# CHAPTER ONE INTRODUCTION

## **Background of Study**

Automatically understanding spoken natural language enables a variety of applications. Spoken language is our most natural form of interaction when working with people, but has remained out of reach as a reliable interface between humans and machines. As mobile and ubiquitous computing expands into everyday life the desire for natural, low-friction interfaces increases. Building spoken language understanding (SLU) systems requires solving several sub-problems, with each posing a significant challenge in its own right. Currently deployed systems such as Apple Siri, Google Now, and Microsoft Cortana rely on fairly primitive speech interfaces characterized by a user speaking one of several possible commands, and the system executing the recognized command. These systems are powerful examples of the possibilities for spoken language interfaces, but fall short of our dream of directing a system by having a conversation in the way we might when working with a human trying to complete the same tasks. The shortcomings of modern speech interfaces are a direct result of the difficulty of designing automated SLU systems. In this project we build machine learning approaches to simplify and improve each component of a spoken language system to make progress towards natural, conversational speech interfaces for automated systems.

## **Statement of the Problem**

The shortcomings of modern speech interfaces are a direct result of the difficulty of designing automated SLU systems. In this project we build machine learning approaches to simplify and improve each component of a spoken language system to make progress towards natural, conversational speech interfaces for automated systems.

## **Aim and Objectives of the Study**

The aim of the project is to build a neural network for recognizing lexical words in speech using python.

The objectives are:

1. Speech denoising with a neural network.
2. Word recognition with a convolutional neural network.

## **Scope of the Project**

The project focus only on audio and not visual.

## **Limitation of the Study**

The neural network is limited to small GPU cycle based on the capabilities of the hardware used for the project.

## **Significance of the Study**

The will enable the creation of a framework for recognizing speech patterns of the word spoken in Nigeria to facilitate better translations to other languages.

## **Project Organization**

The project is divided into five chapters. The outlines are presented below:

**Chapter One: Introduction**

Chapter one introduces this project work, the background of the study, the statement of the problem, the aim and objectives, the scope of the study, limitations of the study, the significance of the study, project organization, and the definition of terms.

**Chapter Two: Literature review**

This chapter focuses on the literature review, and the contributions of other scholars on the subject matter being discussed.

**Chapter Three: Methodology and Design**

This chapter is concerned with the presentation of the results of system analysis and design. It presents the research methodology used in the development of the system to facilitate an understanding and effective future implementation of the system.

**Chapter Four: System Implementation Evaluation**

This chapter describes the system implementation and documentation, analysis of modules, and system requirements for implementation.

**Chapter Five: Summary, Conclusion, and** **Recommendation**

The chapter provides a summary of major findings, conclusions, and recommendations based on the study conducted.

# CHAPTER TWO LITERATURE REVIEW



## **Introduction**

A look at what the algorithms, neural networks and dataset used the project are and a review of related literature.

## **Neural Networks**

Inspiration to create artificial neural networks (ANN), commonly referred to as neural networks (NN), was taken from the entirely different way in which human brain processes and computes information, compared to conventional digital computers. The human brain is a highly complex, nonlinear, and parallel information-processing system with the capability to organize it is structural components, known as neurons, to perform several different kinds of computation (e.g., perception, pattern recognition or motor functions). (Werner, 2020).

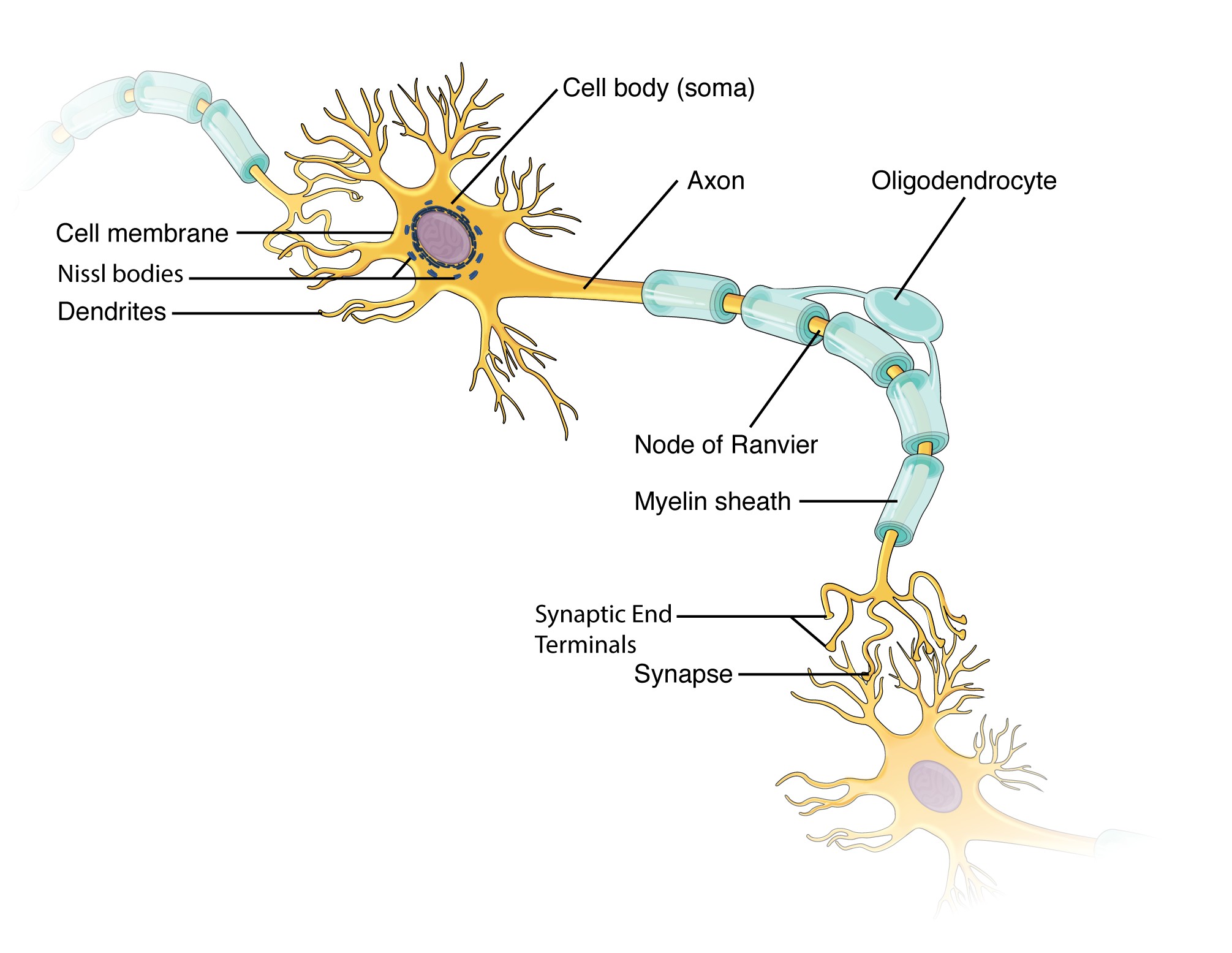


Figure 2.1: Neuron anatomy (Werner, 2020).

As shown in figure [2](#_bookmark2).1 every neuron has a branch-like structure of den-drites which serve as receivers of information from connected neurons or an- other kind of receptor/cell. From dendrites, the information is then passed as an electric signal through the cell body (soma) to the axon. The purpose of the axon is the transportation of the signal to adjacent cells. On the end of the axon is another branch-like structure called synaptic end terminals (or axon terminals). Those are the transfer points for the information to other cells. (Werner, 2020).

An essential feature of the brain, which is called brain plasticity, is the ability to adapt to its surrounding environment. And just as the plasticity is the key to the functioning of the brain, the same applies to the artificial neural networks and their artificial neurons.

In its most general form, a neural network is a machine that tries to model how the human brain performs a particular task or function of interest. The network is usually implemented by using electronic components or is simulated by software on a digital computer. With this in mind, let us define neural networks as follows:

A neural network is a massively parallel distributed processor made up of simple processing units that has a natural propensity for stor- ing experiential knowledge and making it available for use. It re- sembles the brain in two respects:

1. Knowledge is acquired by the network from its environment through a learning process.
2. Inter-neuron connection strengths, known as synaptic weights, are used to store the acquired knowledge. (Werner, 2020).

The mentioned learning process is performed by a learning algorithm, which is a function that modifies the synaptic weights of the network to fit it to a specific problem or objective. Apart from modifying the internal weights, some networks are capable of even modifying their topology to suit themselves to a problem better.

In the following sections, I will go through the first concepts and history of neural networks, their learning algorithms, and advanced types of NN.

As shown in figure 2.1, every neuron has a branch-like structure of den- drites which serve as receivers of information from connected neurons or an- other kind of receptor/cell. From dendrites, the information is then passed as an electric signal through the cell body (soma) to the axon. The purpose of the axon is the transportation of the signal to adjacent cells. On the end of the axon is another branch-like structure called synaptic end terminals (or axon terminals). Those are the transfer points for the information to other cells. (Werner, 2020).

An essential feature of the brain, which is called brain plasticity, is the ability to adapt to its surrounding environment. And just as the plasticity is the key to the functioning of the brain, the same applies to the artificial neural networks and their artificial neurons.

