Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 06, June-2020

Trigger Word Recognition using LSTM

Kalyanam Supriya, Anemu Divya, Balaga Vinodkumar, Gedala Ram Sai
Department of Computer Science and Engineering
Aditya Institute of Technology and Management
Tekkali, Andhra Pradesh, India

***Abstract*— A Trigger word is a word that you use to wake up a virtual voice assistant, for example "Hey Siri" or "Hey Alexa". First, we need a good labelled training data to train our model. We could record 10 seconds audio clip of people saying positive ("activate" in this case) and negative (words that are not "activate") examples and label manually when the trigger words were spoken by the people. Labelling the data manually is complex and time consuming. Instead, the training data is generated artificially. We would then need 3 types of audio clips:**

1. Positive examples of people saying the word "activate", 1 or 2 seconds each

2. Negative examples of people saying random words, 1 or 2 seconds each

3. Background noise, for example coffee shop or office, 10 seconds each

The training data that we have generated need to be pre-processed before it is sent to a machine learning model. Due to the variation of air pressure sound can be produced. The input data to the model is the spectrogram for each generated audio due to which the target will be the labels created earlier. In recent years, Deep Learning (DL) has occupied increasing attention within the industry and academic world for its high performance in various domains. Today, Recurrent Neural Network (RNN) and Convolutional Neural Network (CNN) are the most popular forms of DL architectures used. We are doing Trigger Word Recognition on speech data by using Long Short-Term Memory (LSTM).

Keywords*— *Trigger word,virtual voice assistant, Positive examples, . Negative examples, Background noise

I. INTRODUCTION

Trigger word detection is the technology that allows devices like Amazon Alexa, Google Home, Apple Siri, and Baidu DuerOS to wake up upon hearing a particular word. In our implementation, the trigger word is "Activate." As it hears you say every time "Activate," it will give a "chiming" sound. You are able to record a clip of yourself talking, and have the algorithm to trigger a chime sound when it recognises you saying "Activate". Although we can also extend it to run on your laptop so that every time you say "Activate" it opens up your favourite app, or turns on a network connected lamp in your house, or triggers any other event. Data synthesis is an effective way to create a large training set for speech problems, specifically trigger word detection. Using a spectrogram and a 1D conv layer is a common pre-processing step need to passing audio data to an RNN, GRU or LSTM. A deep learning model can be used to build a very effective trigger word detection system if it follows an end-to-end approach. We know that the Deep Learning model prediction takes time because it processes the audio data asynchronous from the input audio streaming. An effective way to reduce the delay is a sliding/moving input window.

Speech as a computer interface has numerous benefits over traditional interfaces such as a GUI with mouse and keyboard: speech is natural boon for humans as it requires no special training, and it improves multitasking by leaving the eyes and hands free, and is often more faster and more efficient to transmit than using other conventional input methods. Even many of us know that many tasks are solved with traditional interfaces such as visual, pointing interfaces, and/or keyboards, but also speech has the potential to be a better interface for a number of tasks where full natural language communication is useful and the recognition performance of the Speech Recognition (SR) system is sufficient to perform the tasks accurately. This includes the circumstances where the user tries to multitask while hands-busy or eyes-busy applications such as where the user has objects to manipulate or equipment/devices to control while the user asking assistance from the computer.

A development set is the data that you would use to optimize your model against during the development process. A machine learning process has a training (or train) set, which is the data you train the model with, and a test set, which you use to determine the performance of the model. The development dataset consists of 1500 audio files divided into the five categories, each containing 300 files. The number of different sound types within each category is not balanced.

Why Trigger Word Work So Good?

The Trigger Word also called Hot Word, is a brick, a module, of speech recognition in the global sense of the term. It is rather a word or a series of words which will be used to trigger the recording of the user's voice of a speech recognition system. If In any case we do a retrospective of Google news, we realize that their assistants like to listen a little more than what is planned that is to read. This "Keyword" system used to activate the system for two reasons:

Respect the privacy of users so that only the sequence that will follow the Wake-up-word, i.e. the one that corresponds to the intention, is recorded.

Reduce system consumption and performance as constant recording and analysis of audio files is a very heavy task if performed continuously.

