

**Research Work**

Machine Learning in Voice Detection



Professors: Student:

Jacinto Estima 201800011 Daniel Marçal

Vala Rohani

Contents

[**Introduction** 3](#_Toc41147556)

[**Literature Review** 4](#_Toc41147557)

[**Implementation** 9](#_Toc41147558)

[**Discussion on the Findings** 12](#_Toc41147559)

[**Conclusion** 12](#_Toc41147560)

[**References** 13](#_Toc41147561)

# **Introduction**

Voice detection or speech recognition is a subfield of computer science that develops methodologies and technology to enable a recognition and make a translation into a computer understandable language. In Machine Learning some speech recongnition systems require training, where an individual speaker reads some text into the system, and the the computer recognizes the fine-tune, and starts to increase its accuracy by adding more of those speeches into the model of voice detection.

In the early 50’s the consensus amongst computer scientists was that speech signal first needed to first be split into little phonetic units, then those units could be grouped into words. Even it seemed like it would work, the approach back in that time did not gave good results.

The first ever speech recognizer was called “Audrey” by Bell Labs in 1952, it could only recognize spoken numbers between 1 and 9, but the results weren’t promising either. A few years later, a team called DARPA created “harpy”, this used a 15000 connected nodes and each one represented any possible utterance, it was a brute-force search algorithm, that mapped the speech into the right nodes, to get the results. It was slightly better.

Some years after the Hidden Markov Model (HMM) was created, where each utterance was represented as a state, and used a statistical method to predict what a word/number was. They recongized that one word could pronounced differently from person to person, and even the same person could say the same word, but differently, it can have different durations. So this model was maintained throughout the 80’s and 90’s.

After that, it was time for neural networks, where famous Geoffrey Hinton kept on trying to devolop a neural network model until a couple years ago it started outperforming everything. The key was to give more data and computing power, and that is Deep Learning.

But our goal here is to see Machine Learning combined with Deep Learning in action like in Siri, Echo or Google. So, in this work it will be presented a today’s framework used and a Neural Network example, in order to understand how they complement each other, and take a closer approach on the created model with a phyton code.

# **Literature Review**

One of the biggest problems in this topic is the fact that people speech in different speeds. One person might say “Hello!” when another says “Heeelloooooo!”, so that results in different sound files, with some different data, where one is much longer than the other. Both files should recognize the exact same text, wich is the word “hello”, and over the years it was really hard to align all the different files that recognize the same text automatically.

To make it work, engineers created something , that processes in addition to a deep neural network.

FIRST STEP:

First thing is to give the machine the sound files. But this files files come in sound waves, so how do they turn into numbers or bits, so a computer can read it? Basically, sound waves are one-dimensional. At every moment in time, we can obtain a single value based on the height of the wave.

Here we have a file that is a sound clip of the word “Hello”:

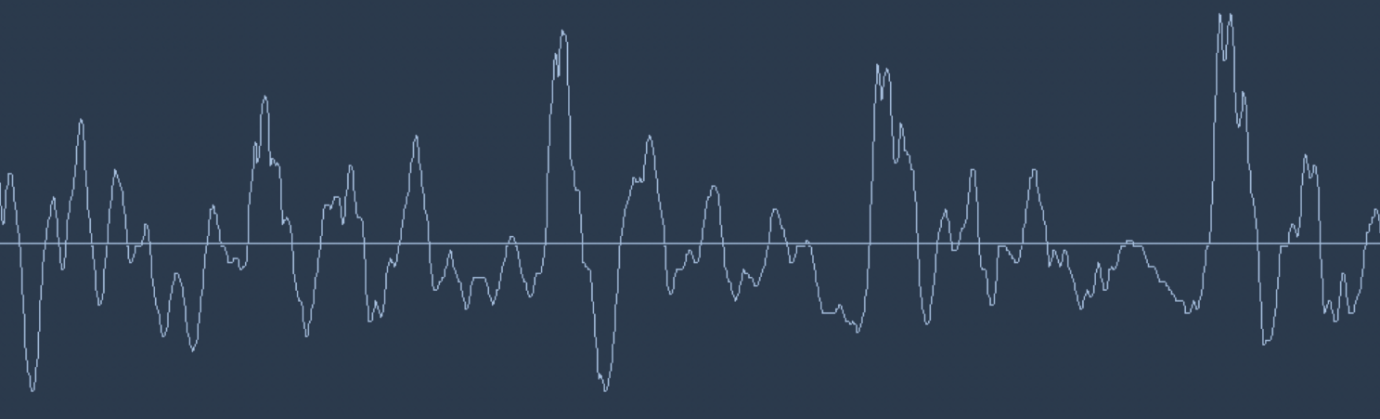


Figure1

Now we are going to zoom in a specific area, and then register the heigh of the wave in equally spaced points:

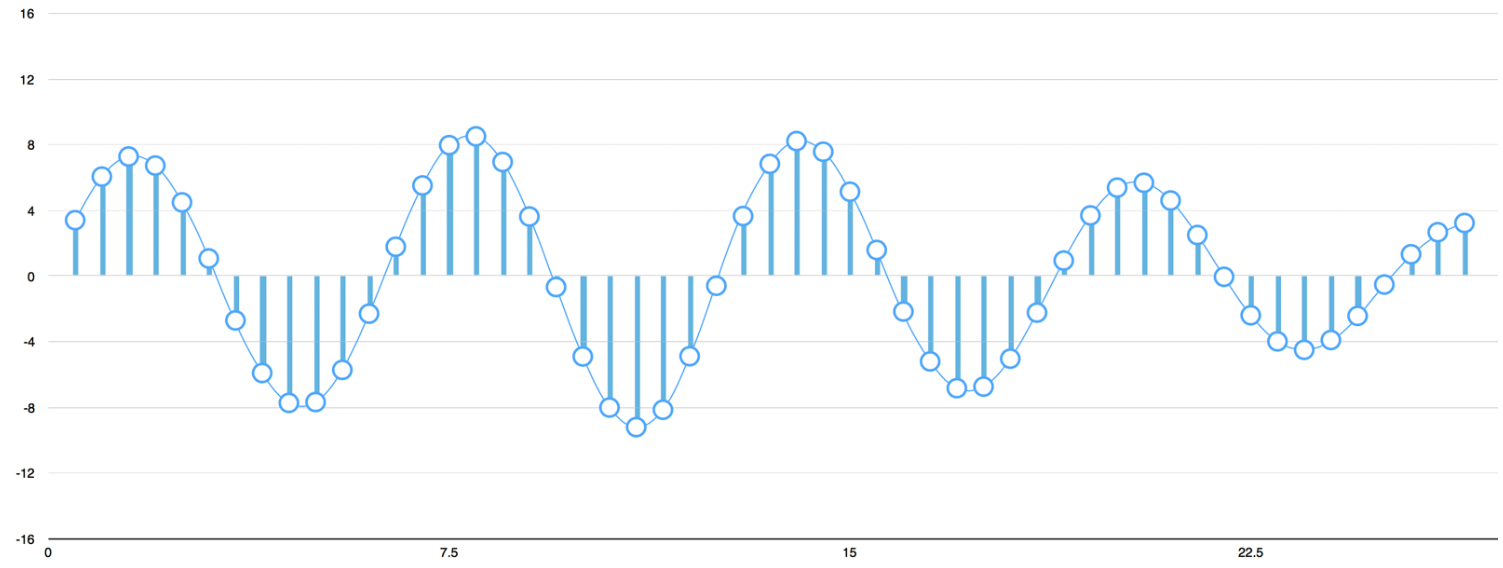


Figure2

What was done in Figure2 is called sampling. It consists in reading every millisecond and record a number that represents the heigh of the sound wave at that point in time.