In its most general form, a neural network is a machine that tries to model how the human brain performs a particular task or function of interest. The network is usually implemented by using electronic components or is simulated by software on a digital computer. With this in mind, let us define neural networks as follows:

A neural network is a massively parallel distributed processor made up of simple processing units that has a natural propensity for stor- ing experiential knowledge and making it available for use. It re- sembles the brain in two respects:

1. Knowledge is acquired by the network from its environment through a learning process.
2. Inter-neuron connection strengths, known as synaptic weights, are used to store the acquired knowledge. (Werner, 2020).

The mentioned learning process is performed by a learning algorithm, which is a function that modifies the synaptic weights of the network to fit it to a specific problem or objective. Apart from modifying the internal weights, some networks are capable of even modifying their topology to suit themselves to a problem better.

In the following sections, I will go through the first concepts and history of neural networks, their learning algorithms, and advanced types of NN.

### **Multilayer Perceptron**

The multilayer perceptron (MLP) consists of multiple layers of neurons that interact using weighted connections. After the first input layer, there are usually several more hidden layers, followed by the last output layer. There are no connections between neurons in one layer, while all neurons in one layer are fully connected to neurons in adjacent layers. While this statement tends to create an impression that all the information from one layer is copied to all the neurons of the adjacent layer, it is not necessarily so. Since the connection weights are usually real numbers, some of the connections may effectively be rendered insignificant. (Werner, 2020). There is an example of a multilayer NN in figure 2.2[.](#_bookmark6)

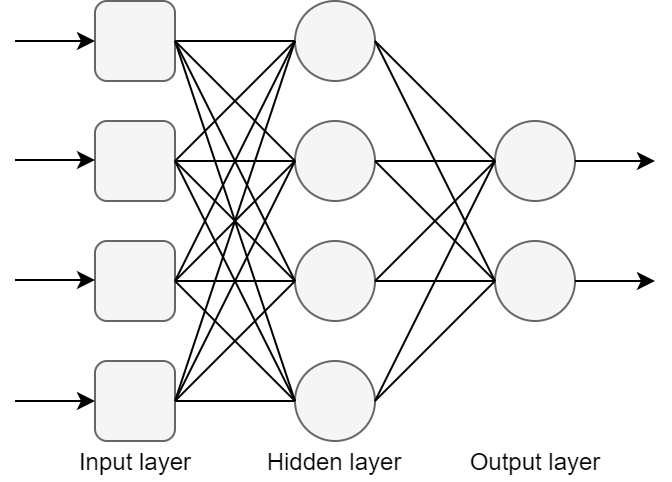


Figure 2.2: Multilayer NN example schema

Another difference over the simple perceptron is the activation function. Each model of neuron includes a nonlinear activation function that is differentiable.

Those characteristics, however, are responsible for the deficiencies in our understanding of the behavior of the network. The distributed non-linearity and high connectivity of the network make the analysis of the network quite hard to undertake. Moreover, the use of hidden layers makes the learning process harder to visualize.

### **Backpropagation**

Backpropagation, short for “backward propagation of errors”, is an algorithm for supervised learning of ANNs, which is using the gradient descend. In this

section, especially in the formal definition, I’ve drawn from  [(Werner, 2020).](#_bookmark86) It proceeds in two phases:

1. In the forward phase, the connection weights of the network are fixed, and the input is propagated through the network, layer by layer, until it reaches the output. In other words, the network performs inference.
2. In the backward phase, the error is computed by comparing the net- work output with the desired output. The resulting error is then propagated through the network, but this time from the output layer to the in- put. In this phase, appropriate adjustments are made to the connection weights based on the amount of error introduced by the corresponding connection. (Werner, 2020).

### **Recurrent Neural Networks**

A standard neural network takes in a fixed size vector as input, which limits its usage in situations that involve a series type input with no predetermined size. In other words, vanilla neural networks are unable to take into account any context or history, even if the current input is in a way based on the previous one. That is the reason why the recurrent neural networks (RNN) were designed and implemented.

The RNNs are made to take a series of input with no predetermined size. A single input item from the series is related to others, and likely influences its neighbors. So while they remember information from the training phase, they also learn things from prior inputs while generating outputs.

A recurrent network can take one or more input vectors and produce one or more output vectors. The outputs are influenced not just by weights applied on inputs like a regular NN, but also by a “hidden” state vector representing the context based on prior inputs/outputs. That means the same input could produce a different output depending on previous inputs in the series. (Werner, 2020).

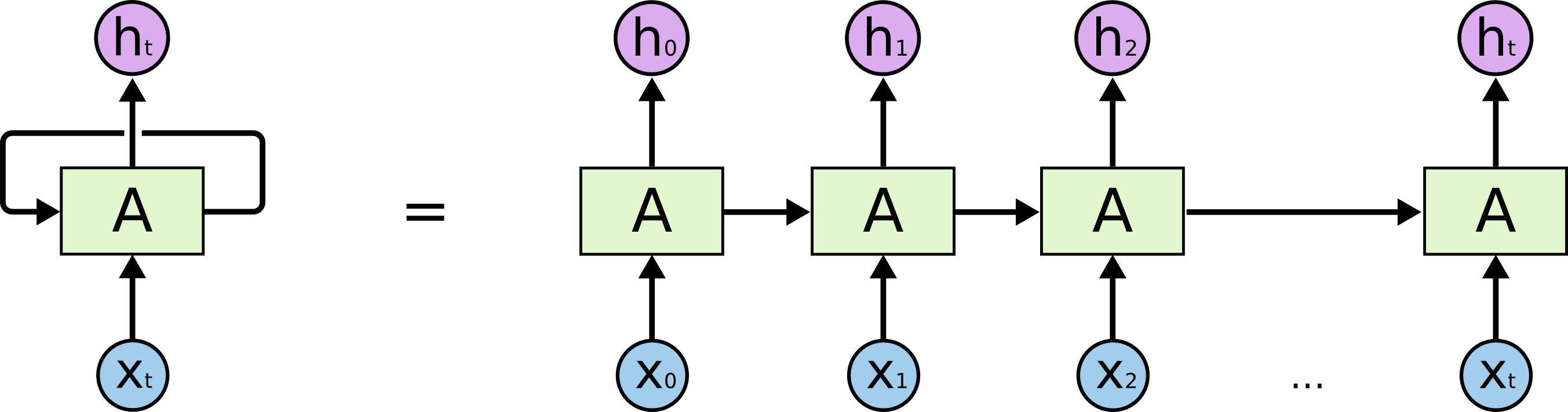


Figure 2.3: A recurrent neural network layer unfolding by (Werner, 2020).

The above-described concept is pictured in figure 2.3. On the unfolded neural network (the right side of the picture), we can see that part of the information from the NN in time 0 is passed to the network in time 1. That continues throughout the whole series of inputs, and in theory, the whole history of inputs and calculations will influence every other computation of the network.

There are many varieties, solutions, and constructive elements of recurrent neural networks. The difficulty of a recurrent network is that if we take into account each time step, each one of them must create its layer of neurons. That causes severe computational complexity. Besides, multilayer implementations are computationally unstable, since they tend to disappear or go off the scale. Restricting the computation to a fixed time window resolves those problems, but the model will not reflect the long-term trends. Various approaches are trying to improve the model of RNN memory and the mechanisms of remem- bering and forgetting. (Werner, 2020).

### **The Problem of Long-Term Dependencies**

Sometimes, for a task, we have enough recent information to be able to com- pute the output with certainty. For example, imagine a language model that predicts the next word based on the previous inputs/outputs. If it tries to predict the last word in the sentence “Clouds in the sky,” we do not need any context other then the rest of the sentence – it is quite apparent what would the last word regard (the sky/heaven). In such cases, where there is only a small gap between the necessary information, the standard RNNs can learn to use the recent information.

However, there are also times when we need a broader context. If a model were to predict the last words in the sentence “I grew up in France . . . I speak French fluently,” the recent context would probably suggest a name of a lan- guage. If it were to clarify which one, it would need the previous context, up to the beginning of the sentence. It is not rare that the gap between the necessary information and the place where it is needed becomes very large. Unfortu- nately, as the gap grows, the standard recurrent networks become unable to learn how to connect these pieces of information. As was mentioned above, in theory, the RNNs should handle such long-term dependencies. However, in practice, RNNs are not able to learn this.

### **Long-Short-Term Memory Networks**

Long-short-term memory (LSTM) networks are a special type of recurrent neural networks capable of learning long-term dependencies as well as short- term ones. They work very well on a large variety of problems and are cur- rently widely used. LSTMs are specifically designed to avoid the problem mentioned above of long-term dependencies, and they also manage to avoid the vanishing gradient problem.