II.RELATED WORK

The study focused on the speech recognition systems where speech plays an important role in humans. speech is a good interface than any traditional interfaces like any other graphical user interfaces. since people in digital era wants to do their work in efficient way by doing multitasking. This includes the circumstances where the user tries to multitask while hands-busy or eyes-busy applications such as where the user has objects to manipulate or equipment/devices to control while the user asking assistance from the computer. So, keeping these all in my mind we developed a trigger word system in which the trigger word is "Activate." As it hears you say every time "Activate," it will give a "chiming" sound. You are able to record a clip of yourself talking, and have the algorithm to trigger a chime sound when it recognises you saying "Activate”. Although we can also extend it to run on your laptop so that every time you say "Activate" it opens up your favourite app, or turns on a network connected lamp in your house, or triggers any other event. Data synthesis is an effective way to create a large training set for speech problems, specifically trigger word detection.

The main aim is to detect the word “activate” as our trigger word. Trigger word detection works by identifying the trigger word when anyone says before sending it to a machine learning model based on user recordings by listening to a stream of audio and by extracting features.

Automatic Speech recognition (ASR) is the ability of a computer to convert a speech audio signal into its textual transcription. Some motivations for building ASR systems are, presented in order of difficulty, to improve human–computer interaction through spoken language interfaces, to solve difficult problems such as speech to speech translation, and to build intelligent systems that can process spoken language as proficient as humans.

The WUW speech recognition task is similar to Key-Word spotting. However, WUW-SR is different in one important aspect of being able to discriminate the specific word/phrase used only in alerting context (and not in the other; e.g. referential contexts). Specifically, the sentence ‘‘Computer, begin PowerPoint presentation’’ exemplifies the use of the word ‘‘Computer’’ in alerting context. On the other hand, in the sentence ‘‘My computer has dual Intel 64-bit processors each with quad cores’’ the word computer is used in referential (non-alerting) context. Traditional key-word spotters will not be able to discriminate between the two cases. The discrimination will be only possible by deploying higher level natural language processing subsystem in order to discriminate between the two. However, for applications deploying such solutions is very difficult (practically impossible) to determine in real time if the user is speaking to the computer or about the computer.

Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. It can not only process single data points (such as images), but also entire sequences of data (such as speech or video). For example, LSTM is applicable to tasks such as unsegmented, connected handwriting recognition, speech recognition and anomaly detection in network traffic or IDS's (intrusion detection systems).

LSTM networks are well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series. LSTMs were developed to deal with the exploding and vanishing gradient problems that can be encountered when training traditional RNNs. Relative insensitivity to gap length is an advantage of LSTM over RNNs, hidden Markov models and other sequence learning methods in numerous applications. There are several architectures of LSTM units. A common architecture is composed of a cell (the memory part of the LSTM unit) and three "regulators", usually called gates, of the flow of information inside the LSTM unit: an input gate, an output gate and a forget gate. Some variations of the LSTM unit do not have one or more of these gates or maybe have other gates. For example, gated recurrent units (GRUs) do not have an output gate.

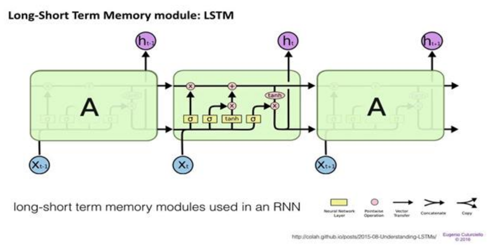

Long-Short Term Memory module: LSTM
h_{t-1}
h_t
h_{t+1}
A
tanh
A
x_{t-1}
x_t
x_{t+1}
long-short term memory modules used in an RNN

Fig-a Long-Short Term Memory Architecture

III LITERATURE SURVEY

Fengpei Ge and Yonghong Yan have proposed an Intelligent model for voice trigger algorithm utilizing Deep Neural Network. This System works with the objective function of state-level minimum Bayes risk for training, customizing the decoding network to absorb the ambient noise and background speech. In this System We adopt a two-stage classification strategy to integrate the phonetic knowledge and model-based classification into detecting wake-up words [1]. Gautam Tiwari and Arindam Mandal have implemented a system direct modelling of raw audio with DNN for wake word detection. This work builds on the Conventional speech recognition systems typically extract a compact feature representation based on prior knowledge such as Log-Mel filter bank energy (LFBE). Such a feature is then used for training a deep neural network (DNN) acoustic model (AM) [2]. Ralf Schluter, Hermann Ney have jointly proposed a system in which it first focus on the evaluation of each of the classical gated architectures for language modeling for large vocabulary speech recognition. Namely, it evaluates the highway network, lateral network, LSTM and GRU. It found that the highway connections enable both standalone feedforward and recurrent neural language models to benefit better from the deep structure and provide a slight improvement of recognition accuracy after interpolation with count models [3]. Todor Ganchev and Ilyas Potamitis presented an integrated system that uses speech as a natural input modality to provide user-friendly access to information and entertainment devices installed in a real home environment. It worked on the implementation of the front-end signal pre-processing block that consist of an array of 8 microphones connected to a multi-channel soundcard and of workstations performing all signal pre-processing tasks [4]. Igor Stefanović, Eleonora Nan jointly worked two implementations of the wake word detection module, based on Pocketsphinx and Snowboy voice recognition engines. Experiments have shown that Pocketsphinx has better accuracy than Snowboy, but its performance is limited on RPI2 [5]. Shilpa Srivastava, Ashutosh Kumar Singh, Sanjay Kumar Nayak presented a speech command module enables the voice control in the home lab. The main intent was to build the lightweight unit that can operate on the low power embedded devices such as Raspberry