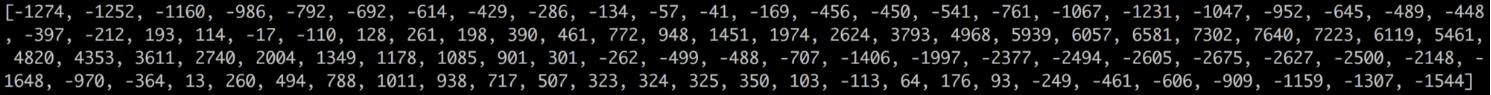
And the result we obtained for the word “Hello” for the first 100 samples on a 16,000 times per second was this one:

Figure3

After that, the engineers start to spoting a problem in sampling that was mencioned in Figure2 , wich was the fact that if they only were getting some points, and not the entire wave file, they were losing important data. But thanks to the Nyquist theorem, they were able to reconstruct the original sound wave from the gaps between the points.

SECOND STEP:

What we have now is an array of numbers, where each number represents the amplitude of the sound wave at 1/16,000th of a second intrevals.

This array could just go to the Neural Network, but trying to recognize voice patterns by processing these samples directly is difficult. So we need to pre-process our Sampled Sound Data. To start, we need to group our sampled audio into 20 milisecond long chunks.

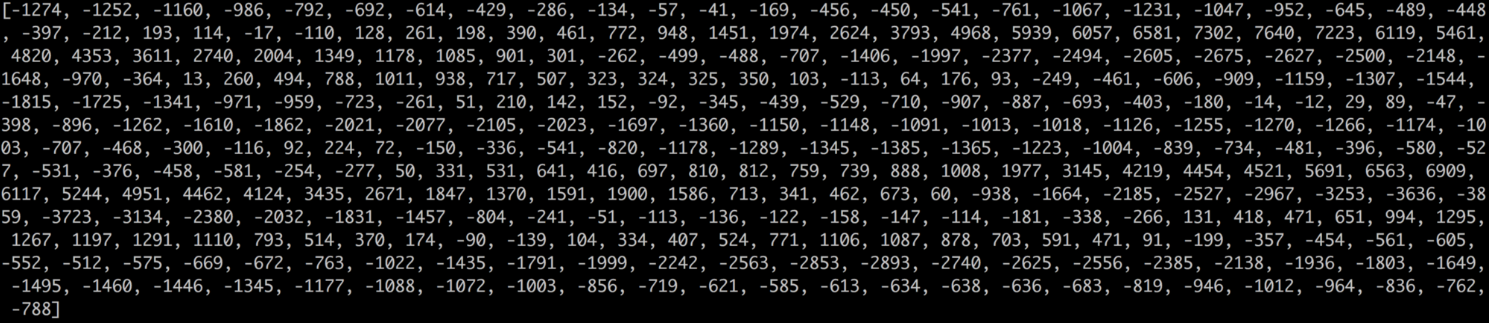
The first 20 miliseconds of audio looks like this:

Figure4

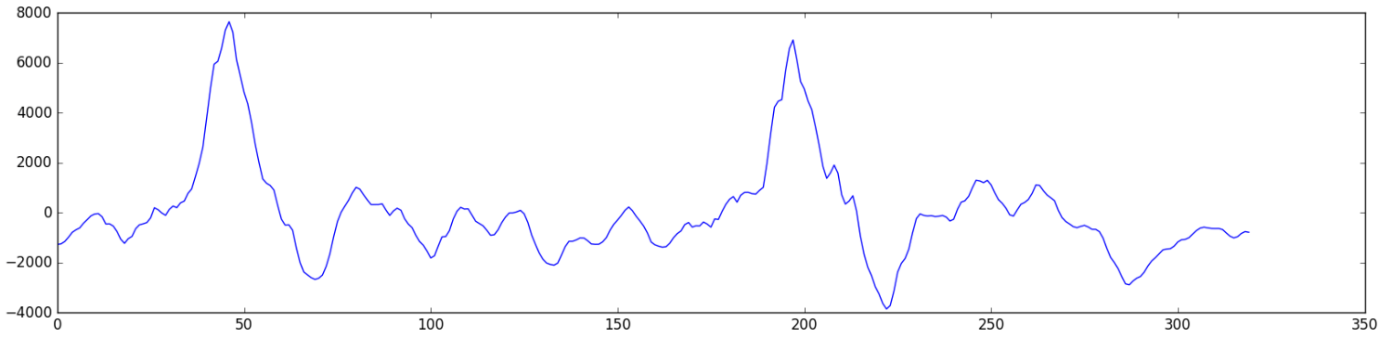
That’s the result of the first 320 samples, making a line graph and plotting those numbers should give us a closer approximation to the original sound file of the 20 milisecond period of time:

Figure5

This is only a 1/50th of a second long! Even this short recording is a complex mix of many frequencies of sound, we still can be able to identify some low sounds, mid-range sounds, and even some high-pitched sounds. Taken all together, we can obtain a complex human sound.