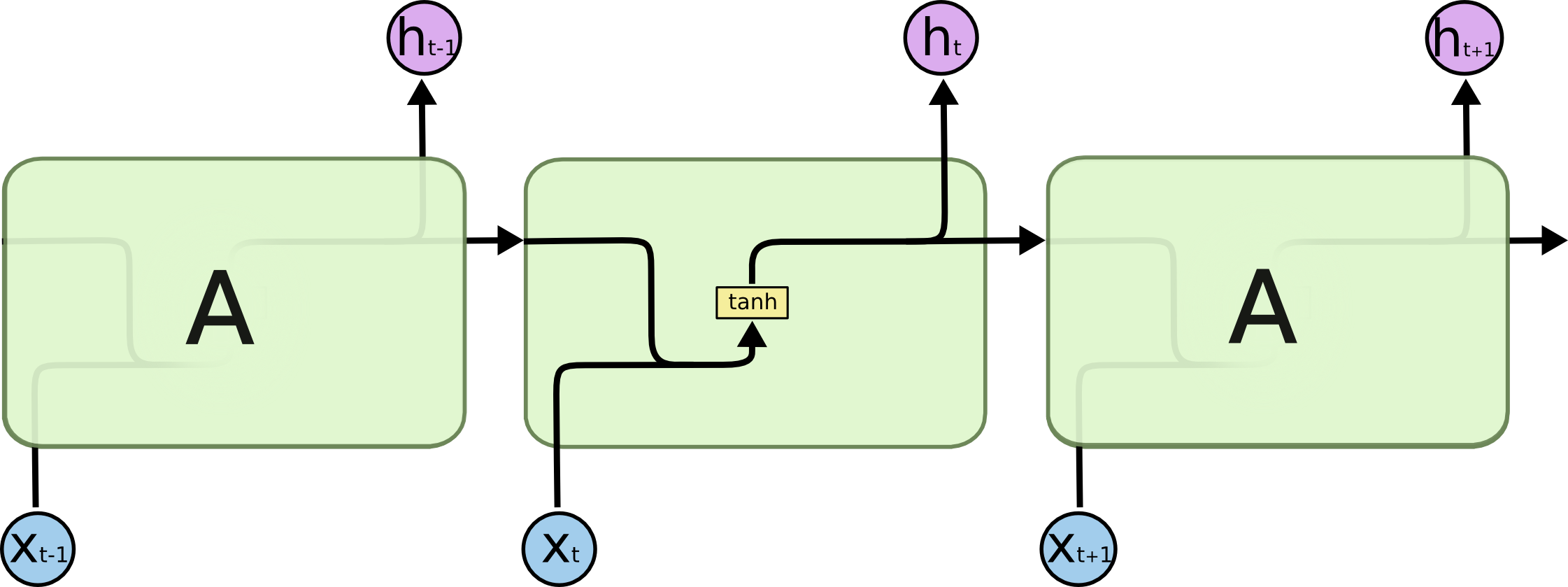


Figure 2.4: Standard RNN chain structure by (Werner, 2020).

Recurrent neural networks are in the form of a chain of repeating elements (neurons). In standard RNNs, those elements are usually of a straightforward structure – for example, one layer of a hyperbolic tangent (tanh). LSTM

networks have that same chain structure, but each of the elements implements a much more complex structure. Instead of one layer, they implement four layers that uniquely interact with each other.

The difference between the inner structures of standard and LSTM recurrent networks can be seen in figures 2.4 for the standard approach and 2.5 for the LSTMs. In the second diagram, each black line transmits a whole vector from one node to another. Merging lines mean vector concatenation, and branching lines are copying content without any alteration. Pink circles represent pointwise operators – vector addition or multiplication. Lastly, the yellow rectangles are the trained layers of the neural network. (Werner, 2020).

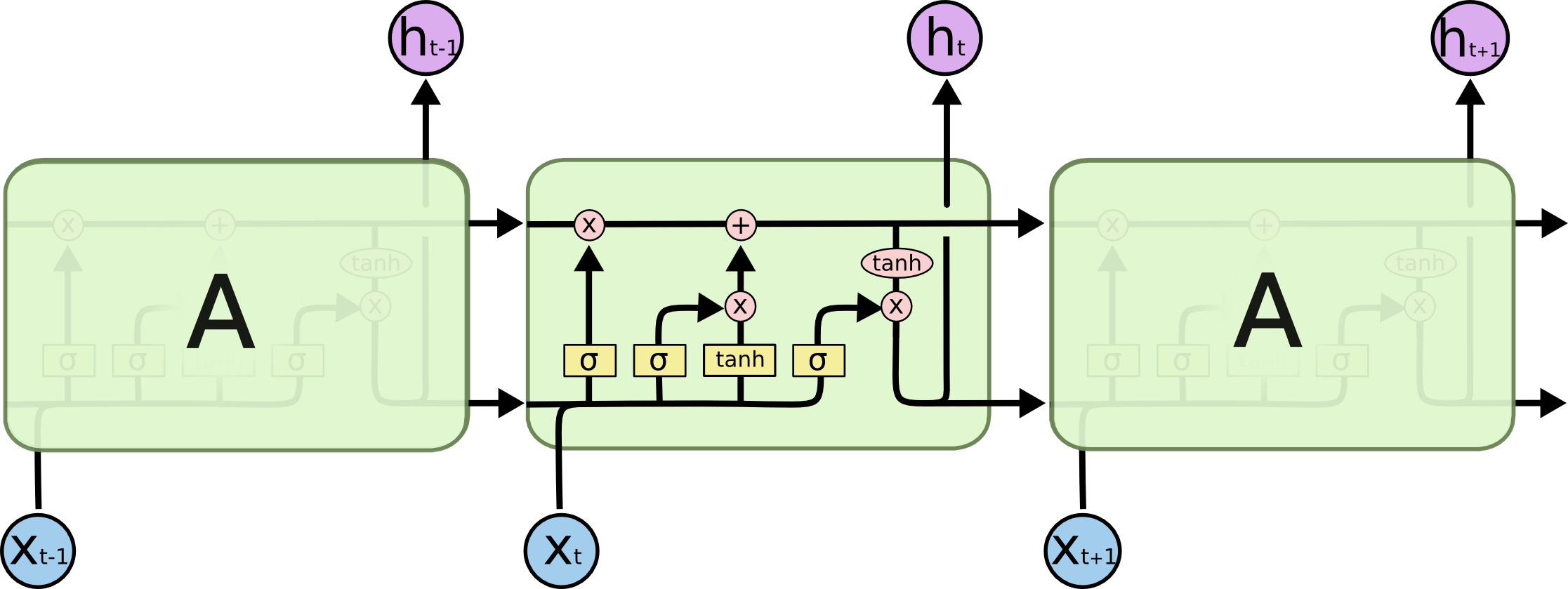
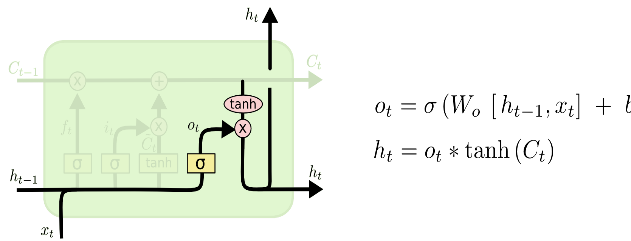
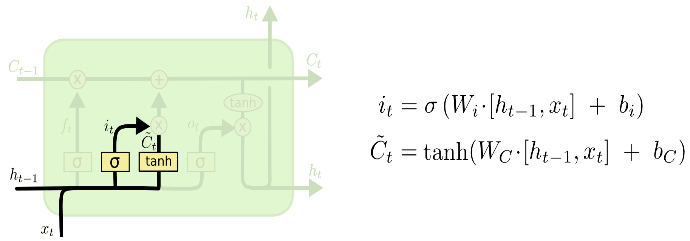
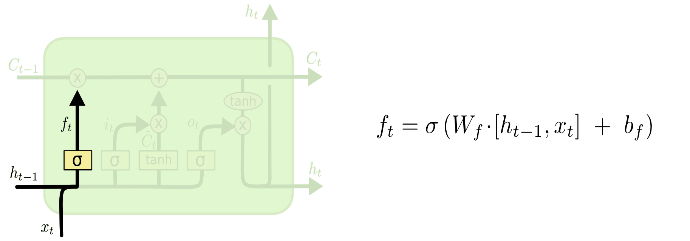


Figure 2.5: LSTM chain structure by (Werner, 2020).

The main feature of LSTMs is the cell/cellular state. That is the top horizontal line in the network cell. The neurons can remove or add information to a cellular state, while so-called gates carefully regulate the changes. They are composed of a sigmoid function and a pointwise multiplication operation. The sigmoid function returns values between 0 and 1, and it determines the magnitude of the change to the cellular state, whether it is information removal or addition.

To explain the information flow process through the LSTM cell, let us go step by step through it  [(Werner, 2020):](#_bookmark90)



The first gate layer (b) The second gate layer (c) The third gate layer

Figure 2.6: Three LSTM gate layers by (Werner, 2020).

## **Speech Recognition**

Both in human and electronic communication, the speech information is en- coded in the form of a continuously varying (analog) waveform that can be transmitted, recorded, manipulated, and ultimately decoded by a human listener. The primary analog form of the message is an acoustic waveform, which we call the speech signal. Those can be converted to an electrical waveform by a microphone, further manipulated by analog and digital signal processing, and then converted back to acoustic form by a loudspeaker or another electronic device.

Before we can apply any (digital) processing techniques, we must convert to the acoustic waveform, analog signal, to a sequence of numbers – digital signal. System or tool able to do such converting is called A-to-D converter, which creates digital representation by sampling the waveform in a very high rate, applies filters to preserve a prescribed bandwidth, and then reduces the sampling rate to the desired sampling rate. This discrete-time representation is the starting point for most digital signal processing applications.

Several areas that work with the digital representation of the speech signal follow. (Werner, 2020).

### **Speech coding**

Perhaps the most widespread applications of digital speech are the A-to-D and D-to-A converting mentioned briefly above. It is commonly referred to as speech coding or speech compression, and its goal is to compress the digital waveform representation of speech into a lower bit-rate representation. The D-to-A decoder part of the system is often called the synthesizer because it must reconstruct the speech waveform from the discrete digital data.

