that the prime (‘) in σ’ and tanh’ represent the first derivative of both these functions.

In the next steps, we are going back to the parameters in each gate.

Output gate:

$$\frac{dL}{do^3}=\frac{dL}{dh^3}\frac{dh^3}{do^3}=\frac{dL}{dh^3}\times\tanh(c^3)$$

$$\frac{dL}{dW_o}=\frac{dL}{do^3}\frac{do^3}{dW_o}=(\frac{dL}{do^3}\times\sigma'(o^3)\cdot(X^3)^T)^T$$

$$\frac{dL}{dB_o}=\frac{dL}{do^3}\frac{do^3}{dB_o}=\frac{dL}{do^3}\times\sigma'(o^3)$$

New canditate gate :

$$\frac{dL}{dn^3}=\frac{dL}{dc^3}\frac{dc^3}{dn^3}=\frac{dL}{dc^3}\times i^3$$

$$\frac{dL}{dW_n}=\frac{dL}{dc^3}\frac{dc^3}{dW_n}=(\frac{dL}{dc^3}\times\tanh'(n^3)\cdot(X^3)^T)^T$$

$$\frac{dL}{dB_n}=\frac{dL}{dc^3}\frac{dc^3}{dB_n}=\frac{dL}{dc^3}\times\tanh'(n^3)$$

Input gate:

$$\frac{dL}{di^3}=\frac{dL}{dc^3}\frac{dc^3}{di^3}=\frac{dL}{dc^3}\times n^2$$

$$\frac{dL}{dW_i}=\frac{dL}{dc^3}\frac{dc^3}{dW_i}=(\frac{dL}{dc^3}\times\tanh'(i^3)\cdot(X^3)^T)^T$$

$$\frac{dL}{dB_i}=\frac{dL}{dc^3}\frac{dc^3}{dB_i}=\frac{dL}{dc^3}\times\tanh'(i^3)$$

Forget gate:

$$\frac{dL}{df^3}=\frac{dL}{dc^3}\frac{dc^3}{df^3}=\frac{dL}{dc^3}\times c^2$$

$$\frac{dL}{dW_f}=\frac{dL}{dc^3}\frac{dc^3}{dW_f}=(\frac{dL}{dc^3}\times\tanh'(f^3)\cdot(X^3)^T)^T$$

$$\frac{dL}{dB_f}=\frac{dL}{dc^3}\frac{dc^3}{dB_f}=\frac{dL}{dc^3}\times\tanh'(f^3)$$

We have calculated the gradient for all the parameters inside the cell. However, we need to keep back propagating until the last cell. Let’s see the last steps:

$$\frac{dL}{dc^2}=\frac{dL}{dc^3}\frac{dc^3}{dc^2}=\frac{dL}{dc^3}\times f^3$$

$$\frac{dL}{dX^3}=\frac{dL}{do^3}\frac{do^3}{dX^3}+\frac{dL}{dn^3}\frac{dn^3}{dX^3}+\frac{dL}{di^3}\frac{di^3}{dX^3}+\frac{dL}{df^3}\frac{df^3}{dX^3}$$

$$\frac{dL}{dX^3}=((\frac{dL}{do^3})^T W_o^T)^T+((\frac{dL}{do^3})^T W_n^T)^T+((\frac{dL}{do^3})^T W_i^T)^T+((\frac{dL}{do^3})^T W_f^T)^T$$

You may see that the information that travels from cell state c³ to c² largelly depends on the outputs of the output gate and the forget gate. At the same time, the output and forget gradients depend on the information that was previously stored in the cell states. These interactions should provide the constant error back propagation.

Going further back into the global input (X³), we add what is coming from all four gates together.

$$\frac{dL}{dX^3}=\begin{bmatrix}\frac{dL}{dx^3}\\ \frac{dL}{dh_1^2}\\ \frac{dL}{dh_2^2}\end{bmatrix}\rightarrow\begin{matrix}\frac{dL}{dx^3}=\frac{dL}{dX^3}[1]\\ \frac{dL}{dh^2}=\frac{dL}{dX^3}[2,3]\end{matrix}$$

$$\frac{dL}{dW_x}=\frac{dL}{dW_x^1}+\frac{dL}{dW_x^2}+\frac{dL}{dW_x^3}$$

$$\frac{dL}{dB_x}=\frac{dL}{dB_x^1}+\frac{dL}{dB_x^2}+\frac{dL}{dB_x^3}$$

Finally, we deconcatenate the hidden state from the global input vector, go through the remaining cells and add all the gradients with respect to the parameters from all cells together.

CHAPTER 4: IMPLEMENTATION

4.1 Libraries

Open CV:

OpenCV is a huge open-source library for computer vision, machine learning, and image processing. OpenCV supports a wide variety of programming languages like Python, C++, Java, etc. It can process images and videos to identify objects, faces, or even the handwriting of a human. When it is integrated with various libraries, such as NumPy which is a highly optimized library for numerical operations, then the number of weapons increases in your Arsenal i.e., whatever operations one can do in NumPy can be combined with OpenCV.

NumPy:

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

OS Module:

The OS module in Python provides functions for interacting with the operating system. OS comes under Python’s standard utility modules. This module provides a portable way of using operating system-dependent functionality. The OS and os.path modules include many functions to interact with the file system.

Matplotlib:

Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

MediaPipe:

MediaPipe offers open source cross-platform, customizable ML solutions for live and streaming media. MediaPipe library provides detection solutions for hands, face, pose etc. We are using MediaPipe Holistic Solution for obtaining the Key Points.

TensorFlow:

TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

Google Trans:

Google trans is a free and unlimited python library that implemented Google Translate API. This uses the Google Translate Ajax API to make calls to such methods as detect and translate.

4.2 Pre-processing

4.2.1 Key points extraction using MP holistic:

- **MediaPipe holistic:**

The MediaPipe Holistic pipeline integrates separate models for pose, face and hand components, each of which are optimized for their particular domain. Mediapipe is a multi stage pipeline, which treats the different regions using a region appropriate image resolution.

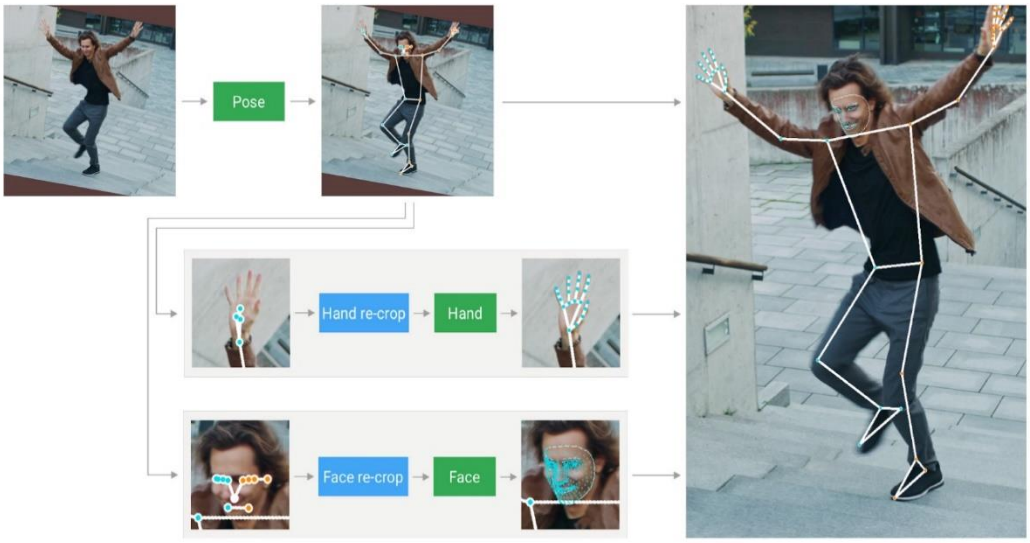


Figure 4.1 MediaPipe holistic

At first, the human pose with BlazePose's pose detector and subsequent landmark model. Then, using the inferred pose landmarks three regions of interest (ROI) crops for each hand and the face are derived, and a re-crop model is employed to improve the ROI.