Pi [6]. In past researches it is clear that this model is successful with diverse recognition of word sources but it lacks a bit of accuracy in the case of long recordings. Many proposed models are not so successful in correctly classifying the long speech recordings. On the other hand, models incorporating LSTM networks are showing impressive results which is because its capability of dealing the long speech data.

IV DATA SYNTYHESIS

A.Listening to the data: Creating a speech dataset

Let's start by building a dataset for trigger word detection algorithm. A speech dataset should ideally be as close as possible to the application you will want to run it on. In this case, you'd like to detect the word "activate" in working environments (library, home, offices, open-spaces ...). You thus need to create recordings with a mix of positive words ("activate") and negative words (random words other than activate) on different background sounds.

.Listening to the data

One of our friends is helping you out on this project, and they've gone to libraries, cafes, restaurants, homes and offices all around the region to record background noises, as well as snippets of audio of people saying positive/negative words. This dataset includes people speaking in a variety of accents.

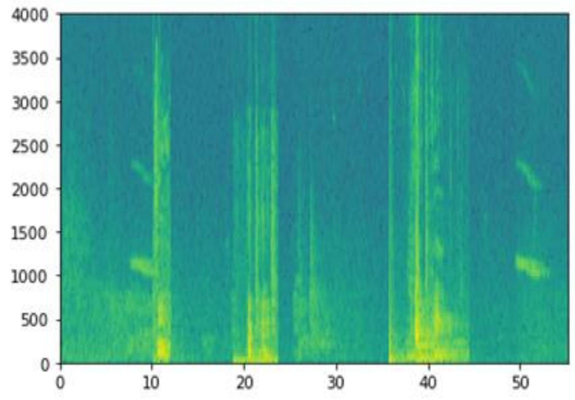
In the raw_data directory, you can find a subset of the raw audio files of the positive words, negative words, and background noise. You will use these audio files to synthesize a dataset to train the model. The "activate" directory contains positive examples of people saying the word "activate". The "negatives" directory contains negative examples of people saying random words other than "activate". There is one word per audio recording. The "backgrounds" directory contains 10 second clips of background noise in different environments.

Some Examples:
IPython.display. Audio("./raw_data/activates/1.wav")
IPython.display. Audio("./raw_data/negatives/4.wav")
IPython.display. Audio("./raw_data/backgrounds/1.wav")
We will use these three types of recordings (positives/negatives/backgrounds) to create a labelled dataset.

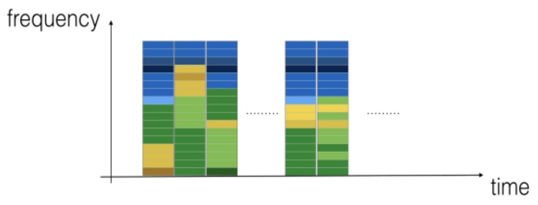
B. From audio recordings to spectrograms:

What really is an audio recording? A microphone records little variation in air pressure over time, and it is these little variations in air pressure that your ear also perceives as sound. You can think of an audio recording is a long list of numbers measuring the little air pressure changes detected by the microphone. We will use audio sampled at 44100 Hz (or 44100 Hertz). This means the microphone gives us 44100 numbers per second. Thus, a 10 second audio clip is represented by 441000 numbers (=10*44100). It is quite difficult to figure out from this "raw" representation of audio whether the word "activate" was said. In order to help your sequence model more easily learn to detect trigger words, we will compute a *spectrogram* of the audio. The spectrogram tells us how much different frequencies are present in an audio clip at a moment in time.