### **Text-to-Speech**

A Text-to-Speech analysis is another area that uses the digital signal. The goal of these systems is to start with text and automatically produce speech. The input is an ordinary text such as a newspaper article or an e-mail message. The system is depicted in figure 2.7. The first block named Linguistic Rules

text

Linguistic

Rules

Synthesis

Algorithm

D-to-A

Converter

speech

Figure 2.7: Text-to-Speech synthesis system inspired by (Werner, 2020).

has the job of converting the ordinary text into a set of sounds, which must be synthesized afterward. The system should include a set of linguistic rules that must determine appropriate sounds such as emphasis, pauses, or rates of speaking.

### **Automatic Speech Recognition**

Moreover, pronounce acronyms and ambiguous words like read, bass, or object, how to pronounce abbreviations like St. (street or Saint?), Dr. (Doctor or drive?), and adequately pronounce specialized terms, and names. After the system builds the pronunciation set of sounds, it is time to synthesize the speech – to create the appropriate sound sequence to represent the text message in the form of speech.

Text-to-Speech synthesis systems are used to do things like to provide voice output from GPS systems, handle call center help desks, or providing information from handheld devices such as foreign language phrasebooks and dictionaries. They are also being used in announcement machines that provide information – stock quotes and airline schedules. Finally, perhaps the most important application is in reading machines for the blind, where an optical character recognition system provides the text input.

### **Speech Recognition**

Quite the opposite of the Text-to-Speech problem described above is the speech recognition or Speech-to-Text problem. It is concerned with the extraction of information from the speech signal. Figure 2.7 shows a diagram of a generic approach to pattern matching problems in ASR. There are several sub-problems in this class, such as

speech

A-to-D

Converter

Feature

Analysis

Pattern

Matching

text

Figure 2.7: Generic pattern matching system inspired by (Werner, 2020).

1. speech recognition, where the goal is to extract the message from the speech signal,
2. speaker recognition, where the goal is to identify the speaker,
3. speaker verification, which verifies a speakers claimed identity from analysis of their speech signal,
4. word spotting, which involves monitoring a speech signal for the occurrence of a specified word or phrase, and
5. automatic indexing of speech recordings based on the recognition of spoken keywords.

The first block in the pattern matching system diagram 2.7 converts the analog speech waveform to the digital signal using an A-to-D converter. The feature analysis module, the second block, converts the sampled speech signal to aset of feature vectors. The third and final block in the system called pattern matching block dynamically time aligns the set of feature vectors representing the speech signal with a concatenated set of stored patterns. Then it chooses the identity associated with the pattern, which is the closest match to the time-aligned set of feature vectors of the speech signal. The symbolic output consists of a set of recognized words in the case of speech recognition. In the case of speaker recognition, it would be the identity of the best matching talker.

Probably the most common use of speech recognition is in portable com- munication devices. Spoken name speech recognition in cell phones enables voice dialing, and today basically every smart phone has its own voice assistant that is capable of multitude functions including speech-to-text when writing messages, the device options management, or information retrieval from the internet. (Werner, 2020).

The long-time dream of speech researchers are automatic language trans- lation systems. The goal of such systems is to convert spoken words in one language to spoken words in another language to facilitate natural language voice dialogues between people speaking different languages.

## **Related Literature**

Werner, (2020) This thesis deals with automatic speech recognition (ASR) using recurrent neural networks (RNN). The goal is to analyze the state-of-the-art in those fields and propose a suitable Czech open-source voice dataset and an RNN model. The output is a trained speech-to-text model, a new open-source dataset, and a system allowing accessible data preprocessing and further extension of datasets. The dataset of choice is the Czech Parliament meetings (CPM) transcribed recordings, and the model used is the DeepSpeech open-source project. The secondary source of speech data is the rest of the recording gathered from the CPM website. Part of the preprocessing relied on the usage of a voice activity detection (VAD) model, which was used as a reference for the audio segmentation. The trained model achieved 12.66 % WER (Word Error Rate) and 4.63 % CER (Character Error Rate), which were sufficient values for the final dataset transcription. After preprocessing, the final dataset consisted of over 580000 speech utterances of ranging length roughly from 1 up to 70 seconds.

Maas (2015) described approaches to improving individual components for each sub-task associated with spoken language understanding. The methods primarily rely on machine-learning-based approaches to replace hand-engineered approaches and consistently found that learning from data with minimal assumptions about a problem result in improved performance. In particular, we focus on neural network approaches to problems.

Koller (2010) thesis deals with accented speech recognition. The performance of a hybrid large vocabulary continuous speech recognizer, combining multi-layer perceptrons and hidden markov models, degrades heavily in the presence of African Portuguese varieties in broadcast news. Adapted and newly trained variety-specific acoustic and language models are shown to improve recognition significantly by up to 21.1%.

Dua et al. (2022) study extends the role of the CNN-based approach to robust and uncommon speech signals (tonal) using its own designed database for target research. The main objective of this research work was to develop a speech-to-text recognition system to recognize the tonal speech signals of Gurbani hymns using a CNN. Further, the CNN model, with six layers of 2DConv, 2DMax Pooling, and 256 dense layer units (Google’s TensorFlow service) was also used in this work, as well as Praat for speech segmentation. Feature extraction was enforced using the MFCC feature extraction technique, which extracts standard speech features and features of background music as well. The study reveals that the CNN-based method for identifying tonal speech sentences and adding instrumental knowledge performs better than the existing and conventional approaches. The experimental results demonstrate the significant performance of the present CNN architecture by providing an 89.15% accuracy rate and a 10.56% WER for continuous and extensive vocabulary sentences of speech signals with different tones.

Gravellier (2020) goal of the master thesis is to build the Automatic Speech Recognition system for these three tests using State of The Art techniques. After a study of the specific requirements from the doctors, we create the different blocks of an ASR system on the Kaldi toolkit: the acoustic models, the language model and the lexicon. By testing on the Evolex corpus, we optimize the system and adapt it to the target use.

Stenman (2015) thesis aims to give an introduction to speech recognition and discuss its use in robotics. An evaluation of Google Speech, using Google’s speech API, in regards to word error rate and translation speed, as well as a comparison between Google Speech and Pocketsphinx is made. The results show that Google Speech presented lower error rates on general purpose sentences but, due to the high average translation speed and the inability to specify vocabulary, was not suitable for voice-controlled moving robots.

Bensch (2021) This work explores and evaluates diFFerent methods to improve automatic speech recognition of new and rare words. The results show that the use of an n-gram Language Model (LM) improves the overall performance of our baseline ASR model, by decreasing its WER by 1.8%. Especially the recognition of rare words was significantly improved. For example, words seen once (one-shot-learning) improved in accuracy from 57% up to 84%. However, due to the fixed vocabulary in the language model, zero-shot learning (recognition of new words) was eliminated.

# CHAPTER THREE METHODLOGY AND DESIGN



## **Introduction**

This chapter covers the analysis models (data flow diagram, sequence, class and entity relationship diagram) of the project and the higher-level solution (programming language) approach used.

## **Automatic Speech Recognition**

Problems like audio classification start with a sound clip and predict which class that sound belongs to, from a given set of classes. For Speech-to-Text problems, your training data consists of:

Input features (X): audio clips of spoken words

Target labels (y): a text transcript of what was spoken

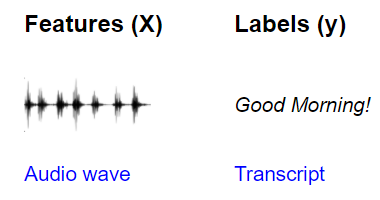


Figure 3.1: Automatic Speech Recognition

Automatic Speech Recognition uses audio waves as input features and the text transcript as target labels (Image by Author)

The goal of the model is to learn how to take the input audio and predict the text content of the words and sentences that were uttered.

## **Data pre-processing**

The transforms that are used to process audio data for deep learning models. With human speech as well we follow a similar approach. There are several Python libraries that provide the functionality to do this, with librosa being one of the most popular.

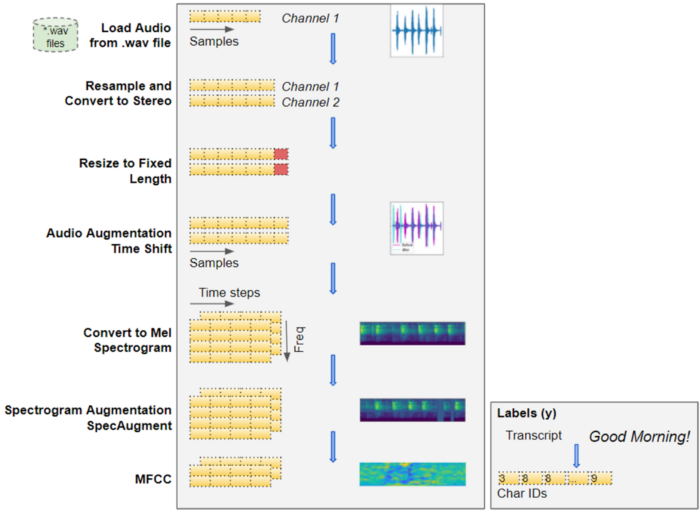


Figure 3.2: Transforming raw audio waves to spectrogram images for input to a deep learning model

### **Load Audio Files**

1. Start with input data that consists of audio files of the spoken speech in an audio format such as “.wav” or “.mp3”.
2. Read the audio data from the file and load it into a 2D Numpy array. This array consists of a sequence of numbers, each representing a measurement of the intensity or amplitude of the sound at a particular moment in time. The number of such measurements is determined by the sampling rate. For instance, if the sampling rate was 44.1kHz, the Numpy array will have a single row of 44,100 numbers for 1 second of audio.
3. Audio can have one or two channels, known as mono or stereo, in common parlance. With two-channel audio, we would have another similar sequence of amplitude numbers for the second channel. In other words, our Numpy array will be 3D, with a depth of 2.