The full resolution input frame is cropped into these ROIs and then task specific face and hand models are applied to estimate their corresponding landmarks. Finally, we merge all those landmarks with those of the pose model to yield the full 1662 landmarks.For the identification of ROIs for face and hands, it utilizes a tracking approach similar to the one we use for standalone face and hand pipelines.It assumes that the object doesn't move significantly between frames and uses estimation from the previous frame as a guide to the object region on the current one. However, during fast movements, the tracker can lose the target, which requires the detector to re-localize it in the image. MediaPipe Holistic uses pose prediction (on every frame) as an additional ROI prior to reduce the response time of the pipeline when reacting to fast movements. This also enables the model to retain semantic consistency across the body and its parts by preventing a mix up between left and right hands or body parts of one person in the frame with another.

In addition, the resolution of the input frame to the pose model is low enough that the resulting ROIs for face and hands are still too inaccurate to guide the re-cropping of those regions, which require a precise input crop to remain lightweight. To close this accuracy gap, we use lightweight face and hand re-crop models that play the role of spatial transformers and cost only ~10% of corresponding model's inference time.

Landmark Models

MediaPipe Holistic utilizes the pose, face and hand landmark models in MediaPipe Pose, MediaPipe Face Mesh and MediaPipe Hands respectively to generate a total of 543 landmarks (33 pose landmarks, 468 face landmarks, and 21 hand landmarks per hand).

Hand Recrop Model

For cases when the accuracy of the pose model is low enough that the resulting ROIs for hands are still too inaccurate, we run the additional lightweight hand re-crop model that play the role of spatial transformer and cost only ~10% of hand model inference time.

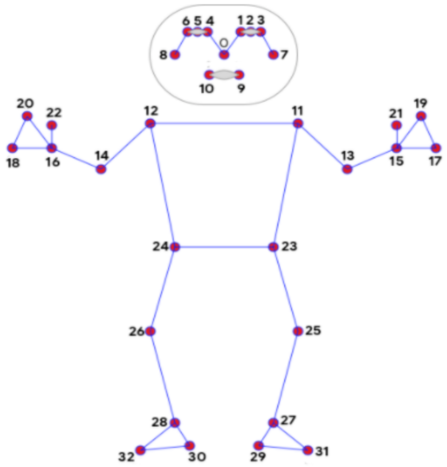
- **MediaPipe Pose:**

MediaPipe Pose is a ML solution for high-fidelity body pose tracking, inferring 33 3D landmarks and background segmentation mask on the whole body from RGB video frames utilizing our BlazePose research that also powers the ML Kit Pose Detection API.

pose_landmarks

A list of pose landmarks. Each landmark consists of the following:

- x and y: Landmark coordinates normalized to [0.0, 1.0] by the image width and height respectively.
- z: Represents the landmark depth with the depth at the midpoint of hips being the origin, and the smaller the value the closer the landmark is to the camera. The magnitude of z uses roughly the same scale as x.
- visibility: A value in [0.0, 1.0] indicating the likelihood of the landmark being visible (present and not occluded) in the image.



0. nose
1. left_eye_inner
2. left_eye
3. left_eye_outer
4. right_eye_inner
5. right_eye
6. right_eye_outer
7. left_ear
8. right_ear
9. mouth_left
10. mouth_right
11. left_shoulder
12. right_shoulder
13. left_elbow
14. right_elbow
15. left_wrist
16. right_wrist
17. left_pinky
18. right_pinky
19. left_index
20. right_index
21. left_thumb
22. right_thumb
23. left_hip
24. right_hip
25. left_knee
26. right_knee
27. left_ankle
28. right_ankle
29. left_heel
30. right_heel
31. left_foot_index
32. right_foot_index

Figure 4.2 Pose landmarks

- **MediaPipe Face Mesh:**

MediaPipe Face Mesh is a solution that estimates 468 3D face landmarks in real-time even on mobile devices. It employs machine learning (ML) to infer the 3D facial surface, requiring only a single camera input without the need for a dedicated depth sensor.

Utilizing lightweight model architectures together with GPU acceleration throughout the pipeline, the solution delivers real-time performance critical for live experiences.

Face Landmark Model

MediaPipe Face Mesh is a solution that estimates 468 3D face landmarks in real-time even on mobile devices. It employs machine learning (ML) to infer the 3D facial surface, requiring only a single camera input without the need for a dedicated depth sensor. Utilizing lightweight model architectures together with GPU acceleration throughout the pipeline, the solution delivers real-time performance critical for live experiences.

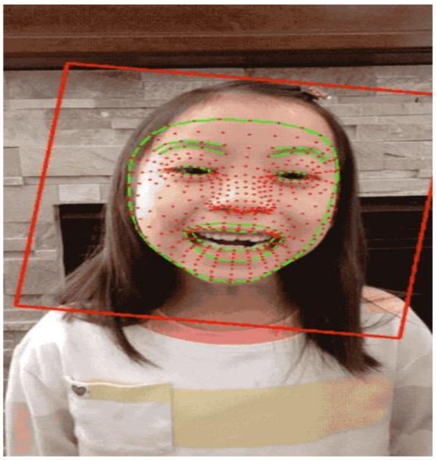


Figure 4.3 Face landmarks

face_landmarks

A list of 468 face landmarks. Each landmark consists of x, y and z. x and y are normalized to [0.0, 1.0] by the image width and height respectively. z represents the landmark depth with the depth at center of the head being the origin, and the smaller the value the closer the landmark is to the camera. The magnitude of z uses roughly the same scale as x.

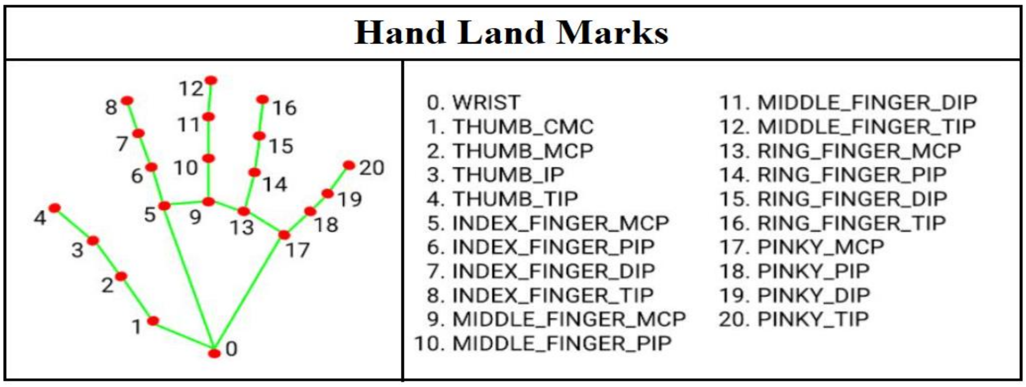
- **MediaPipe Hands:**

MediaPipe Hands is a high-fidelity hand and finger tracking solution. It employs machine learning (ML) to infer 21 3D landmarks of a hand from just a single frame. Whereas current state-of-the-art approaches rely primarily on powerful desktop environments for inference, our method achieves real-time performance on a mobile phone, and even scales to multiple hands.