Let's see an example.
IPython.display.
Audio("audio_examples/example_train.wav")
x =
graph_spectrogram("audio_examples/example_train.wav")



The graph above represents how active each frequency is (y axis) over a number of time-steps (x axis).



The dimension of the output spectrogram depends upon the hyperparameters of the spectrogram software and the length of the input. In this, we will be working with 10 second audio clips as the "standard length" for our training examples.

The number of timesteps of the spectrogram will be 5511. we see later that the spectrogram will be the input x into the network, and so Tx=5511.

Now you can define,

Tx = 5511 # The number of time steps input to the model from the spectrogram
n_freq = 101 # Number of frequencies input to the model at each time step of the spectrogram
Note that even with 10 seconds being our default training example length, 10 seconds of time can be discretized to different numbers of value. You've seen 441000 (raw audio) and 5511 (spectrogram). In the former case, each step represents 10/441000 ~ 0.000023 seconds. In the second case, each step 10/5511 ~ 0.0018 represents seconds.

For the 10sec of audio, the key values you will see in this assignment are:

- 441000 (raw audio)
- 5511 = Tx (spectrogram output, and dimension of input to the neural network)
- 10000 (used by the pydub module to synthesize audio)

- 1375 = Ty (the number of steps in the output of the GRU you'll build)

Note that each of these representations correspond to exactly 10 seconds of time. It's just that they are discretizing them to different degrees. All of these are hyperparameters and can be changed (except the 441000, which is a function of the microphone). We have chosen values that are within the standard ranges used for speech systems.

Consider the Ty = 1375 number above. This means that for the output of the model, we discretize the 10s into 1375 time-intervals (each one of length 10/1375 ~ 0.0072s) and try to predict for each of these intervals whether someone recently finished saying "activate."

Consider also the 10000 number above. This corresponds to discretizing the 10sec clip into 10/10000 = 0.001 second intervals. 0.001 seconds is also called 1 millisecond, or 1ms. So when we say we are discretizing according to 1ms intervals, it means we are using 10,000 steps.

Ty = 1375 # The number of time steps in the output of our model

C Generating a single training example:

Because speech data is hard to acquire and label, you will synthesize your training data using the audio clips of activates, negatives, and backgrounds. It is quite slow to record lots of 10 second audio clips with random "activates" in it. Instead, it is easier to record lots of positives and negative words, and record background noise separately (or download background noise from free online sources).

To synthesize a single training example, you will:

- Pick a random 10 second background audio clip.
- Randomly insert 0-4 audio clips of "activate" into this 10sec clip.
- Randomly insert 0-2 audio clips of negative words into this 10sec clip.

Because we had synthesized the word "activate" into the background clip, you know exactly when in the 10sec clip the "activate" makes its appearance. You'll see later that this makes easier to generate the labels as well.

We will use the pydub package to manipulate audio. Pydub converts raw audio files into lists of Pydub data structures (it is not important to know the details here). Pydub uses 1ms as the discretization interval (1ms is 1 millisecond = 1/1000 seconds) which is why a 10sec clip is always represented using 10,000 steps.

Overlaying positive/negative words on the background:

Given a 10sec background clip and a short audio clip (positive or negative word), you need to be able to "add" or "insert" the word's short audio clip onto the background. To ensure audio segments inserted onto the background do not overlap, you will keep track of the times of previously inserted audio clips. We will be inserting multiple clips of positive/negative words onto the background, and We don't want to insert an "activate" or a random word somewhere that overlaps with another clip you had previously added.

For clarity, when you insert a 1sec "activate" onto a 10sec clip of cafe noise, you end up with a 10sec clip that sounds like someone saying "activate" in a cafe, with "activate" superimposed on the background cafe noise. You do not end up with an 11 sec clip.

Creating the labels at the same time you overlay

Recall also that the labels represent whether or not someone has just finished saying "activate." Given a background clip, we can initialize = 0 for al tl, since the clip doesn't contain any "activates." When we insert or overlay an "activate" clip, we will also update labels for , so that 50 steps of the output now have target label 1. We will train a GRU to detect when someone has finished saying "activate". For example, suppose the synthesized "activate" clip ends at the 5sec mark in the 10sec audio exactly halfway into the clip. Recall that Ty = 1375, so timestep 687= int (1375*0.5) corresponds to the moment at 5sec into the audio. So, we will set. Further, we would quite satisfied if the GRU detects "activate" anywhere within a short time-internal after this moment, so we actually set 50 consecutive values of the label to 1. Specifically, we have .

