DLS /C5/ WEEK3 /A2 SI

Trigger word detection

Trigger Word Detection

Welcome to the second and last programming assignment of Week 3!

In this week's videos, you learned about applying deep learning to speech recognition. In this assignment, you will construct a speech dataset and implement an algorithm for trigger word detection (sometimes also called keyword detection, or wake word detection).

- 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 certain word.
- For this exercise, our trigger word will be "activate". Every time it hears you say "activate", it will make a "chiming" sound.
- By the end of this assignment, you will be able to record a clip of yourself talking, and have the algorithm trigger a chime when it detects you saying "activate".
- After completing this assignment, perhaps you can also extend it to run on your laptop so that every time you say "activate" it starts up your favorite app, or turns on a network connected lamp in your house, or triggers some other event?

In this assignment you will learn to:

- Structure a speech recognition project
- Synthesize and process audio recordings to create train/dev datasets
- Train a trigger word detection model and make predictions

Let's get started!

Important Note on Submission to the AutoGrader

Before submitting your assignment to the AutoGrader, please make sure you are not doing the following:

1. You have not added any *extra* `print` statement(s) in the assignment.
2. You have not added any *extra* code cell(s) in the assignment.
3. You have not changed any of the function parameters.
4. You are not using any global variables inside your graded exercises. Unless specifically instructed to do so, please refrain from it and use the local variables instead.
5. You are not changing the assignment code where it is not required, like creating *extra* variables.

If you do any of the following, you will get something like, `Grader not found` (or similarly unexpected) error upon submitting your assignment. Before asking for help/debugging the errors in your assignment, check for these first. If this is the case, and you don't remember the changes you have made, you can get a fresh copy of the assignment by following these instructions.

Table of Contents

Packages

In [1]:
```
import numpy as np
from pydub import AudioSegment
import random
import sys
import io
import os
import glob
import IPython
from td_utils import *
%matplotlib inline
```

1 - Data synthesis: Creating a Speech Dataset

Let's start by building a dataset for your 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 ...).
- Therefore, you need to create recordings with a mix of positive words ("activate") and negative words (random words other than activate) on different background sounds. Let's see how you can create such a dataset.

1.1 - Listening to the Data

- One of your 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.

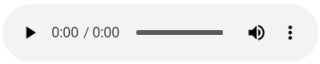
Run the cells below to listen to some examples.

In [2]:
```
IPython.display.Audio("./raw_data/activates/1.wav")
```

Out[2]:

In [3]:
```
IPython.display.Audio("./raw_data/negatives/4.wav")
```

Out[3]:



1.2 - From Audio Recordings to Spectrograms

What really is an audio recording?

- A microphone records little variations 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 as 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 44,100 numbers per second.
  - Thus, a 10 second audio clip is represented by 441,000 numbers ($= 10 \times 44,100$).

Spectrogram

- 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 any moment in time.
- If you've ever taken an advanced class on signal processing or on Fourier transforms:
  - A spectrogram is computed by sliding a window over the raw audio signal, and calculating the most active frequencies in each window using a Fourier transform.
  - If you don't understand the previous sentence, don't worry about it.

Let's look at an example.

5]:
```
IPython.display.Audio("audio_examples/example_train.wav")
```

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

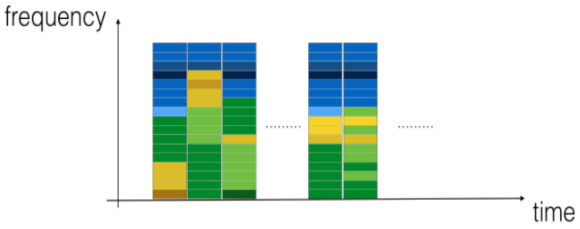


Figure 1: Spectrogram of an audio recording

- The color in the spectrogram shows the degree to which different frequencies are present (loud) in the audio at different points in time.
- Green means a certain frequency is more active or more present in the audio clip (louder).
- Blue squares denote less active frequencies.
- The dimension of the output spectrogram depends upon the hyperparameters of the spectrogram software and the length of the input.
- In this notebook, 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.
  - You'll see later that the spectrogram will be the input x into the network, and so $T_x = 5511$.

1.3 - Generating a Single Training Example

Benefits of synthesizing data

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).

Process for Synthesizing an audio clip

- 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 10 sec. clip
  - Randomly insert 0-2 audio clips of negative words into this 10 sec. clip
- Because you had synthesized the word "activate" into the background clip, you know exactly when in the 10 second clip the "activate" makes its appearance.
  - You'll see later that this makes it easier to generate the labels $y^{\langle t \rangle}$ as well.

Pydub

Visualizing the labels

- Here's a figure illustrating the labels $y^{\langle t \rangle}$ in a clip.
  - We have inserted "activate", "innocent", "activate", "baby."
  - Note that the positive labels "1" are associated only with the positive words.