But the pre-processing its not over yet, we still need to make the data easier to the Neural Network be able to process it. Now, we will break this file into different component parts, separating it in different pitch categories: low, next to low, mid, next to high and high. Then we can create a fingerprint containing how much energy is in each of those frequency bands (from low to high).

That’s possible if we use the Fourier mathematic operation, it makes the breaks we need and after that we add how much energy is contained in each one.

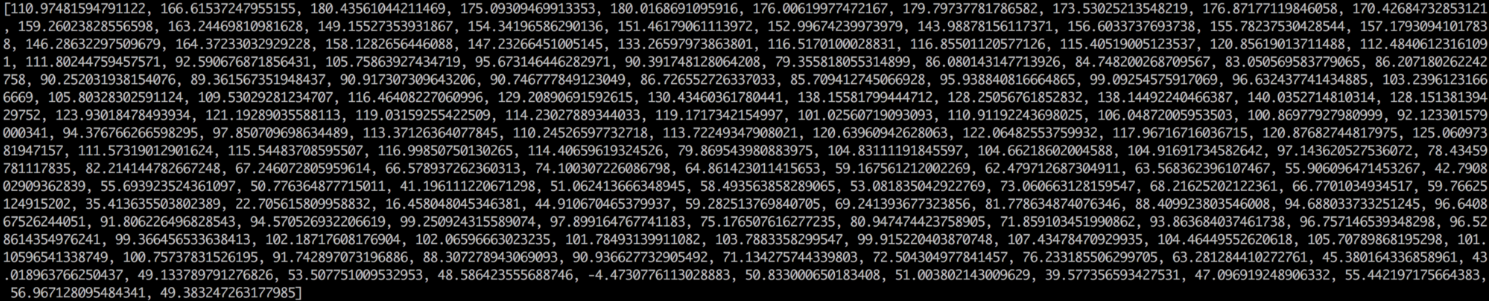
This result show us a score from low to high pitch and it’s energy (in the 50hz band) of our 20 milisecond sound file:

Figure6

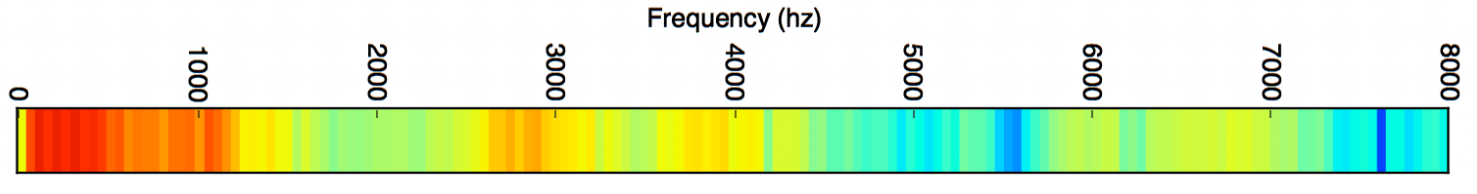
But what if we represented in chart like this, where we can see that we have a lot of energy (red colour) on the low frequencies and less energy (blue colour) in the high frequencies:

Figure7

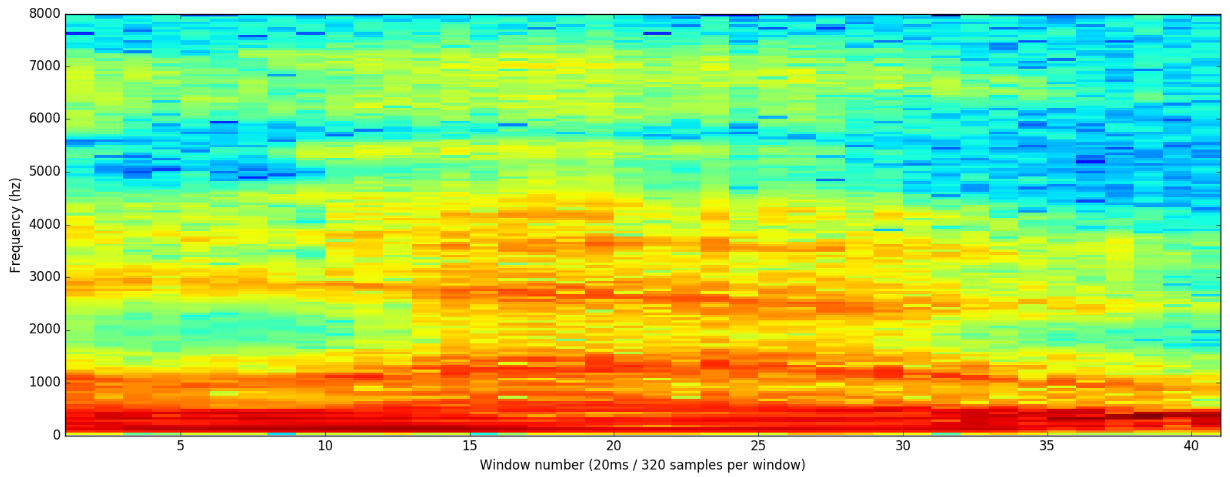
And then, repeat the same process for the entire 20 milissecond audio file we end up with a spectrogram:

Figure8

The spectrogram it’s a great visualization tool, because we can see the entire pitch patterns in the audio data. A Neural Netwirk can find patterns like this in the data more easily than raw sound waves. So we can use this data representation to input in the Neural Network.

THIRD STEP:

Now we obtained the exact format pretended, and it will be easier to be precessed. Let’s start by feeding our deep Neural Network. We will use as input the 20 milisecond audio chunks, for each audio slice, the model and the algorithm will try to figure out the letter that the sound currently spoken corresponds to.

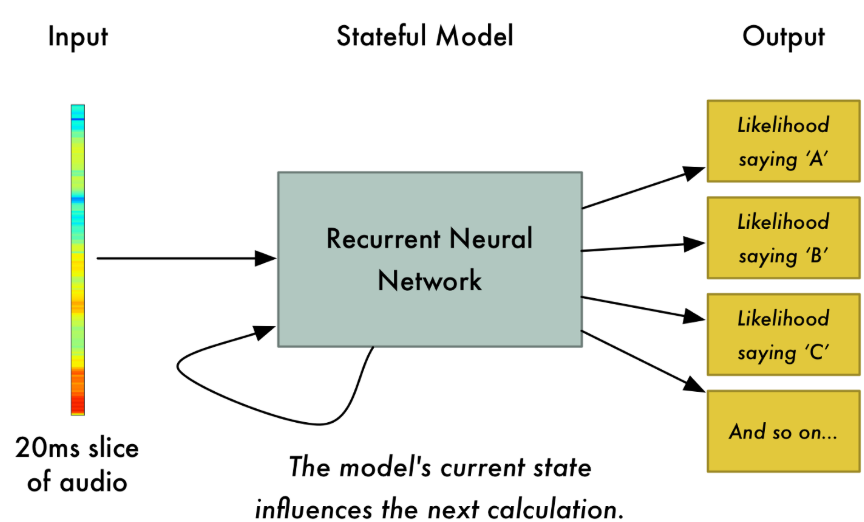


Figure9

The Neural Network that is being used, is a Recurrent Neural Network, it consists on having a memory that influences future predictions, where each of the letters that the model predicts, should affect the likelihood of the next letter that it will predict too. For example, a pearson speaking, can say “HEL” , than it will be very likely for the next letters be “LO” to finish the word “HELLO”. So having that memory will help the Neural Network to get better accuracy in next predictions.