This is another reason for synthesizing the training data: It's relatively straightforward to generate these labels as described above. In contrast, if you have 10sec of audio recorded on a microphone, it's quite time consuming for a person to listen to it and mark manually exactly when "activate" finished. Here's a figure illustrating the labels , for a clip which we have inserted "activate", "innocent", activate", "baby." Note that the positive labels "1" are associated only with the positive words.

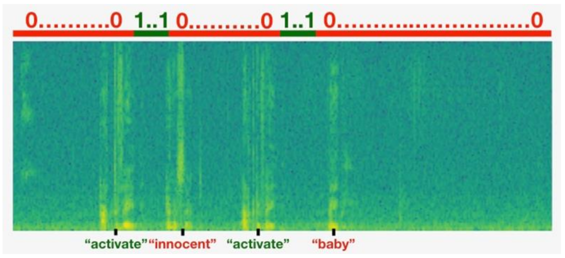


Fig -b Illustrating Positive Words

To implement the training set synthesis process, you will use the following helper functions. All of these functions will use a 1ms discretization interval, so the 10sec of audio is always discretized into 10,000 steps.

- get_random_time_segment(segment_ms) gets a random time segment in our background audio.

- is_overlapping (segment_time, existing_segments) checks if a time segment overlaps with existing segments.

● insert_audio_clip (background, audio_clip, existing_times) inserts an audio segment at a random time in our background audio using get_random_time_segment and is_overlapping.

● insert_ones (y, segment_end_ms) inserts 1's into our label vector y after the word "activate".

Thefunction get_random_time_segment(segment_ms) returns a random time segment onto which we can insert an audio clip of duration segment_ms.

def get_random_time_segment(segment_ms):

Gets a random time segment of duration segment_ms in a 10,000 ms audio clip.

Arguments:

segment_ms -- the duration of the audio clip in ms ("ms" stands for "milliseconds")

Returns:

segment_time -- a tuple of (segment_start, segment_end) in ms

segment_start = np. random. randint (low=0, high=10000-segment_ms) # Make sure segment doesn't run past the 10sec background

segment_end = segment_start + segment_ms - 1

return (segment_start, segment_end)

Next, suppose you have inserted audio clips at segments (1000,1800) and (3400,4500). I.e., the first segment starts at step 1000, and ends at step 1800. Now, if we are considering inserting a new audio clip at (3000,3600) does this overlap with one of the previously inserted segments? In this case, (3000,3600) and (3400,4500) overlap, so we should decide against inserting a clip here. For the purpose of this function, define (100,200) and (200,250) to be overlapping, since they overlap at timestep 200. However, (100,199) and (200,250) are non-overlapping.

Implement is_overlapping (segment_time, existing_segments) to check if a new time segment overlaps with any of the previous segments. We should follow 2 steps:

1. Create a "False" flag, that you will later set to "True" if you find that there is an overlap.

2. Loop over the previous_segments' start and end times. Compare these times to the segment's start and end times. If there is an overlap, set the flag defined in (1) as True. You can use:

```
    for ....:

            if ... <= ... and ... >= ...

                ...
```

Hint: There is overlap if the segment starts before the previous segment ends, and the segment ends after the previous segment starts.

def is_overlapping (segment_time, previous_segments):

Checks if the time of a segment overlaps with the times of existing segments.

Arguments:

segment_time -- a tuple of (segment_start, segment_end) for the new segment

previous_segments -- a list of tuples of (segment_start, segment_end) for the existing

segments

Returns:

True if the time segment overlaps with any of the existing segments, False otherwise

segment_start, segment_end = segment_time

Now, we use the previous helper functions to insert a new audio clip onto the 10sec background at a random time, but making sure that any newly inserted segment doesn't overlap with the previous segments.

Implement insert_audio_clip () to overlay an audio clip onto the background 10sec clip. You will need to carry out 4 steps:

1.Get a random time segment of the right duration in ms.

2.Make sure that the time segment does not overlap with any of the previous time segments. If it is overlapping, then go back to step 1 and pick a new time segment.

3.Add the new time segment to the list of existing time segments, so as to keep track of all the segments you've inserted.

4.Overlay the audio clip over the background using pydub. We have implemented this for you.