### **Convert to uniform dimensions: sample rate, channels, and duration**

1. We might have a lot of variation in our audio data items. Clips might be sampled at different rates, or have a different number of channels. The clips will most likely have different durations. As explained above this means that the dimensions of each audio item will be different.
2. Since our deep learning models expect all our input items to have a similar size, we now perform some data cleaning steps to standardize the dimensions of our audio data. We resample the audio so that every item has the same sampling rate. We convert all items to the same number of channels. All items also have to be converted to the same audio duration. This involves padding the shorter sequences or truncating the longer sequences.
3. If the quality of the audio was poor, we might enhance it by applying a noise-removal algorithm to eliminate background noise so that we can focus on the spoken audio.

### **Data Augmentation of raw audio**

We could apply some data augmentation techniques to add more variety to our input data and help the model learn to generalize to a wider range of inputs. We could Time Shift our audio left or right randomly by a small percentage, or change the Pitch or the Speed of the audio by a small amount.

### **Mel Spectrograms**

This raw audio is now converted to Mel Spectrograms. A Spectrogram captures the nature of the audio as an image by decomposing it into the set of frequencies that are included in it.

### **MFCC**

For human speech, in particular, it sometimes helps to take one additional step and convert the Mel Spectrogram into MFCC (Mel Frequency Cepstral Coefficients). MFCCs produce a compressed representation of the Mel Spectrogram by extracting only the most essential frequency coefficients, which correspond to the frequency ranges at which humans speak.

### **Data Augmentation of Spectrograms**

We can now apply another data augmentation step on the Mel Spectrogram images, using a technique known as SpecAugment. This involves Frequency and Time Masking that randomly masks out either vertical (ie. Time Mask) or horizontal (ie. Frequency Mask) bands of information from the Spectrogram. NB: I’m not sure whether this can also be applied to MFCCs and whether that produces good results.

We have now transformed our original raw audio file into Mel Spectrogram (or MFCC) images after data cleaning and augmentation.

We also need to prepare the target labels from the transcript. This is simply regular text consisting of sentences of words, so we build a vocabulary from each character in the transcript and convert them into character IDs.

This gives us our input features and our target labels. This data is ready to be input into our deep learning model.

## **Architecture**

There are many variations of deep learning architecture for ASR. Two commonly used approaches are:

1. A CNN (Convolutional Neural Network) plus RNN-based (Recurrent Neural Network) architecture that uses the CTC Loss algorithm to demarcate each character of the words in the speech. eg. Baidu’s Deep Speech model.
2. An RNN-based sequence-to-sequence network that treats each ‘slice’ of the spectrogram as one element in a sequence eg. Google’s Listen Attend Spell (LAS) model.

At a high level, the model consists of these blocks:

1. A regular convolutional network consisting of a few Residual CNN layers that process the input spectrogram images and output feature maps of those images.

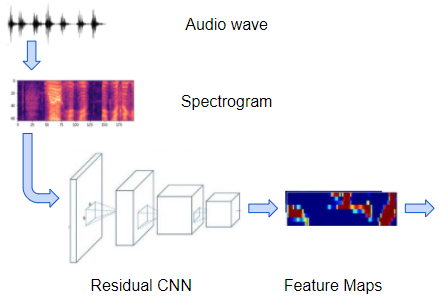


Figure 3.3: Spectrograms are processed by a convolutional network to produce feature maps

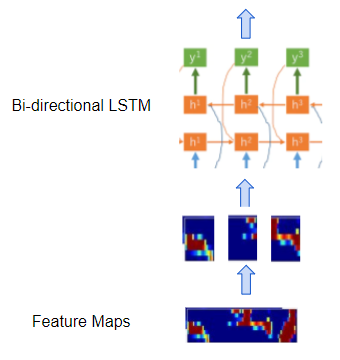
1. A regular recurrent network consisting of a few Bidirectional LSTM layers that process the feature maps as a series of distinct timesteps or ‘frames’ that correspond to our desired sequence of output characters. (An LSTM is a very commonly used type of recurrent layer, whose full form is Long Short Term Memory). In other words, it takes the feature maps which are a continuous representation of the audio, and converts them into a discrete representation.

Figure 3.4: Recurrent network processes frames from the feature maps (Image by Author)

1. A linear layer with softmax that uses the LSTM outputs to produce character probabilities for each timestep of the output.

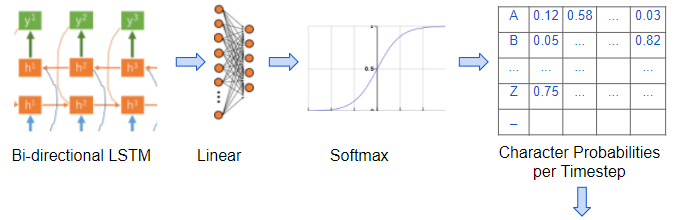


Figure 3.5: Linear layer generates character probabilities for each timestep (Image by Author)

1. We also have linear layers that sit between the convolution and recurrent networks and help to reshape the outputs of one network to the inputs of the other.

So the model takes the Spectrogram images and outputs character probabilities for each timestep or ‘frame’ in that Spectrogram.

## **Align the sequences**

The eventual goal is to map those timesteps or ‘frames’ to individual characters in our target transcript.

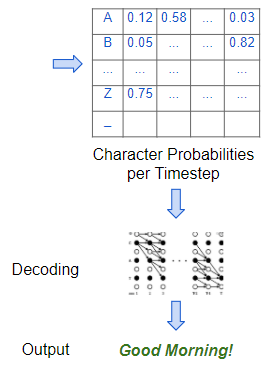


Figure 3.6: The model decodes the character probabilities to produce the final output

But for a particular spectrogram, how do we know how many frames there should be? How do we know exactly where the boundaries of each frame are? How do we align the audio with each character in the text transcript?

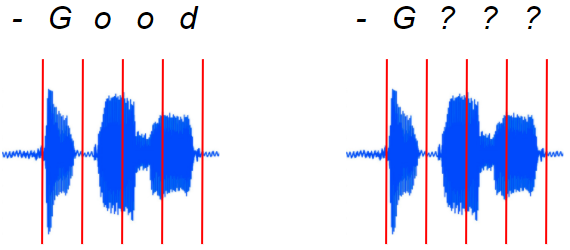


Figure 3.7: On the left is the alignment we need. But how do we get it?

The audio and the spectrogram images are not pre-segmented to give us this information.

1. In the spoken audio, and therefore in the spectrogram, the sound of each character could be of different durations.
2. There could be gaps and pauses between these characters.
3. Several characters could be merged together.
4. Some characters could be repeated. eg. in the word ‘apple’, how do we know whether that “p” sound in the audio actually corresponds to one or two “p”s in the transcript?

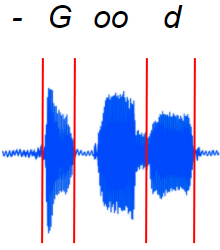


Figure 3.8: In reality, spoken speech is not neatly aligned for us

This is actually a very challenging problem, and what makes ASR so tough to get right. It is the distinguishing characteristic that differentiates ASR from other audio applications like classification and so on. The way we tackle this is by using an ingenious algorithm with a fancy-sounding name — it is called Connectionist Temporal Classification, or CTC for short.

## **CTC Algorithm — Training and Inference**

CTC is used to align the input and output sequences when the input is continuous and the output is discrete, and there are no clear element boundaries that can be used to map the input to the elements of the output sequence.

What makes this so special is that it performs this alignment automatically, without requiring you to manually provide that alignment as part of the labeled training data. That would have made it extremely expensive to create the training datasets.

As we discussed above, the feature maps that are output by the convolutional network in our model are sliced into separate frames and input to the recurrent network. Each frame corresponds to some timestep of the original audio wave. However, the number of frames and the duration of each frame are chosen by you as hyperparameters when you design the model. For each frame, the recurrent network followed by the linear classifier then predicts probabilities for each character from the vocabulary.

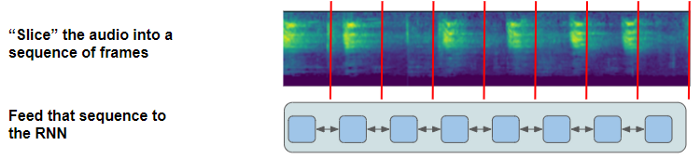


Figure 3.9: The continuous audio is sliced into discrete frames and input to the RNN

The job of the CTC algorithm is to take these character probabilities and derive the correct sequence of characters.

To help it handle the challenges of alignment and repeated characters that we just discussed, it introduces the concept of a ‘blank’ pseudo-character (denoted by “-”) into the vocabulary. Therefore the character probabilities output by the network also include the probability of the blank character for each frame.