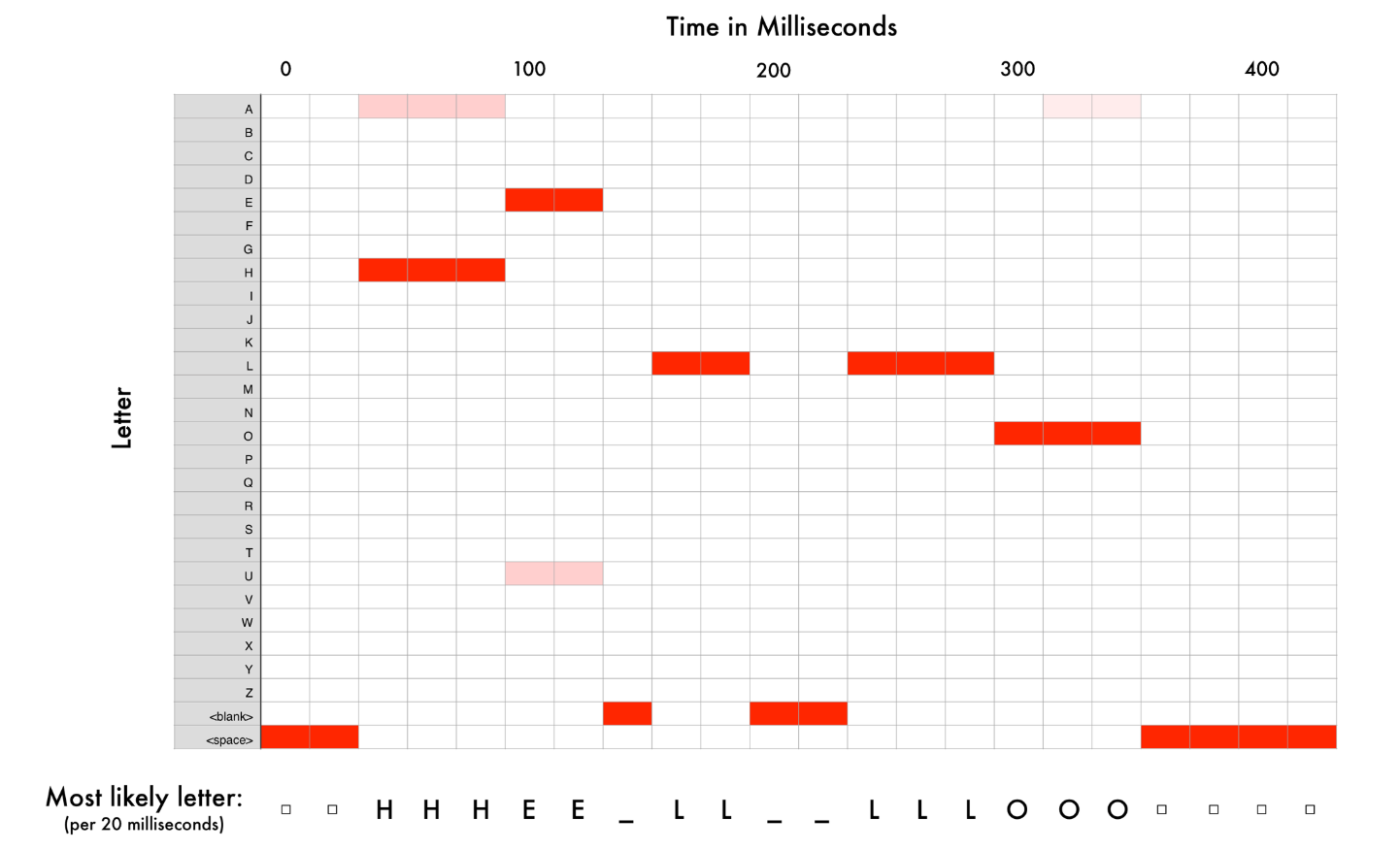
After we run the file through the Neural Network, one chunk at the time, we get a map of each audio part to the letters that most likely be spoken during that sound. This is the mapping of the word Hello, and it’s most likely letters:

Figure10

We can see that, when it was pronouncing the “H”, some gave “A” as the result, but many more “H’s” than “A’s”, the same happened in the “E” where the model read “U”. Overall it had a fairly good accuracy, but there were some repeated letters and some blank spaces, those were all removed.