Finally, implement code to update the labels, assuming as just inserted an "activate." In the code below, y is a (1,1375) dimensional vector. If the "activate" ended at time step t, then set as well as for up to 49 additional consecutive values. However, make sure you don't run off the end of the array and try to update y[0] [1375], since the valid indices are y[0][0] through y[0][1374] because Ty=1375. So if "activate" ends at

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 06, June-2020

step 1370, you would get only y[0][1371] = y[0][1372] = y[0][1373] = y[0][1374] = 1.

Implement create_training_example (). You will need to carry out the following steps:

1.Initialize the label vector ‘y’ as a NumPy array of zeros and shape (1, Ty).

2.Initialize the set of existing segments to an empty list.

3.Randomly select 0 to 4 "activate" audio clips, and insert them onto the 10sec clip. Also insert labels at the correct position in the label vector y.

4.Randomly select 0 to 2 negative audio clips, and insert them into the 10sec clip.

D.Full training set

Till now we implemented the code needed to generate a single training example. We used this process to generate a large training set. To save time, we've already generated a set of training examples.

# Load pre-processed training examples
X = np. load("./XY_train/X.npy")
Y = np. load("./XY_train/Y.npy")

E. Development set

To test our model, we recorded a development set of 25 examples. While our training data is synthesized, we want to create a development set using the same distribution as the real inputs. Thus, we recorded 25 10-second audio clips of people saying "activate" and other random words, and labeled them by hand. This follows the principle that we should create the dev set to be as similar as possible to the test set distribution; that's why our dev set uses real rather than synthesized audio.

# Load pre-processed dev set examples

X_dev = np. load("./XY_dev/X_dev.npy")

Y_dev = np. load("./XY_dev/Y_dev.npy")

V.IMPLEMENTATION

A.Fit the model:
Trigger word detection takes a long time to train. To save time, we've already trained a model for about 3 hours on a GPU using the architecture, and a large training set of about 4000 examples. Let's load the model.
model=load_model(‘/models/tr_model.h5’). You can train the model further, using the Adam optimizer and binary cross entropy loss, as follows. This will run quickly because we are training just for one epoch and with a small training set of 26 examples.
opt=Adam (lr=0.0001, beta_1=0.9, beta_2=0. 999.decay=0.01)
model. compile (loss=’binary_crossentrophy’, optimizer=opt, metrics=[“accuracy”])
model. Fit (X, Y, batch_size=5, epochs=1)

B.Architecture:
The 1D convolutional step inputs 5511 timesteps of the spectrogram (10 seconds), outputs a 1375 step output. It extracts low-level audio features similar to how 2D convolutions extract image features. Also helps speed up the model by reducing the number of timesteps.
The two GRU layers read the sequence of inputs from left to right, then ultimately uses a dense+sigmoid layer to make a prediction. Sigmoid makes the range of each label between 0~1. Being 1, corresponding to the user having just said "activate".

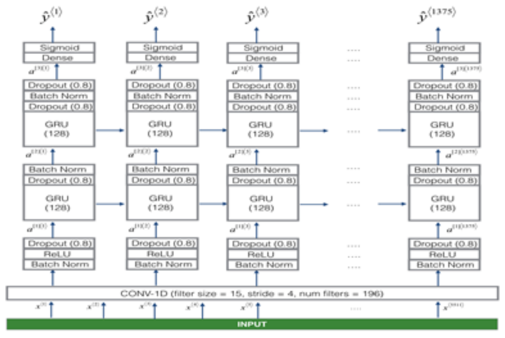


Fig-c Working of Conv-1D Layer

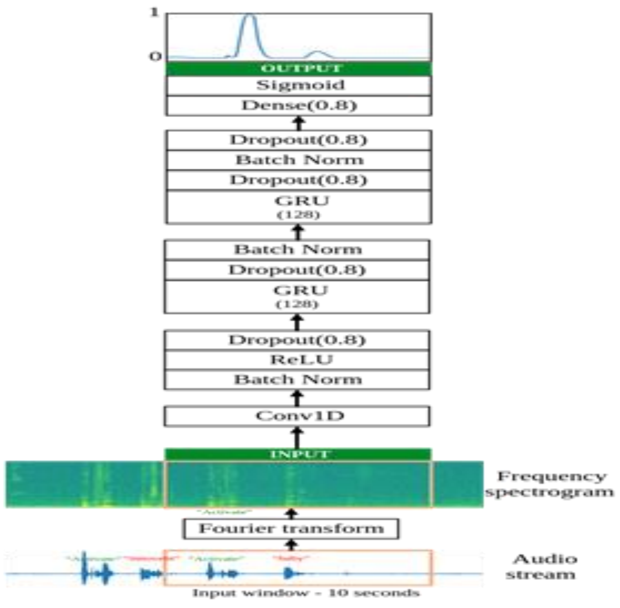


Fig-dArchitecture (Methodology)