Note that a blank is not the same as a ‘space’. A space is a real character while a blank means the absence of any character, somewhat like a ‘null’ in most programming languages. It is used only to demarcate the boundary between two characters.

CTC works in two modes:

1. CTC Loss (during Training): It has a ground truth target transcript and tries to train the network to maximize the probability of outputting that correct transcript.
2. CTC Decoding (during Inference): Here we don’t have a target transcript to refer to, and have to predict the most likely sequence of characters.

### **CTC Decoding**

1. Use the character probabilities to pick the most likely character for each frame, including blanks. eg. “-G-o-ood”

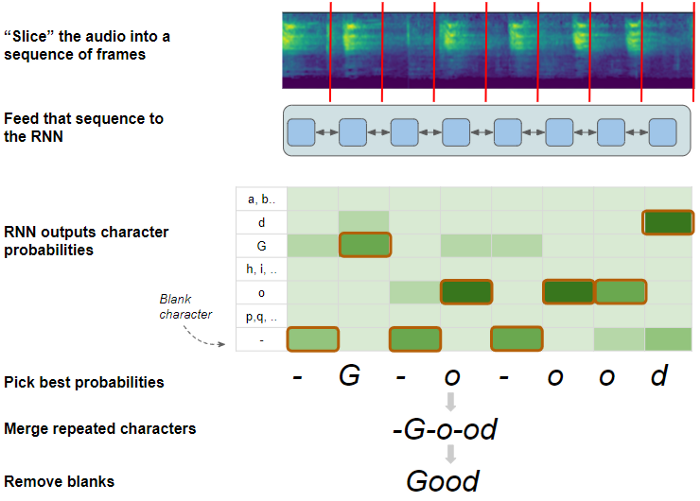


Figure 3.10: CTC Decode algorithm

1. Merge any characters that are repeated, and not separated by a blank. For instance, we can merge the “oo” into a single “o”, but we cannot merge the “o-oo”. This is how the CTC is able to distinguish that there are two separate “o”s and produce words spelled with repeated characters. eg. “-G-o-od”
2. Finally, since the blanks have served their purpose, it removes all blank characters. eg. “Good”.

### **CTC Loss**

The Loss is computed as the probability of the network predicting the correct sequence. To do this, the algorithm lists out all possible sequences the network can predict, and from that it selects the subset that match the target transcript.

To identify that subset from the full set of possible sequences, the algorithm narrows down the possibilities as follows:

1. Keep only the probabilities for characters that occur in the target transcript and discard the rest. eg. It keeps probabilities only for “G”, “o”, “d”, and “-”.
2. Using the filtered subset of characters, for each frame, select only those characters which occur in the same order as the target transcript. eg. Although “G” and “o” are both valid characters, an order of “Go” is a valid sequence whereas “oG” is an invalid sequence.

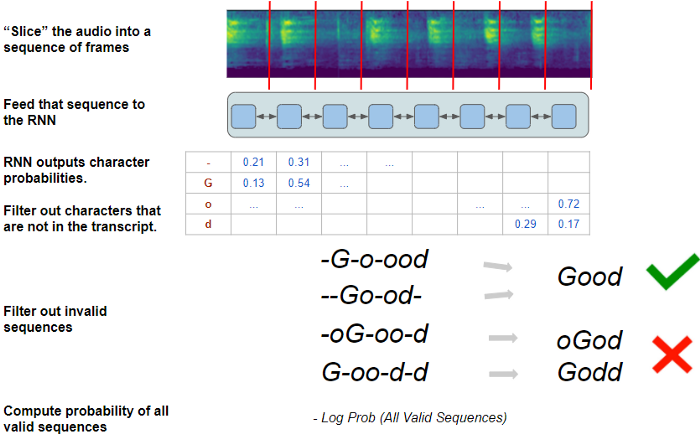


Figure 3.11: CTC Loss algorithm

With these constraints in place, the algorithm now has a set of valid character sequences, all of which will produce the correct target transcript. eg. Using the same steps that were used during Inference, “-G-o-ood” and “ — Go-od-” will both result in a final output of “Good”.

It then uses the individual character probabilities for each frame, to compute the overall probability of generating all of those valid sequences. The goal of the network is to learn how to maximize that probability and therefore reduce the probability of generating any invalid sequence.

Strictly speaking, since a neural network minimizes loss, the CTC Loss is computed as the negative log probability of all valid sequences. As the network minimizes that loss via back-propagation during training, it adjusts all of its weights to produce the correct sequence.

To actually do this, however, is much more complicated than what I’ve described here. The challenge is that there is a huge number of possible combinations of characters to produce a sequence. With our simple example alone, we can have 4 characters per frame. With 8 frames that gives us 4 \*\* 8 combinations (= 65536). For any realistic transcript with more characters and more frames, this number increases exponentially. That makes it computationally impractical to simply exhaustively list out the valid combinations and compute their probability.

Solving this efficiently is what makes CTC so innovative. It is a fascinating algorithm and it is well worth understanding the nuances of how it achieves this. That merits a complete article by itself which I plan to write shortly. But for now, we have focused on building intuition about what CTC does, rather than going into how it works.

## **Metrics — Word Error Rate (WER)**

After training our network, we must evaluate how well it performs. A commonly used metric for Speech-to-Text problems is the Word Error Rate (and Character Error Rate). It compares the predicted output and the target transcript, word by word (or character by character) to figure out the number of differences between them.

A difference could be a word that is present in the transcript but missing from the prediction (counted as a Deletion), a word that is not in the transcript but has been added into the prediction (an Insertion), or a word that is altered between the prediction and the transcript (a Substitution).

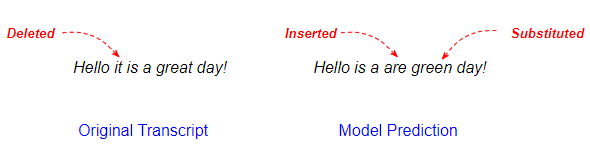


Figure 3.12: Count the Insertions, Deletions, and Substitutions between the Transcript and the Prediction

The metric formula is fairly straightforward. It is the percent of differences relative to the total number of words.

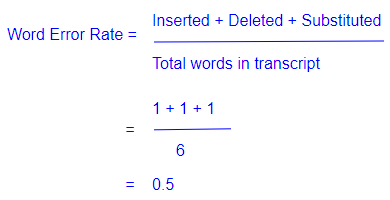


Figure 3.13: Word Error Rate computation

## **Language Model**

So far, our algorithm has treated the spoken audio as merely corresponding to a sequence of characters from some language. But when put together into words and sentences will those characters actually make sense and have meaning?

A common application in Natural Language Processing (NLP) is to build a Language Model. It captures how words are typically used in a language to construct sentences, paragraphs, and documents. It could be a general-purpose model about a language such as English or Korean, or it could be a model that is specific to a particular domain such as medical or legal.

Once you have a Language Model, it can become the foundation for other applications. For instance, it could be used to predict the next word in a sentence, to discern the sentiment of some text (eg. is this a positive book review), to answer questions via a chatbot, and so on.

## **Neural Network Models**

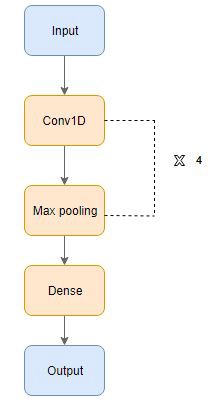


Figure3.14: Architecture Model

## **Hardware Setup**

1. CPU: 2.6GHz core i5
2. RAM: 8GB
3. GPU: 2GB Nvidia GeForce

**CHAPTER FOUR**

**SYSTEM IMPLEMENTATION EVALUATION**

**4.1 Introduction**

This section describes in detail how the new system will be implemented in order to assure its efficacy. It illustrates instances of functional (new) systems as well as how the system will be implemented.

* 1. **System Testing and Evaluation**

The developed system should be tested for a variety of reasons. For example, only via testing will we be able to detect and address any problems in the new system. Unit and integration testing were used in this project to confirm the design's efficacy and efficiency, as well as to ensure the new system satisfies its functional requirements and is error-free.

**Unit Testing**

specific units or single components of the system are examined individually in this part to confirm that specific phases function properly and without problems.

**Integration Testing**

The program was tested via integration testing, in which all of the components were integrated and worked as one. The connection between the different components was examined to ensure that they are correctly integrated and that the units can function as a unit.