Whereas current state-of-the-art approaches rely primarily on powerful desktop environments for inference, our method achieves real-time performance on a mobile phone, and even scales to multiple hands. Providing the accurately cropped hand image to the hand landmark model drastically reduces the need for data augmentation (e.g. rotations, translation and scale) and instead allows the network to dedicate most of its capacity towards coordinate prediction accuracy.

In, the pipeline the crops can also be generated based on the hand landmarks identified in the previous frame, and only when the landmark model could no longer identify hand presence is palm detection invoked to relocalize the hand.

The pipeline is implemented as a MediaPipe graph that uses a hand landmark tracking subgraph from the hand landmark module, and renders using a dedicated hand renderer subgraph. The hand landmark tracking subgraph internally uses a hand landmark subgraph from the same module and a palm detection subgraph from the palm detection module.



Hand Land Marks

0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Figure 4.4 Hand Land Marks

left_hand_landmarks

A list of 21 hand landmarks on the left hand. Each landmark consists of x, y and z. x and y are normalized to [0.0, 1.0] by the image width and height respectively. z represents the landmark depth with the depth at the wrist being the origin, and the smaller the value the closer the landmark is to the camera. The magnitude of z uses roughly the same scale as x.

right_hand_landmarks

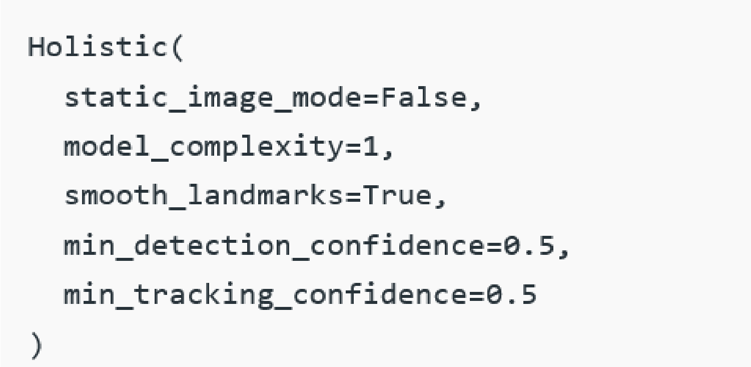
A list of 21 hand landmarks on the right hand, in the same representation as left_hand_landmarks.

4.2.2 Process for key points extraction using MediaPipe holistic:

Step-1:

Initializing Holistic model and drawing utils for detecting and drawing landmarks on the image. Using mediapipe holistic we extract 543 landmarks per frame in a video sequence.

Parameters for the holistic model:



```
Holistic(
  static_image_mode=False,
  model_complexity=1,
  smooth_landmarks=True,
  min_detection_confidence=0.5,
  min_tracking_confidence=0.5
)
```

- **static_image_mode:** It is used to specify whether the input images must be treated as static images or as a video stream. The default value is False.
- **model_complexity:** It is used to specify the complexity of the pose landmark model: 0, 1, or 2. As the model complexity of the model increases the landmark accuracy and latency increase. The default value is 1.
- **smooth_landmarks:** This parameter is used to reduce the jitter in the prediction by filtering pose landmarks across different input images. The default value is True.

- **min_detection_confidence:** It is used to specify the minimum confidence value with which the detection from the person-detection model needs to be considered as successful. Can specify a value in [0.0,1.0]. The default value is 0.5.
- **min_tracking_confidence:** It is used to specify the minimum confidence value with which the detection from the landmark-tracking model must be considered as successful. Can specify a value in [0.0,1.0]. The default value is 0.5.

Step-2:

Detecting Face and Hand landmarks from the image. Holistic model processes the image and produces landmarks for Face, Left Hand, Right Hand and also detects the Pose.

1.Capture the frames continuously from the camera using OpenCV.

2.Converting the BGR image to an RGB image and make predictions using initialized holistic model.

3.The predictions made by the holistic model are saved in the results variable from which we can access the landmarks using results.face_landmarks, results.right_hand_landmarks, results.left_hand_landmarks respectively.

4.The detected landmarks can be drawn on the image using the draw_landmarks function from drawing utils.

Pre-Processed Image:

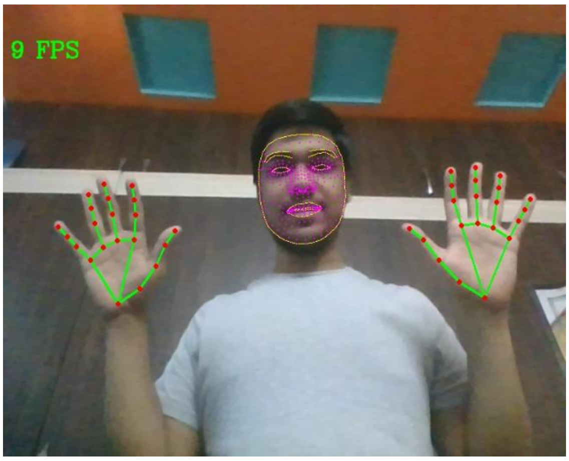
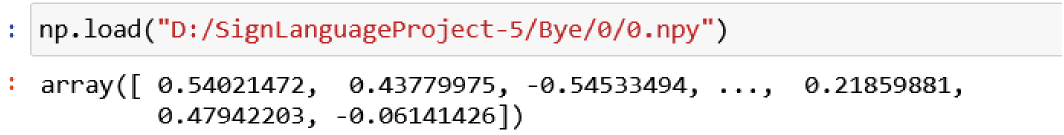


Figure 4.5 Pre-Processed Image

All landmarks are concatenated in the form an array for a frame which contains total 1662 Key Points values.

Pre-processing Stage output:



```
: np.load("D:/SignLanguageProject-5/Bye/0/0.npy")

: array([ 0.54021472,  0.43779975, -0.54533494, ...,  0.21859881,
          0.47942203, -0.06141426])
```

4.3 ISL to Text conversion stage

After preprocessing the frames of the video, the sign language gesture has to be converted to text by training and testing/classifying using Neural Networks.

For this, we are using LSTM (Long Short Term Memory) model which is a type of RNN for the detection of hand gestures and converting it into text as it is able to learn long term dependencies.

We recorded 30 video sequences for each sign language gesture and data from all sequences is combined into an array and given as input to LSTM neural network model

The input to the LSTM neural network is of the shape (30,1662) that is each video sequence containing 30 frames and each frame containing 1662 landmarks values.

The obtained data from the recorded video gesture sequences is split into train and test data and using this data, the LSTM model is trained.