That leaves us with 3 possible transcriptions/pronounciations: “Hello”, “Hullo” and “Aullo”. They are all very similar if you try to speak it. The Neural Network it’s predicting one letter at the time, so it will come up with some similar transcriptions.

The main trick here is to combine all the different pronounciations of the same word with the likelihood scores (exact word), based on a large database. That way we can take out the least likely results and keep the most possible ones. As it was with the word hello when you pronouciate it with “Hullo” or “Aullo”.

But what if someone really want to say “Hullo!”? Then the transcription will be incorrect. Imagine if there was an app called like that and you say: “Siri, open Hullo” , you really ment Hullo, not Hello. Well, in that case, Siri database of spoken words need to expand and to be trained more and more to recognize the new word, until then it is most likely to translate it to “Hello”.

# **Implementation**

Now that we know how the speech recognition works, it’s time to see it in action in a pyhton code. There are a lot of recognizers in the world, such as google, amazon and so on. Even python as his own speech recongnition library. We can build our code in two ways, to detect what an audio file is saying (like an .mp3 or .wav file), or even what our microphone can capture.

Let’s start with the audio file recognizer from google, with a file of a person saying “Hello”:

**import** speech\_recognition **as** sr

r = sr.Recognizer()

hello=sr.AudioFile('hello-1.wav')

**with** hello **as** source:

audio = r.record(source)

**try**:

s = r.recognize\_google(audio)

print( "Text: " +s)

**except** Exception as e:

print( "Exception: " +str(e))

Fisrt thing we import the Speech\_recognition library and callout the recognizer() function. Next we define our data, wich is the “hello-1.wav” file, and then we use our data as source to the recognizer in order to print the text that it heard. If everything goes right, we obtain a result like this:

Text: hello

If we wanted to build a code for a Voice detection program we have to make a few changes, by not giving data and defining the source to be from the microphone:

**import** speech\_recognition **as** sr

r = sr.Recognizer()

**with** sr.Microphone() **as** source:

audio = r.record(source)

**try**:

s = r.recognize (audio)

print( "You said: " +s)

**except** Exception as e:

print( "Exception: " +str(e))

But in this way seems to easy right? What if we could build our own speech recognizer and use our data to build a model using Deep Neural Network. It’s a model that will learn spoken numbers. So we will use a data set of people saying numbers, build a Neural Network, train our data and test it to see if we can recognize any other spoken numbers:

FISRT STEP:

**import** tflearn

**import** speech\_data

First we need to import TL learn, that is a high level library built on top of tensor flow that is easier to read and great for fast prototyping. The other import is refrent to a helper class created by me that we called speech data, that will help fetch data from the internet and make it in the right format for us tu use.

SECOND STEP:

learning\_rate = 0.0001

training\_iters = 300000

After that we define something we call hyperparameters or tuning ops. The learning rate is what we apply to this wait updating process, the greater the learning rate, the faster our network trains, the lower the learning rate, the more accurate our network predicts, basically it represents a trade-off between time and accuracy. Next we define how many we want to train for, selecting the amount of iteractions.

THIRD STEP:

batch = word\_batch = speech\_data.mfcc\_batch\_genertor(64)

X, Y = next(batch)

trainX, trainY = X, Y

testX, testY, X, Y

Now that we have our hyper parameters, it’s time to fetch our data, using the helper class speech\_data, that is specifically a batch generator function, that downloads a set of WAV files, each one of these files is a recording a different spoken digit, and every file has is own digit written in his label, if we want to return it for some studies. The labels will be returned as a batch, that we can split it into training and testing data.