**4.3 System Installation**

In order to use the proposed application on any computer system, the following steps need to be taken:

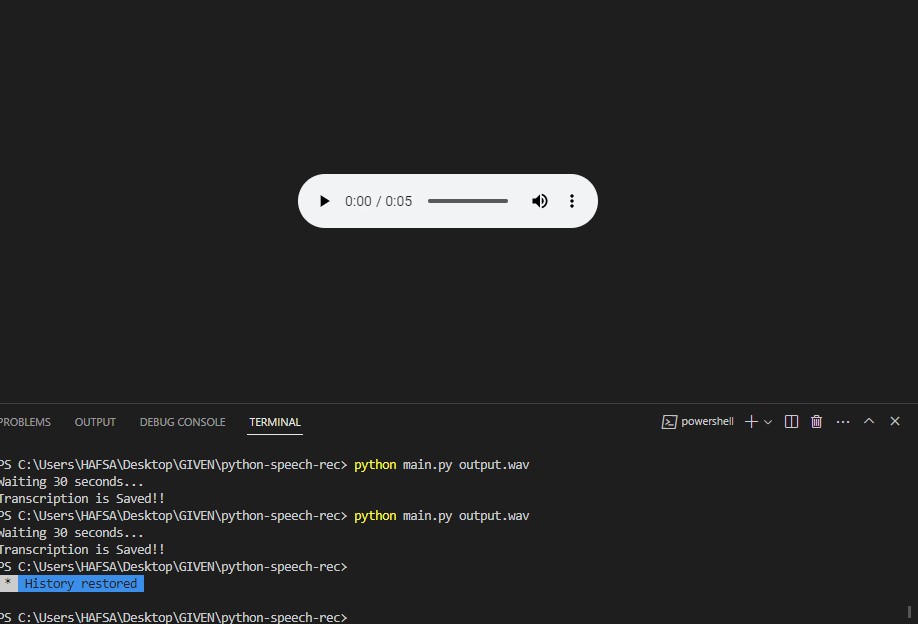
1. Make sure, android studio, JDK, and Android emulator are installed on the system.
2. Copy your project folder to any location of your choice.
3. Open the project folder in Visual Studio Code
4. In the terminal run “flutter pub get” to get all the dependencies in the pubspec.yaml file
5. Select the Android emulator as the device to be used.
6. Locate the main.dart file and run the file in debug mode.

**4.4 Security Measures**

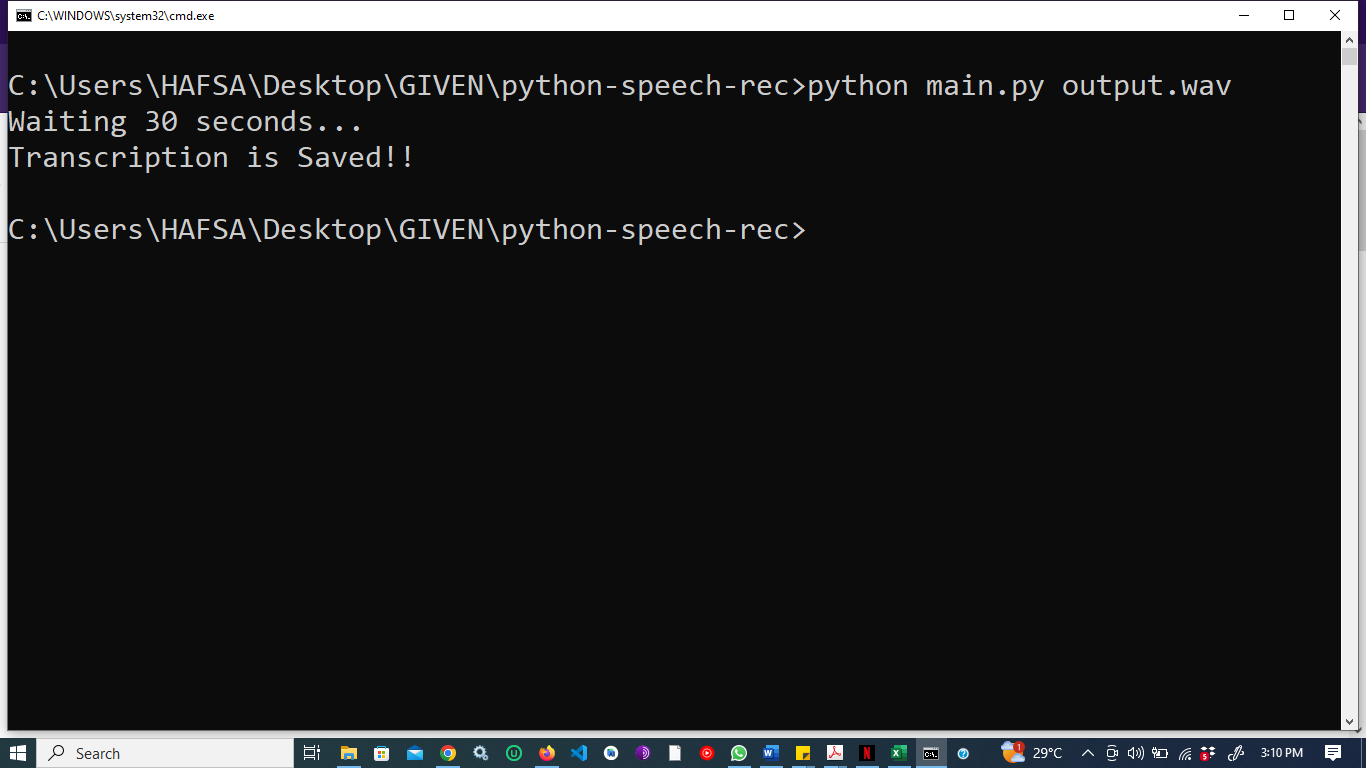
Since the scope of the application is public, literally all the information is made available to any user (students and admin), but some functionalities are restricted to the admin, functionalities that have to do with creating the student accounts, creating the candidates, managing the voting periods etc are restricted from the general student. The restriction is carried out by using passwords when the application is accessed.

**4.5 Sample Outputs**

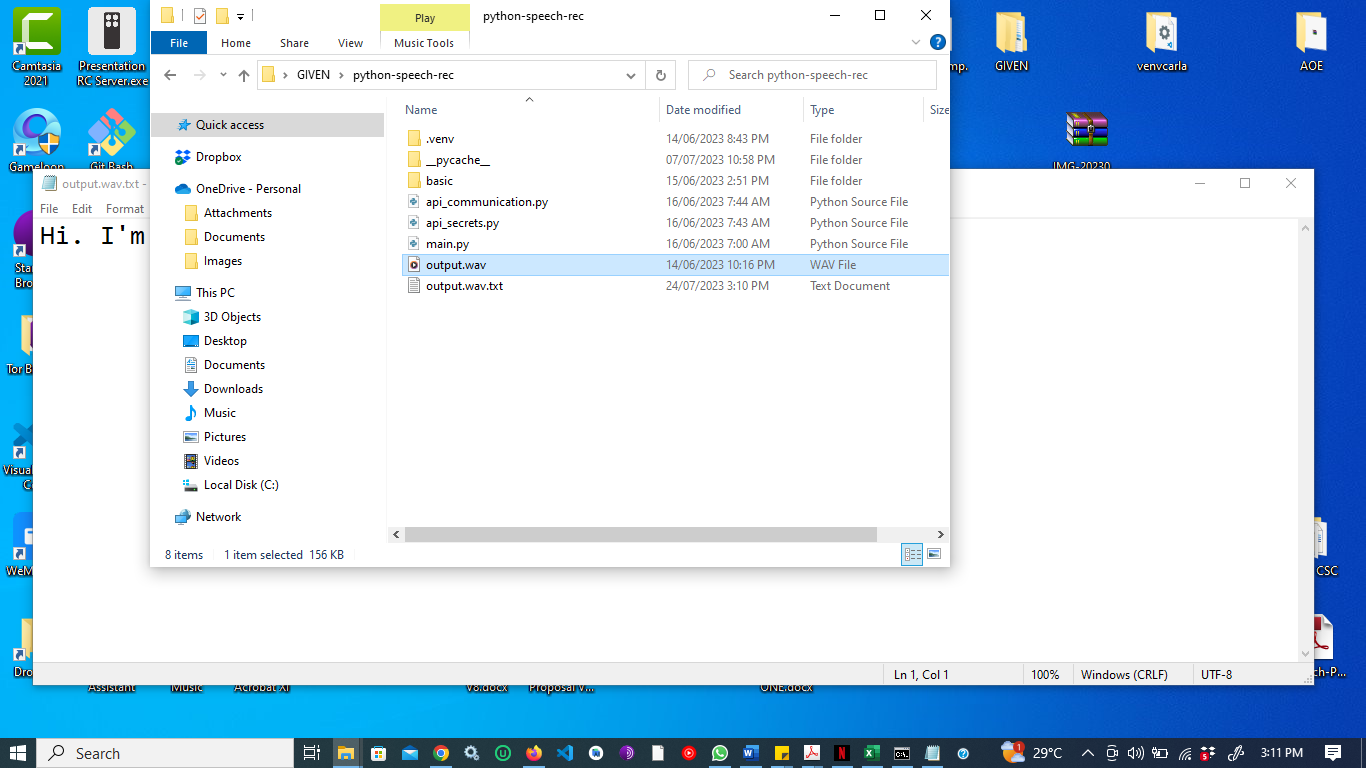
These describe and give the pictorial representation of the program or software; it shows and gives clear understanding of the design, and displays all the interfaces