Algorithm

```
def model(input_shape):
    Function creating the model's graph in Keras.
    Argument:
    input_shape -- shape of the model's input data (using Keras conventions)
    Returns:
    model -- Keras model instance
   X_input = Input (shape = input_shape)

    # Step 1: CONV layer
```

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 06, June-2020

```
    X = Conv1D (196, kernel_size=15, strides=4) (X_ input) # CONV1D
    X = BatchNormalization () (X)   # Batch normalization
    X = Activation('relu') (X)   # ReLu activation
    X = Dropout (0.8) (X)   # dropout (use 0.8)
    # Step 2: First GRU Layer
    X = GRU (units = 128, return_sequences = True) (X) # GRU (use 128 units and return the sequences)
    X = Dropout (0.8) (X)    # dropout (use 0.8)
    X = BatchNormalization () (X)    # Batch normalization
    # Step 3: Second GRU Layer
    X = GRU (units = 128, return_sequences = True) (X) # GRU (use 128 units and return the sequences)
    X = Dropout (0.8) (X)    # dropout (use 0.8)
    X = BatchNormalization () (X)   # Batch normalization
    X = Dropout (0.8) (X)   # dropout (use 0.8)
    # Step 4: Time-distributed dense layer
    X = TimeDistributed (Dense (1, activation = "sigmoid")) (X) # time distributed (sigmoid)
    model = Model (inputs = X_input, outputs = X)

    return model
```

Instead of using this whole code to train, there is already trained a model on a GPU using the architecture, which has a large training set of about 4000 examples.

model= load. model (‘. /models/tr_model.h5’).

C.Predictions

Now that you have built a working model for trigger word detection, let's use it to make predictions. This code snippet runs audio (saved in a wav file) through the network [7]. Once you've estimated the probability of having detected the word "activate" at each output step, you can trigger a "chiming" sound to play when the probability is above a certain threshold. Further, might be near 1 for many values in a row after "activate" is said, yet we want to chime only once. So, we will insert a chime sound at most once every 75 output steps. This will help prevent us from inserting two chimes for a single instance of "activate". (This plays a role similar to non-max suppression from computer vision.)

```
chime_file = "audio_examples/chime.wav"

def chime_on_activate (filename, predictions, threshold):

   audio_clip = AudioSegment.from_wav(filename)

   chime = AudioSegment.from_wav(chime_file)

   Ty = predictions. shape [1]

   # Step 1: Initialize the number of consecutive output steps to 0

   consecutive_timesteps = 0

   # Step 2: Loop over the output steps in the y

   for i in range (Ty):

     # Step 3: Increment consecutive output steps

       consecutive_timesteps += 1

     # Step 4: If prediction is higher than the threshold and more than 75 consecutive output steps have passed

       if predictions [0, i,0] > threshold and consecutive_timesteps > 75:

      # Step 5: Superpose audio and background using pydub

          audio_clip = audio_clip. overlay (chime, position = ((i / Ty) *   audio_clip. duration_seconds) *1000)

          # Step 6: Reset consecutive output steps to 0

          consecutive_timesteps = 0

     audio_clip. export ("chime_output.wav", format='wav')
```

D.Test on samples

Let's explore how model performs on three audio clips from the set. Let’s first listen to the three set clips.

```
IPython.display. Audio("./raw_data/activates/1.wav")

IPython.display. Audio("./raw_data/backgrounds/1.wav")

IPython.display. Audio("./raw_data/negatives/4_0.wav")
```

Now let’s run the model on these audio clips and see if it adds a chime after "activate"!

```
filename = "./raw_data/activates/1.wav"

prediction = detect_triggerword(filename)

chime_on_activate (filename, prediction, 0.5)

IPython.display. Audio("./chime_output.wav")
```