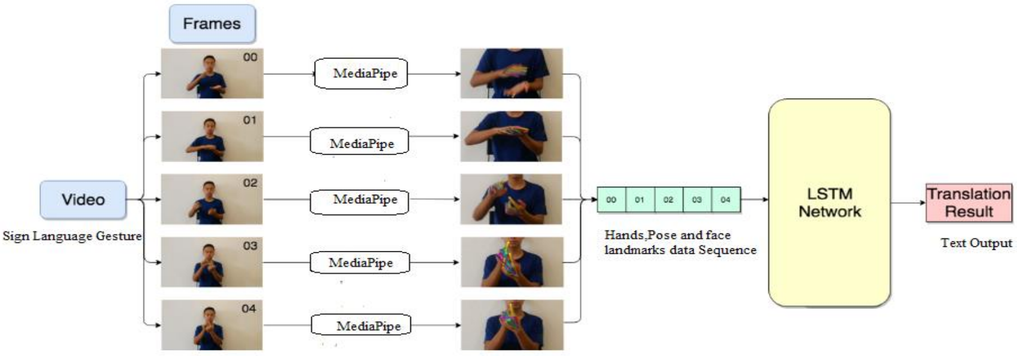
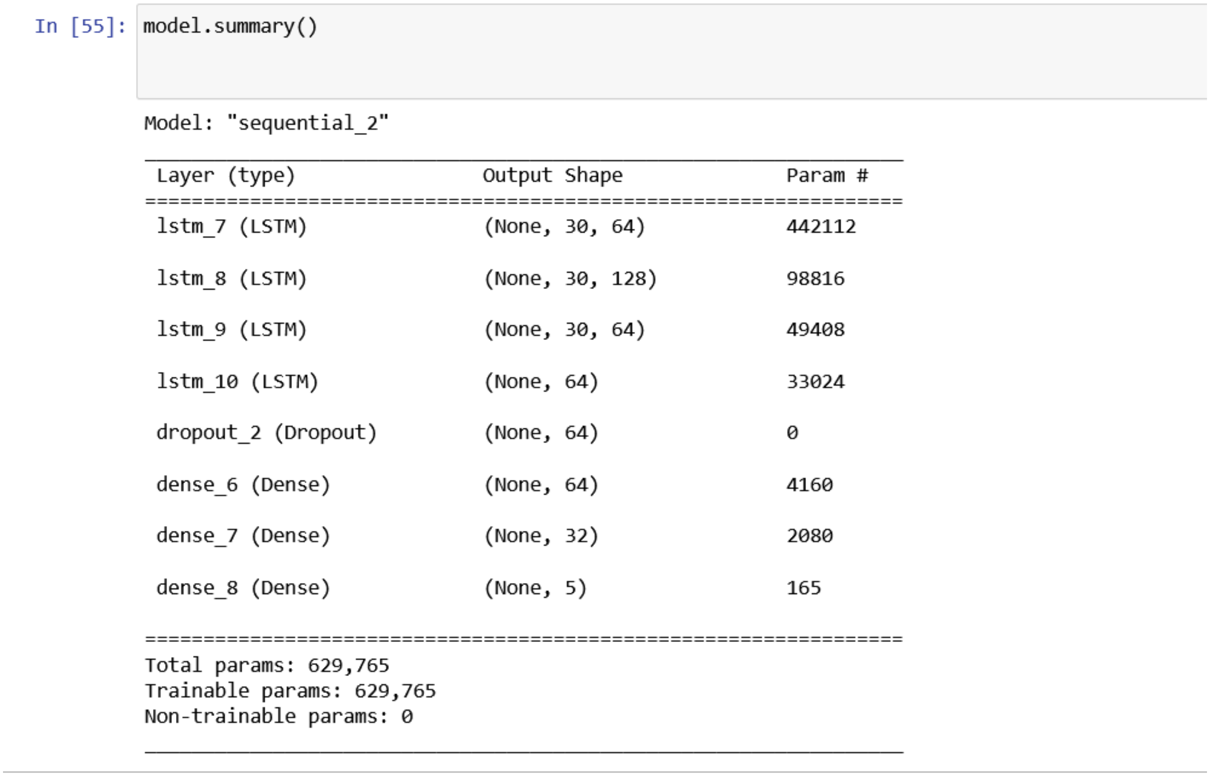


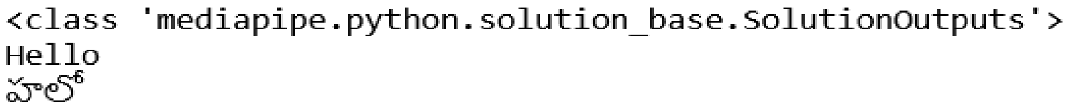
Figure 4.6 ISL to Text system overview

Model Summary:

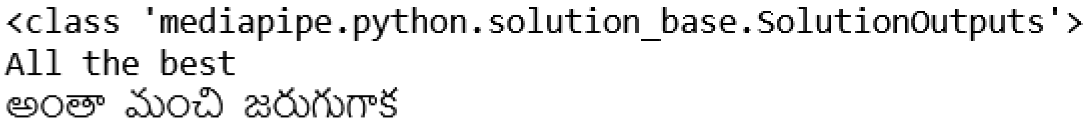


```
In [55]: model.summary()

Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_7 (LSTM)               (None, 30, 64)            442112

 lstm_8 (LSTM)               (None, 30, 128)           98816

 lstm_9 (LSTM)               (None, 30, 64)            49408

 lstm_10 (LSTM)              (None, 64)                33024

 dropout_2 (Dropout)         (None, 64)                0

 dense_6 (Dense)             (None, 64)                4160

 dense_7 (Dense)             (None, 32)                2080

 dense_8 (Dense)             (None, 5)                 165

=================================================================
Total params: 629,765
Trainable params: 629,765
Non-trainable params: 0
_________________________________________________________________
```

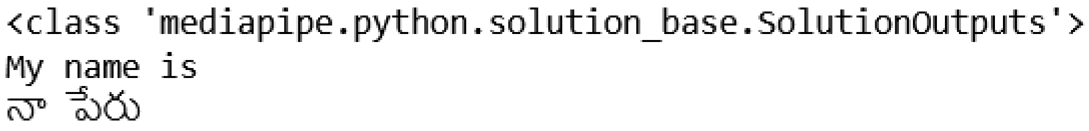
ISL to Text conversion stage outputs:



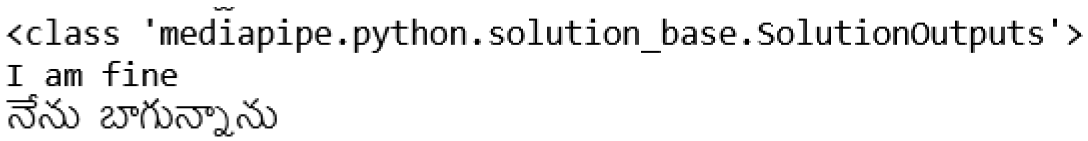
```
<class 'mediapipe.python.solution_base.SolutionOutputs'>
Hello
హాలో
```



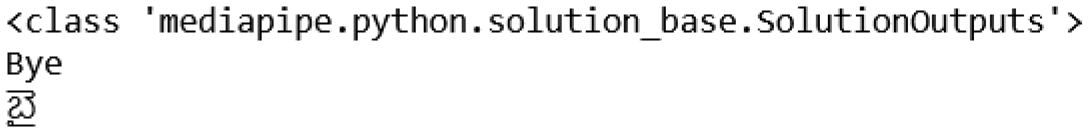
```
<class 'mediapipe.python.solution_base.SolutionOutputs'>
All the best
అంతా మంచి జరుగుగాక
```



```
<class 'mediapipe.python.solution_base.SolutionOutputs'>
My name is
నా పేరు
```



```
<class 'mediapipe.python.solution_base.SolutionOutputs'>
I am fine
నేను బాగున్నాను
```



```
<class 'mediapipe.python.solution_base.SolutionOutputs'>
Bye
బై
```

4.4 Text to speech conversion stage:

The input text should be pre-processed and normalized, followed by linguistic and prosodic processing, which is the translation of words to segments with durations and an F0 contour. Then generating the waveform and the output will be in speech format.