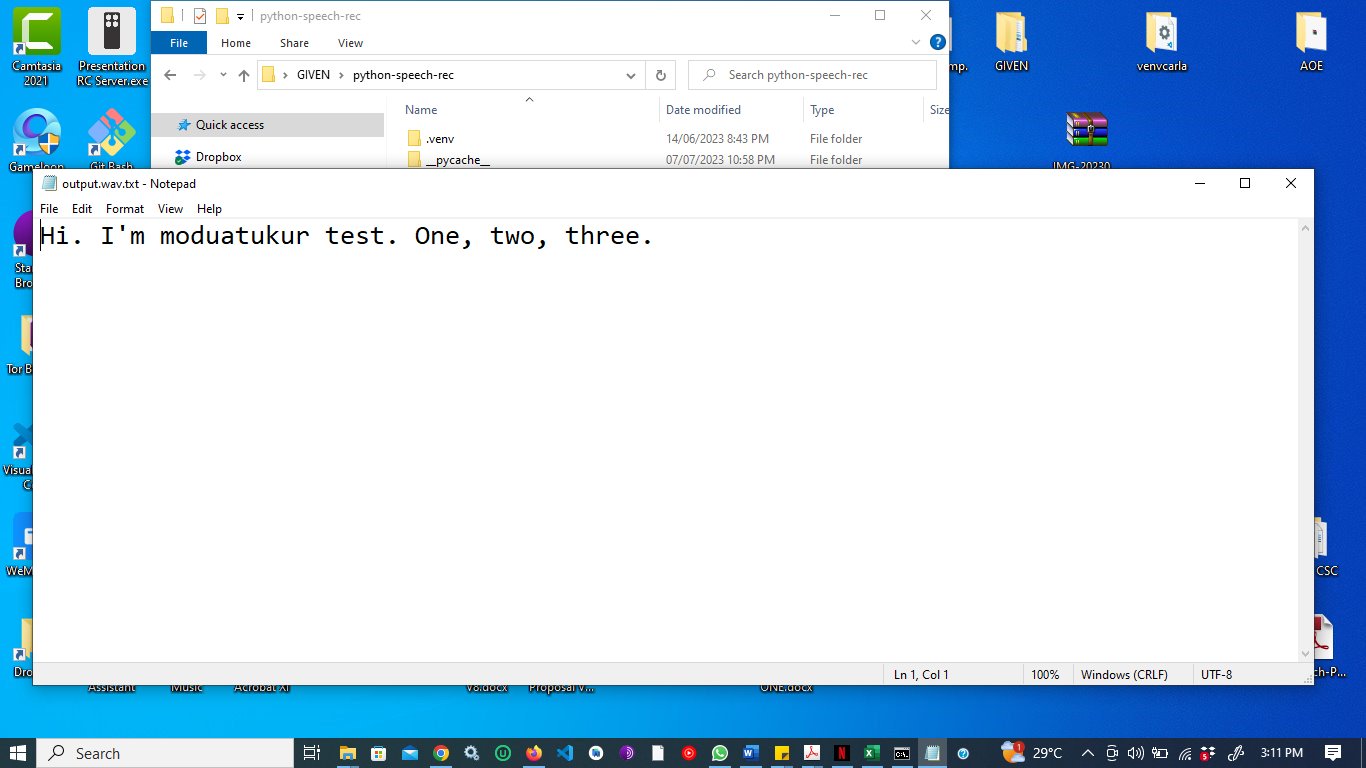
**Fig 4.1 Recording Speech**



**Fig 4.2 Transcription Saved**



**Fig 4.3 Output of Sound in Explorer**



**Fig 4.4 Transcribed Audio**

**CHAPTER FIVE**

**SUMMARY CONCLUSION AND RECOMMENDATION**

**5.1 Summary**

Automated spoken language understanding (SLU) systems have the potential to revolutionize human-machine interactions, providing natural and seamless interfaces. However, the current limitations of speech interfaces stem from the complexities in designing automated SLU systems. In this project, the focus is on leveraging machine learning techniques to enhance each component of a spoken language system and move towards more conversational speech interfaces for automated systems. The objective is to build a neural network that can recognize lexical words in speech using Python, with specific tasks like speech denoising and word recognition. The scope is limited to audio processing, excluding visual aspects. By developing a robust framework for recognizing speech patterns, especially in the context of Nigerian languages, the study aims to contribute to better translations and communication across various languages, paving the way for more efficient and natural human-computer interactions.

**5.2 Conclusion**

In conclusion, the development of automated spoken language understanding (SLU) systems has the potential to transform human-machine interactions, offering natural and intuitive interfaces. While current speech interfaces like Apple Siri and Google Now demonstrate the possibilities, they still fall short of the dream of engaging in truly conversational interactions with automated systems. The challenges lie in designing reliable and efficient SLU systems, which involve tackling various sub-problems. In this project, machine learning approaches have been utilized to simplify and enhance each component of a spoken language system, aiming to move closer to the vision of natural and conversational speech interfaces. By focusing on audio processing and building a neural network for word recognition, the study seeks to advance the field and create a framework that can recognize speech patterns, particularly in Nigerian languages, leading to improved translations and smoother communication across languages. The significance of this research lies in its potential to bridge the gap between humans and machines, offering more seamless and efficient interactions in the realm of automated systems and natural language understanding.

**5.3 Recommendation**

Based on the research and development of automated spoken language understanding (SLU) systems, the following recommendations are proposed to further improve the effectiveness and usability of speech interfaces:

1. Privacy and Security: Ensure stringent privacy and security measures to protect user data, as spoken language interactions may involve sensitive information.
2. Multilingual Support: Extend the SLU system's capabilities to support multiple languages, particularly regional and less common languages, to cater to a broader user base.
3. Cross-Platform Compatibility: Ensure that the SLU system is compatible with various devices and platforms, including smartphones, tablets, smart speakers, and IoT devices.
4. Robust Noise Handling: Develop robust noise reduction and denoising techniques to enhance speech recognition accuracy in noisy environments.

By implementing these recommendations, automated spoken language understanding systems can be further refined and become powerful tools for natural and seamless human-machine interactions. This will enhance user satisfaction, improve productivity, and open up new possibilities for the integration of speech interfaces in various domains, ranging from virtual assistants to smart homes and beyond

**REFERENCES**

Bensch, C. (2021). *Continuous Learning In Automatic Speech Recognition* (thesis). Maastricht University, Maastricht.

Dua, S., Kumar, S. S., Albagory, Y., Ramalingam, R., Dumka, A., Singh, R., Rashid, M., Gehlot, A., Alshamrani, S. S., & AlGhamdi, A. S. (2022). Developing a speech recognition system for recognizing tonal speech signals using a convolutional neural network. *Applied Sciences*, *12*(12), 1–13. https://doi.org/10.3390/app12126223

Gravellier, L. (2020). *Automatic speech recognition system for neurological tests* (thesis). INP Phelma, Grenoble.

Koller, O. T. A. (2010). *Automatic Speech Recognition and Identification of African Portuguese* (thesis). L2F, Lisbon.

Maas, A. L. (2015). *Deep Neural Networks In Speech Recognition* (thesis). Stanford University, Palo Alto.

Stenman, M. (2015). *Automatic speech recognition system for neurological tests* (thesis). Umea University, Umea.

Werner, R. (2020). *Developing an automatic speech recognition system based on Czech spoken language* (thesis). CTU, Prague.

**APPENDIX**

**Communication**

import requests

import time

from api\_secrets import API\_KEY\_ASSEMBLYAI

base\_url = "https://api.assemblyai.com/v2"

url = base\_url + "/transcript"

headers = {

    "authorization": API\_KEY\_ASSEMBLYAI

}

# upload

def upload(filename):

    def read\_f(filename, chunk\_size=5242880):

        with open(filename, "rb") as f:

            while True:

                data = f.read(chunk\_size)

                if not data:

                    break

                yield data

    response = requests.post(base\_url + "/upload",

                                headers=headers,

                                data=read\_f(filename))

    upload\_url = response.json()["upload\_url"]

    return upload\_url

# transcribe

def transcribe(upload\_url):

    data = {

        "audio\_url": upload\_url

    }

    response = requests.post(url, json=data, headers=headers)

    job\_id = response.json()['id']

    return job\_id

# poll

def poll(transcript\_id):

    polling\_url = url + '/' + transcript\_id

    polling\_response = requests.get(polling\_url, headers=headers)

    return polling\_response.json()

def get\_transcription\_result\_url(audio\_url):

    transcript\_id = transcribe(audio\_url)

    while True:

        data = poll(transcript\_id)

        if data['status'] == 'completed':

            return data, None

        elif data['status'] == 'error':

            return data, data['error']

        print('Waiting 30 seconds...')

        time.sleep(30)

# save transcript

def save\_transcript(audio\_url, filename):

    data, error = get\_transcription\_result\_url(audio\_url)

    if data:

        text\_filename = filename + ".txt"

        with open(text\_filename, "w") as f:

            f.write(data['text'])

        print("Transcription is Saved!!")

    elif error:

        print("Error", error)

**Recording**

import pyaudio

import wave

FRAMES\_PER\_BUFFER = 3200

FORMAT = pyaudio.paInt16

CHANNELS = 1

RATE = 16000

p = pyaudio.PyAudio()

stream = p.open(

    format=FORMAT,

    channels=CHANNELS,

    rate=RATE,

    input=True,

    frames\_per\_buffer=FRAMES\_PER\_BUFFER

)

print("start recording")

second = 5

frames = []

for i in range(0, int(RATE/FRAMES\_PER\_BUFFER\*second)):

    data = stream.read(FRAMES\_PER\_BUFFER)

    frames.append(data)

stream.stop\_stream()

stream.close()

p.terminate()

obj = wave.open("output.wav", "wb")

obj.setnchannels(CHANNELS)

obj.setsampwidth(p.get\_sample\_size(FORMAT))

obj.setframerate(RATE)

obj.writeframes(b"".join(frames))

obj.close()

**Main.py**

import sys

from api\_communication import \*

filename = sys.argv[1]

audio\_url = upload(filename)

save\_transcript(audio\_url, filename)