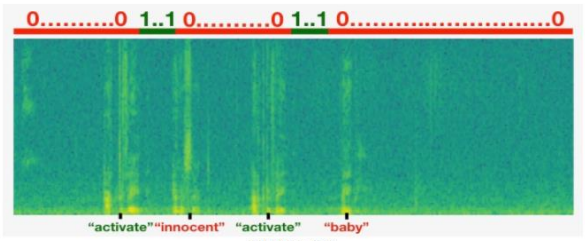


Figure 2

2 - The Model

- Now that you've built a dataset, let's write and train a trigger word detection model!
- The model will use 1-D convolutional layers, GRU layers, and dense layers.
- Let's load the packages that will allow you to use these layers in Keras. This might take a minute to load.

In [31]:
```
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.models import Model, load_model, Sequential
from tensorflow.keras.layers import Dense, Activation, Dropout, Input, Masking, TimeDistributed, LSTM, Conv1D
from tensorflow.keras.layers import GRU, Bidirectional, BatchNormalization, Reshape
from tensorflow.keras.optimizers import Adam
```

2.1 - Build the Model

Our goal is to build a network that will ingest a spectrogram and output a signal when it detects the trigger word. This network will use 4 layers:

```
* A convolutional layer
* Two GRU layers
* A dense layer.
```

Here is the architecture we will use.

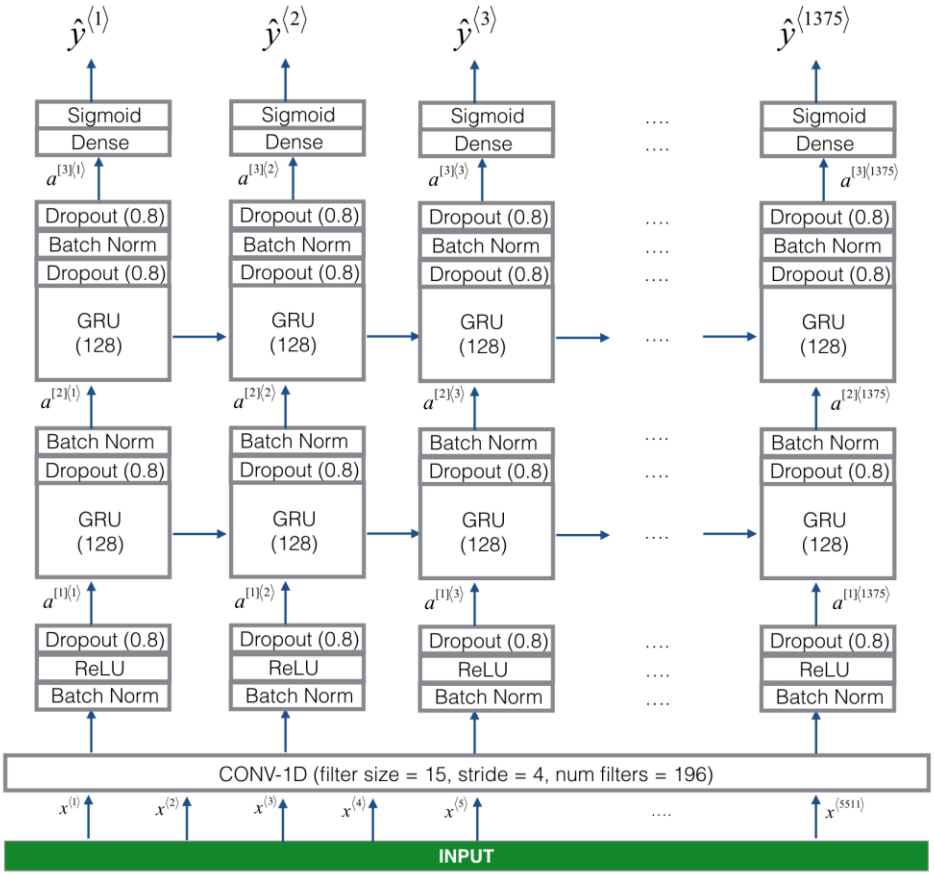


Figure 3

1D convolutional layer

One key layer of this model is the 1D convolutional step (near the bottom of Figure 3).

- It inputs the 5511 step spectrogram. Each step is a vector of 101 units.
- It outputs a 1375 step output
- This output is further processed by multiple layers to get the final $T_y = 1375$ step output.
- This 1D convolutional layer plays a role similar to the 2D convolutions you saw in Course 4, of extracting low-level features and then possibly generating an output of a smaller dimension.
- Computationally, the 1-D conv layer also helps speed up the model because now the GRU can process only 1375 timesteps rather than 5511 timesteps.

GRU, dense and sigmoid

- The two GRU layers read the sequence of inputs from left to right.
- A dense plus sigmoid layer makes a prediction for $y^{\langle t \rangle}$.
- Because y is a binary value (0 or 1), we use a sigmoid output at the last layer to estimate the chance of the output being 1, corresponding to the user having just said "activate".

Unidirectional RNN