FOURTH STEP:

net = tflearn.input\_data( [None,20,80])

net = tflearn.lstm(net, 128, dropout=0.8)

net = tflearn.fully\_connected(net, 10, activation=’softmax’)

net = tflearn. Regression(net, optimizer=’adam’, learning\_rate=learning\_rate, loss=’categorical\_crossentropy’)

It’s time to make our own model usig the Neural Network. How to know what is the best Neural Network model that we should use for this. Sincce spoken words are a sequence of sound waves, we want ot use a recurrent Neural Network, because they are capable of processing sequences. First we difine net with the TFLearn input\_data function, this will act like a gateway to the data get in the Network. It has two parameters: the width (that is the number of features that are extracted from our utterances from our speech\_data class) and the height (is the max lengh of each utterance.

The next line we use TFlearn LSTM (Long Short-term Memory function). This a recurrent net that can remember everything it’s fed, keeping all history of input data. In the parameters we have to chose the number of neurons, to few will lead to bad predictions and to many will over fit our training data, and the dropout value prevents overfitting, by randomly tuning off some neurons during training so data is forced to find new paths.

Another layer makes ir fully connected, meaning that every neuron will be connected to the previous layer’s neurons, the number of classes are 10, since we only are recognizing 10 digits and the activation to “softmax” will convert numerical data into probabilities.

Lastely we create our output layer as a regression wich will output a single predicted number for our utterance. Using “adam” optimizer it will minimize the categorical croos entropy loss over time in order to get a more accurate prediction.

FITH STEP:

model = tf.learn.DNN(net, tensorboard\_verbose=0)

**while** 1:

model.fit(trainX, trainY, n\_epoch=10, validation\_set=(testX, testY), show\_metric=True, batch\_size=64)

\_y=model.predict(X)

model.save( ‘tflearn.lstm.model’ )

print(\_y)

print(y)

Now it’s ready to run the Deep Neural Network, we fit our model, save it for a later use that we might want to do with different data and print the result, that should be a single number.

# **Discussion on the Findings**

To break it down LSTM Neural Networks are use in state-of-the-art speech recognition. We can use TFLearn to quickly build and train a Deep Neural Network to recognize speech and good hyper parameters like the learning rate are those that are balanced between trade-offs like time and accuracy.

During the research, I made a lot of additional installs and other things so I can run the code, it will be in references some of the tools I needed to make research on, so it would make possible to get my results.

The accuracy I got in this last implementation was 0.91, witch is pretty good. It comes with the fact that I fed the model with a lot of data and as I said before it’s key to give more data and computing power in this type of models.

# **Conclusion**

Voice Detetion and Speech Recognition are invading our lives, it has been around for decades, but now with all the known devices that are out there it has become even more popular.

For Speech Recognition we had to overcome limitless challenges like, bad quality microphones, background noise, reverb and echo, different accent variations and so on. All this issues must be in the data of the Neural Network too, so we can deal with them.

It has so many applications like In-car systems, health care, in the military, where they devolop high performance aircrafts, education and to help people with disabilities.

Some security concerns have been related too. It can happen some accidental operations, like if the recognition device is active and starts to listen input innappropriatly, it can give some information away to someone we don’t want to. Then we start to get some advertising of products we were just talking to our familily or friends.

It’s a very powerfull tool overall, but we still need to be careful, because like any other machine learning device, it’s a machine and it needs to be learned how to fuction with it.

# **References**

Zhang, A. (2015). Speech Recognition (Version 2.1) [Software]. Available from https://github.com/Uberi/speech\_recognition#readme.

IISourceII (2016). Tensorflow speech recognition. Available from https://github.com/llSourcell/tensorflow\_speech\_recognition\_demo

Adam Geitgey (2016). Machine Learning is fun Part 6. Available from https://medium.com/@ageitgey/machine-learning-is-fun-part-6-how-to-do-speech-recognition-with-deep-learning-28293c162f7a

David Amos (2017). The Ultimate Guide To Speech Recognition With Python. Available from https://realpython.com/python-speech-recognition/

Wikipedia (2020). Speech recognition. Available from https://en.wikipedia.org/wiki/Speech\_recognition

Wikipedia (2020). Voice activity detection. Available from https://en.wikipedia.org/wiki/Voice\_activity\_detection

Pypi (2017).Speech Recognition 3.8.1. Available from https://pypi.org/project/SpeechRecognition/