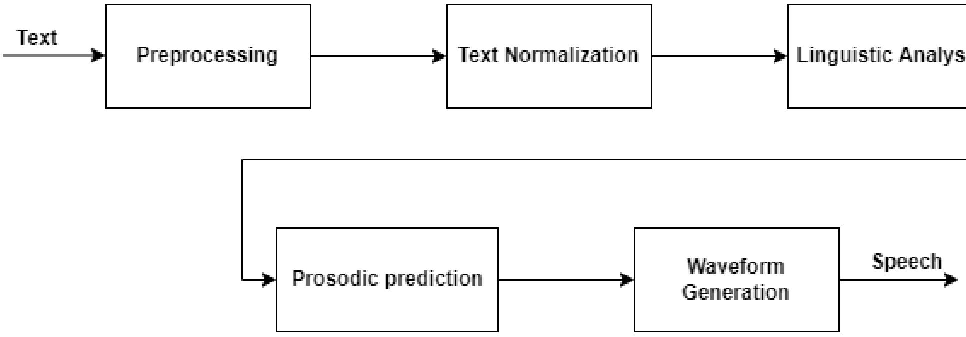


Figure 4.7 Text to Speech conversion

Text Analysis and Processing involves token identification, normalization of non-standard words and Homograph disambiguation.

Linguistic Analysis is the process of understanding the content of the text. Statistical methods are used to find the most probable meaning of the text.

Prosody means, patterns of stress and intonation in a language. Prosodic prediction focuses on how to predict the prosodic information from linguistic text and then how to exploit the predicted prosodic knowledge for various speech applications.

Festival Speech Synthesis System

- There are different tools for Text-to-Speech Synthesis. The Festival Speech Synthesis is one of an open-source Text-to-Speech tools.
- Here, this Festival Speech Synthesis System is used to build natural sounding like human voice.
- Building of New synthetic voices using festival speech synthesis system tool requires the sources of festival, festvox, speech tools and SPTK.

- The festival is a speech synthesis system and it is developed in CSTR (Center for Speech Technology Research), university of Edinburgh. It is multi-lingual.
- Festival offers a general framework for building speech synthesis systems as well as including examples of various modules. As a whole it offers full text to speech through a number APIs: from shell level, though a Scheme command interpreter, as a C++ library, from Java, and an Emacs interface.

Festival Architecture

- A festival speech synthesis system consists of different modules and they all together produce the synthetic speech. The modules present in festival are:

  ➢ Text Analysis and processing

  • Token identification

  • Normalization of non-standard words

  • Homograph disambiguation and Chunking into utterance

  ➢ Linguistic/prosody processing

  ➢ Wave form generation

- To generate natural sounding like human voice model 4hrs 10 minutes of speech was recorded at IISU Thiruvananthapuram. Voice of Ms. Sangeetha, ISRO inertial Systems Unit (IISU) Scientist is recorded for training and building the TTS engine.
- The given speech data is segmented into .wav files (1235 files) with length of 5sec-15sec and prepared corresponding text Prompt file for training the database.
- Each .wav file consists of 16 kHz sampling frequency and quantized with signed 16bits/sample with mono stream.
- The Quality of Synthesized speech is measured using Mel-Cepstral Distrotion (MCD). MCD objective measure is used for testing the generated TTS voice model and the values are 5.31 for around 3.5hrs data and 5.24 for around 4hrs of data.

Text to Speech conversion stage outputs:

hello.wav allthebest.wav bye.wav MYNAMEIS.wav iamfine.wav

4.5 Codes

1. Import and Install Dependencies

```
import cv2
import numpy as np
import os
from matplotlib import pyplot as plt
import time
import mediapipe as mp
```

2. Key points using MP holistic

```
mp_holistic = mp.solutions.holistic# Holistic model
mp_drawing = mp.solutions.drawing_utils # Drawing utilities

def mediapipe_detection(image, model):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # COLOR CONVERSION BGR 2RGB
    image.flags.writeable = False # Image is no longer writeable
    results = model.process(image)# Make prediction,this is actually detection
    image.flags.writeable = True  # Image is now writeable
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR) # COLOR COVERSION RGB 2 BGR
    return image, results


def draw_styled_landmarks(image, results):
    # Draw face connections
    mp_drawing.draw_landmarks(image, results.face_landmarks, 
mp_holistic.FACEMESH_TESSELATION, 
                             mp_drawing.DrawingSpec(color=(80,110,10), 
thickness=1, circle_radius=1), 
                             mp_drawing.DrawingSpec(color=(80,256,121), 
thickness=1, circle_radius=1)
                             )
    # Draw pose connections
    mp_drawing.draw_landmarks(image, results.pose_landmarks, 
mp_holistic.POSE_CONNECTIONS,
                             mp_drawing.DrawingSpec(color=(80,22,10), 
thickness=2, circle_radius=4), 
                             mp_drawing.DrawingSpec(color=(80,44,121), 
thickness=2, circle_radius=2)
                             ) 

    # Draw left hand connections
    mp_drawing.draw_landmarks(image, results.left_hand_landmarks, 
mp_holistic.HAND_CONNECTIONS, 
                             mp_drawing.DrawingSpec(color=(121,22,76), 
thickness=2, circle_radius=4), 
                             mp_drawing.DrawingSpec(color=(121,44,250), 
thickness=2, circle_radius=2)
                             ) 
    # Draw right hand connections  
    mp_drawing.draw_landmarks(image, results.right_hand_landmarks, 
mp_holistic.HAND_CONNECTIONS, 
                             mp_drawing.DrawingSpec(color=(245,117,66), 
thickness=2, circle_radius=4), 
                             mp_drawing.DrawingSpec(color=(245,66,230), 
thickness=2, circle_radius=2)
                             ) 
```

3. Extract Key point Values into a format so that we can use them

```
def extract_keypoints(results):
    pose = np.array([[res.x, res.y, res.z, res.visibility] for res in 
results.pose_landmarks.landmark]).flatten() if results.pose_landmarks else 
np.zeros(33*4)
    face = np.array([[res.x, res.y, res.z] for res in 
results.face_landmarks.landmark]).flatten() if results.face_landmarks else 
np.zeros(468*3)
    lh = np.array([[res.x, res.y, res.z] for res in 
results.left_hand_landmarks.landmark]).flatten() if 
results.left_hand_landmarks else np.zeros(21*3)
    rh = np.array([[res.x, res.y, res.z] for res in 
results.right_hand_landmarks.landmark]).flatten() if 
results.right_hand_landmarks else np.zeros(21*3)
    return np.concatenate([pose,face,lh, rh])
```