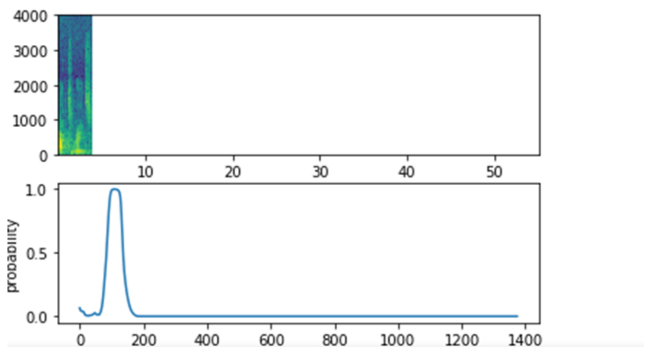


Fig-e Examples of “activate” that are detected (activates)

When the code is executed with the specified input as the filename taking activates dataset contains the positive word "activate" as soon as it detects the word it gives the chime sound as well as we used matplotlib so it shows the output in graph format.

filename = "./raw_data/backgrounds/1.wav"

prediction = detect_triggerword(filename)

chime_on_activate (filename, prediction, 0.5)

IPython.display. Audio("./chime_output.wav")

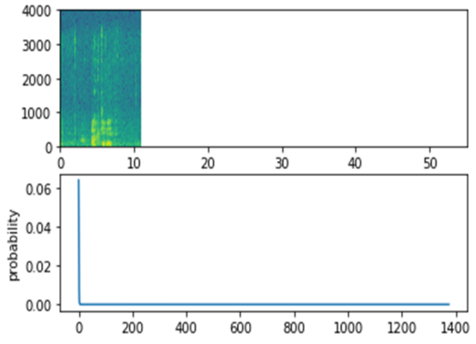


Fig-f Examples of "activate" that are not detected (backgrounds)

When the code is executed with the specified input as the filename taking backgrounds dataset contains some noises as it detects no positive word it gives the no chime sound as well as we used matplotlib so it shows the output in graph format.

filename = ". /raw_data/negatives/.wav"

prediction = detect_triggerword(filename)

chime_on_activate (filename, prediction, 0.5)

IPython.display. Audio("./chime_output.wav")

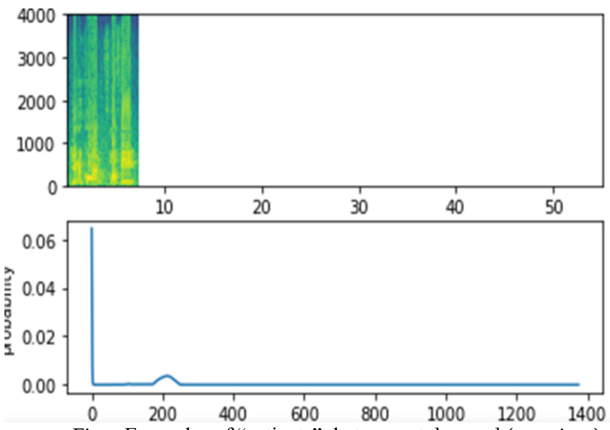


Fig-g Examples of "activate" that are not detected (negatives)

When the code is executed with the specified input as the filename taking negatives dataset contains some random words other than "activate" as soon as it detects the words it gives no Chime Sound As Well As We Used Matplotlib So It Shows The Output In Graph Format.

VI CONCLUSION

In this project, we have proposed a Trigger word detection based on LSTM for Speech data. Users from all over the world want to do their things effortlessly. Our motivation behind doing this project is to increase human-computer interaction. In this we tend to present an algorithmic program for finding wake up or Trigger word. We tend to introduce a completely unique deep learning methodology and apply it in our project. First, we select the audio file, transform the file into dataset, then feed it to the neural network and add chime sound after each wake up word recognised. Deep learning methods such as LSTM show better performance of recognizing the Trigger or Wake up words with accuracy when there are more amounts of training data

VII REFERENCES

[1] "Deep neural network-based Wake-up-Word speech recognition with two-stage detection" by Fengpei Ge, Yonghong Yan
[2] "Direct modeling of raw audio with DNNs for wake word detection" by Gautam Tiwari, Arindam Mandal
[3] "LSTM, GRU, Highway and a Bit of Attention: An Empirical Overview for Language Modeling in Speech Recognition" by Ralf Schluter, Hermann Ney
[4] "A Practical, Real-Time Speech-Driven Home Automation Front-end" by T. Ganchev and I. Potamitis
[5] "Implementation of the Wake Word for Smart Home Automation System" by Igor Stefanović, Eleonora Nan
[6] "Home Automation by Speech Recognition" by Shilpa Srivastava, Ashutosh Kumar Singh, Sanjay Kumar Nayak.