4. Setup folders for collection

```
# Path for exported data, numpy arrays
DATA_PATH = "D:/SignLanguageProject-5"

# Actions that we try to detect
actions = np.array(['All the best','Bye','Hello','I am fine','My name is'])

# Thirty videos worth of data
no_sequences = 30


# Videos are going to be 30 frames in length
sequence_length = 30

# Folder start
start_folder = 30

for action in actions:
    for sequence in range(no_sequences):
        try:
            os.makedirs(os.path.join(DATA_PATH, action, str(sequence)))
        except:
            pass
```

5. Collect Key point Values for Training and Testing

```
cap = cv2.VideoCapture(0)
# Set mediapipe model
with mp_holistic.Holistic(min_detection_confidence=0.05,
min_tracking_confidence=0.05) as holistic:

    # NEW LOOP
    # Loop through actions
    for action in actions:
        # Loop through sequences aka videos
        for sequence in range(no_sequences):
            # Loop through video length aka sequence length
            for frame_num in range(sequence_length):

                # Read feed
                ret, frame = cap.read()

                # Make detections
                image, results = mediapipe_detection(frame, holistic)
                # Draw landmarks
                draw_styled_landmarks(image, results)

                # NEW Apply collection logic
                if frame_num == 0:
                    cv2.putText(image, 'STARTING COLLECTION', (120,200),
                               cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255, 0), 4,
cv2.LINE_AA)
                    cv2.putText(image, 'Collecting frames for {} Video Number
{}'.format(action, sequence), (15,12),
                               cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1,
cv2.LINE_AA)



 # Show to screen
                    cv2.imshow('OpenCV Feed', image)
                    cv2.waitKey(2000)
                else:
                    cv2.putText(image, 'Collecting frames for {} Video Number
{}'.format(action, sequence), (15,12),
                               cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1,
cv2.LINE_AA)

                    # Show to screen
                    cv2.imshow('OpenCV Feed', image)

                # NEW Export keypoints
                keypoints = extract_keypoints(results)
                npy_path = os.path.join(DATA_PATH, action, str(sequence),
str(frame_num))
                np.save(npy_path, keypoints)

                # Break gracefully
                if cv2.waitKey(10) & 0xFF == ord('q'):
                    break

    cap.release()
    cv2.destroyAllWindows()
```

6. Preprocess data and Create Labels and Features

```
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical


label_map = {label:num for num, label in enumerate(actions)}
```

```
rootpath =  "D:/DSP/TestingData-Final-5"
x = os.listdir(rootpath)
label_map = {label:num for num,label in enumerate(x)}
gestures_list = [os.path.splitext(x)[0] for x in os.listdir(rootpath)]
#print(gestures_list)
gestures_array = np.array(gestures_list)
print(label_map)
sequences,labels = [],[]
for gesture in x:
    videos = os.listdir(os.path.join(rootpath,gesture))
    for video in videos:
        window = []
        frames = os.listdir(os.path.join(rootpath,gesture,video))



    for frame in frames:
            res = np.load(os.path.join(rootpath,gesture,video,frame))
            window.append(res)
    sequences.append(window)
    labels.append(label_map[gesture])


X = np.array(sequences)
y = to_categorical(labels).astype(int)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

7. Build and Train LSTM Neural Network

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense,Dropout
from tensorflow.keras.callbacks import TensorBoard
import tensorflow.keras.optimizers as f


log_dir = os.path.join('Logs')
tb_callback = TensorBoard(log_dir=log_dir)


model = Sequential()
model.add(LSTM(64, return_sequences=True, activation='relu',
input_shape=(201,1662)))
model.add(LSTM(128, return_sequences=True, activation='relu'))
model.add(LSTM(64,return_sequences = True,activation = 'relu'))
model.add(LSTM(64,return_sequences = True,activation = 'relu'))
model.add(LSTM(64, return_sequences=False, activation='relu'))
model.add(Dropout(.2))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(gestures_array.shape[0], activation='softmax'))


opt = f.Adam(learning_rate=0.00001)
model.compile(optimizer=opt, loss='categorical_crossentropy',
metrics=['categorical_accuracy'])
```

8. Make Predictions

```
res = model.predict(X_test)
gestures_list[np.argmax(res[4])]
gestures_list[np.argmax(y_test[4])]
```

9. Save Weights

```
model.save('train_model_unequallength_5_2_words.h5')
del model
model.load_weights('action.h5')
```

10. Evaluation using Confusion Matrix and Accuracy

```
from sklearn.metrics import confusion_matrix,
accuracy_score,ConfusionMatrixDisplay
from tensorflow.keras.models import load_model

model_test = load_model('train_model_unequallength_5_2_words.h5')
yhat = model_test.predict(X_test)
ytrue = np.argmax(y_test, axis=1).tolist()
yhat = np.argmax(yhat, axis=1).tolist()

labels_ex = ['All the best','Bye','Hello','I am fine','My name is']
cm = confusion_matrix(ytrue,yhat)
cmd = ConfusionMatrixDisplay(cm)
cmd.plot()

accuracy_score(ytrue, yhat)
```

11. Test in real time

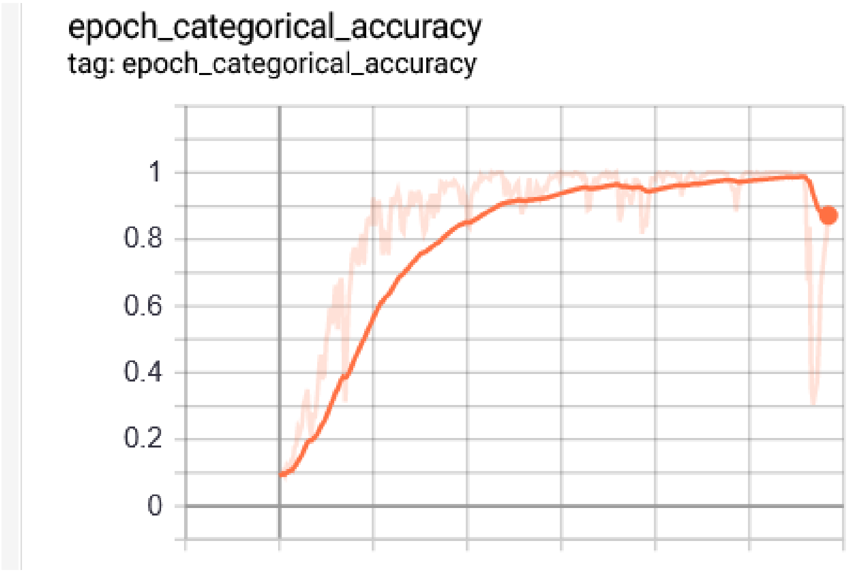
```
from scipy import stats

# 1. New detection variables
from googletrans import Translator
translator = Translator()
from PIL import ImageFont,ImageDraw,Image
sequence = []
#sentence = []
threshold = 0.4

img = np.zeros((200,400,3),np.uint8)
b,g,r,a = 0,255,0,0

cap = cv2.VideoCapture(0)
# Set mediapipe model
with mp_holistic.Holistic(min_detection_confidence=0.05,
min_tracking_confidence=0.05) as holistic:
    while cap.isOpened():
        # Read feed
        ret, frame = cap.read()

        # Make detections
        image, results = mediapipe_detection(frame, holistic)
        print(results)

        # Draw landmarks
        draw_styled_landmarks(image, results)

        # 2. Prediction logic
        keypoints = extract_keypoints(results)
#         sequence.insert(0,keypoints)
#          sequence = sequence[:30]
        sequence.append(keypoints)
        sequence = sequence[-30:]
          if len(sequence) == 30:
            res = model_test.predict(np.expand_dims(sequence, axis=0))[0]
            print(actions[np.argmax(res)])
            #3. Viz logic
            if res[np.argmax(res)] > threshold:
                #if len(sentence) > 0:
                    #if actions[np.argmax(res)] != sentence[-1]:
                        #sentence.append(actions[np.argmax(res)])
                    #else:
                    #sentence.append(actions[np.argmax(res)])
                #if len(sentence) > 5:
                #sentence = sentence[-5:]

            # Viz probabilities
                #image = prob_viz(res, actions, image, colors)
                translation = 
translator.translate(actions[np.argmax(res)],dest='te',src='en')
                print(translation.text)
                cv2.rectangle(image, (0,0), (640, 40), (245, 117, 16), -1)
                cv2.putText(image,actions[np.argmax(res)], 
(3,30),cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)

        # Show to screen
                cv2.imshow('OpenCV Feed', image)

        # Break gracefully
        if cv2.waitKey(10) & 0xFF == ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows()
```

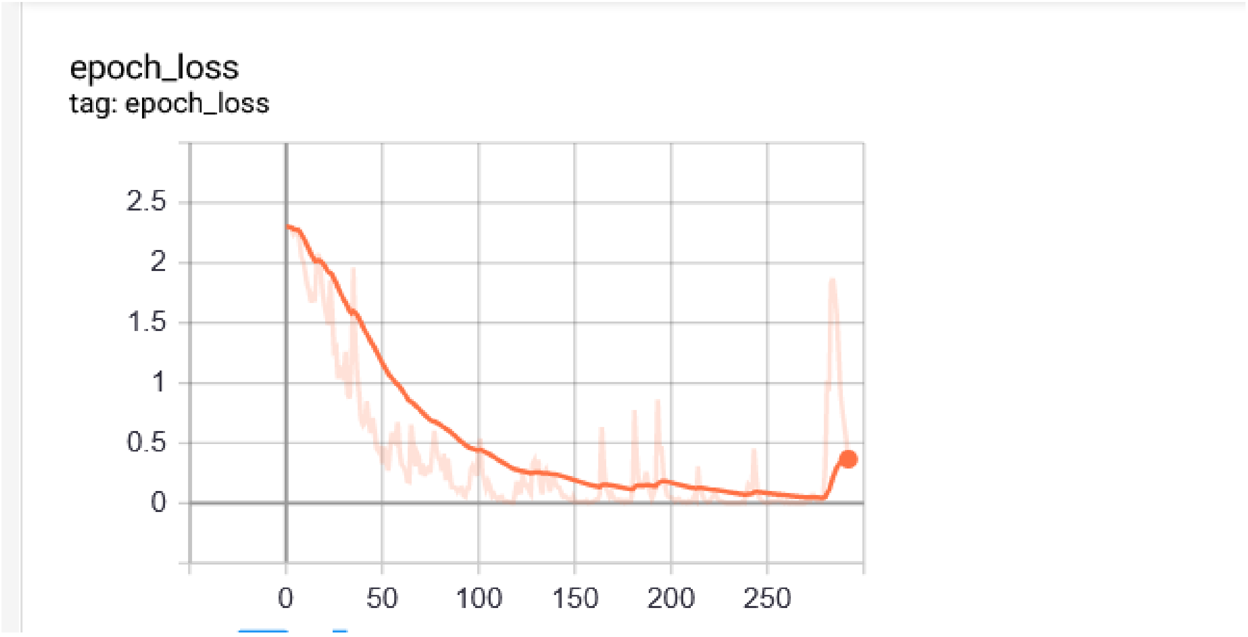
CHAPTER 5: RESULTS

Approach 1:

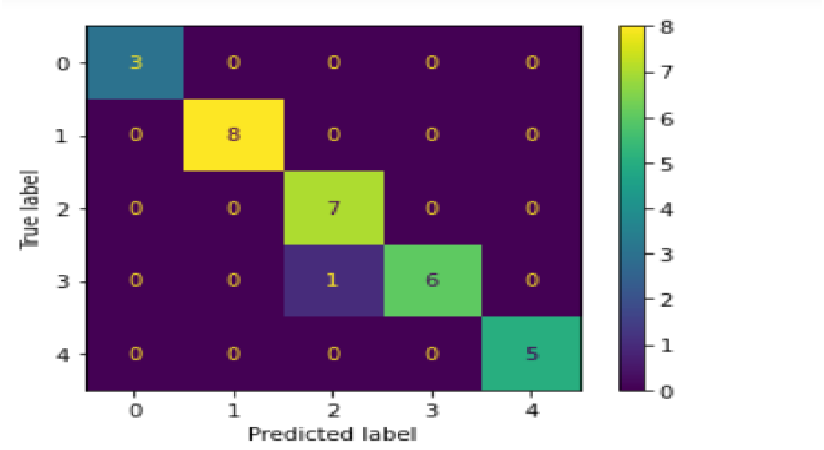
Data set containing video sequences of equal length, that is containing equal no. of frames.

Accuracy and Loss graphs:





Confusion Matrix:



0 – All the best

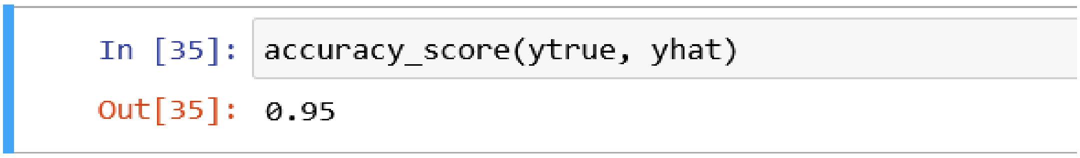
1 – Bye

2 – Hello

3 – I am fine

4 – My name is

Validation Accuracy:

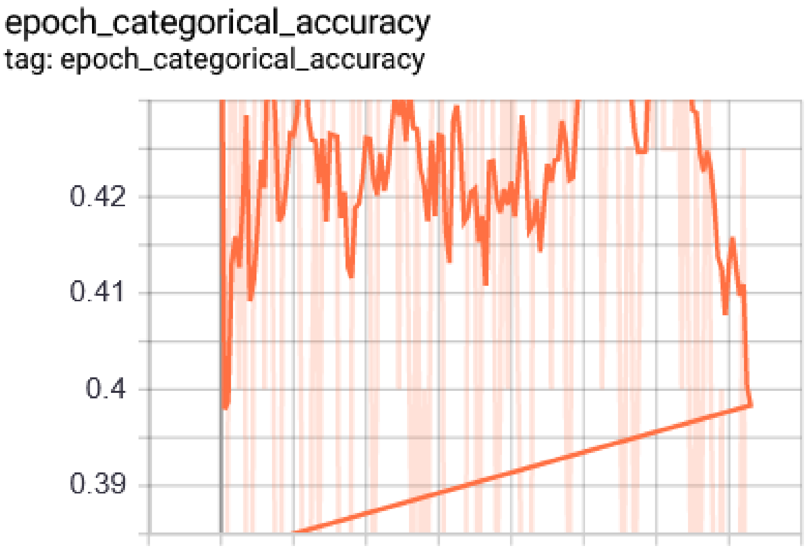


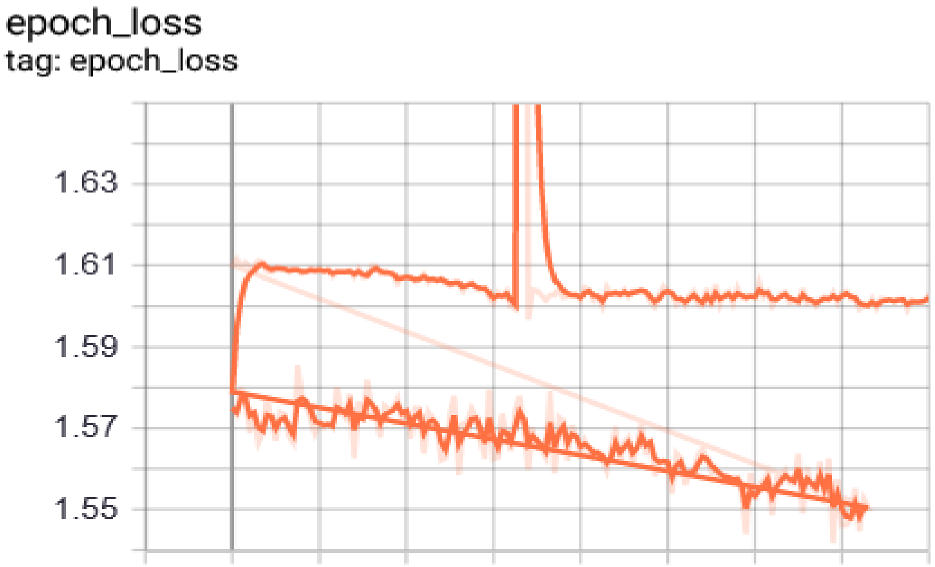
```
In [35]: accuracy_score(ytrue, yhat)
Out[35]: 0.95
```

Approach 2:

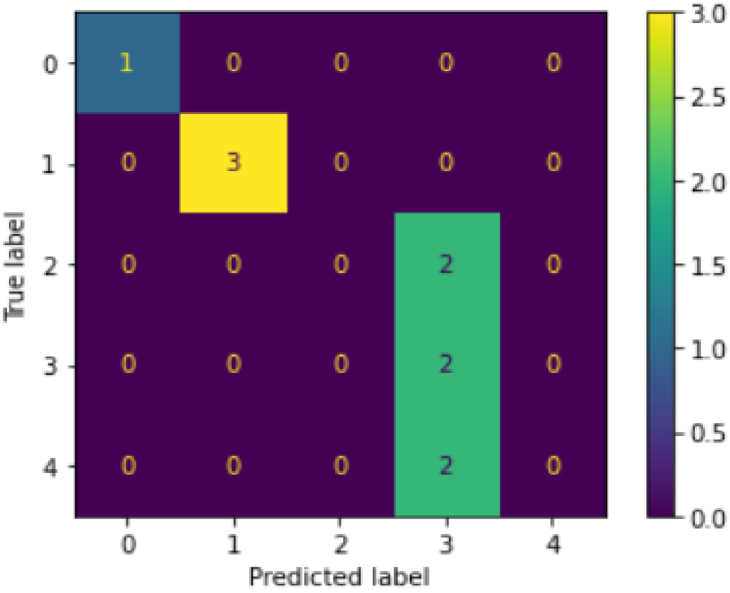
Data Set containing video sequences of unequal length, that is containing unequal no. of frames.

Accuracy and Loss graphs:



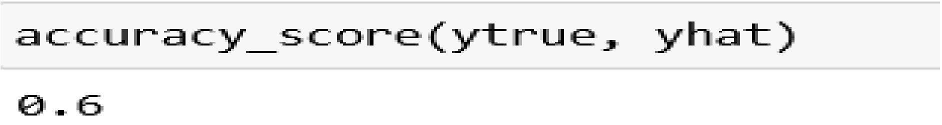


Confusion Matrix:



True label \ Predicted label	0	1	2	3	4
0	1	0	0	0	0
1	0	3	0	0	0
2	0	0	0	2	0
3	0	0	0	2	0
4	0	0	0	2	0

Validation Accuracy:



```
accuracy_score(ytrue, yhat)
```

```
0.6
```

5.1 Conclusion

We used LSTM to create an automated Indian Sign Language to Telugu voice conversion system in this project. To train our model, we employed two distinct datasets. Using the dataset containing unequal-length video sequences, we attained a training accuracy of 45 percent and a validation accuracy of 60 percent. Using the dataset with equal-length video sequences, we achieved training accuracy of 98 percent and validation accuracy of 95 percent. So, based on the facts above, we may conclude that LSTM works best with inputs of comparable length. Though the accuracy gained by training LSTM is low in the initial method, when CNN is merged with LSTM, we may reach a very excellent result. The second strategy, while providing great accuracy, lacks scalability.

5.2 References

1) S. Masood, A. Srivastava, H.C. Thuwal and M. Ahmad, “REAL-TIME SIGN LANGUAGE GESTURE (WORD) RECOGNITION FROM VIDEO SEQUENCES USING CNN AND RNN”, Springer Nature Singapore Pte Ltd. 2018.

2) Kartik Shenoy, Tejas Dastane, Varun Rao, Devendra Vyavaharkar, “REAL-TIME INDIAN SIGN LANGUAGE (ISL) RECOGNITION” ,2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT), IEEE.

3)TalkingHands.co.in, “Talking Hands,” 2014. [Online]. Available: http://www.talkinghands.co.in/. [Accessed: 21- Jul- 2017].

4)A. Agarwal and M. K. Thakur, “Sign Language Recognition using Microsoft Kinect,” Sixth International Conference on Contemporary Computing (IC3), September 2013.

5) MailOnline, ‘'SignAloud gloves translate sign language gestures into spoken English,”2016.[Online].Available:http://www.dailymail.co.uk/sciencetech/article-3557362/SignAloudgloves-translate-sign-language-movements-spoken-English.html. . [Accessed: 10- Feb- 2018].

6)Alexia. Tsotsis, “MotionSavvy Is A Tablet App That Understands Sign Language,”2014.[Online].Available:https://techcrunch.com/2014/06/06/motionsavvy-is-a-tablet-app-thatunderstands-sign-language/. [Accessed: 10 – Feb-2018].

7) P. Paudyal, A. Banerjee and S. K. S. Gupta, “SCEPTRE: a Pervasive, Non-Invasive, and ProgrammableGesture Recognition Technology,” Proceedings of the 21st International Conference on Intelligent User Interfaces, pp. 282-293, 2016.

8)R. Y. Wang and J. Popovic, “Real-Time Hand-Tracking with a Color Glove,” ACM transactions on graphics (TOG), vol. 28, no. 3, 2009.

9)R. Akmeliawati , M. P. L. Ooi and Y. C. Kuang, “Real-Time Malaysian Sign Language Translation using Colour Segmentation and Neural Network,” Instrumentation and Measurement Technology Conference Proceedings, 2007