Alberto Sabater
Data Scientist

Automatic Subtitle Synchronization through ML

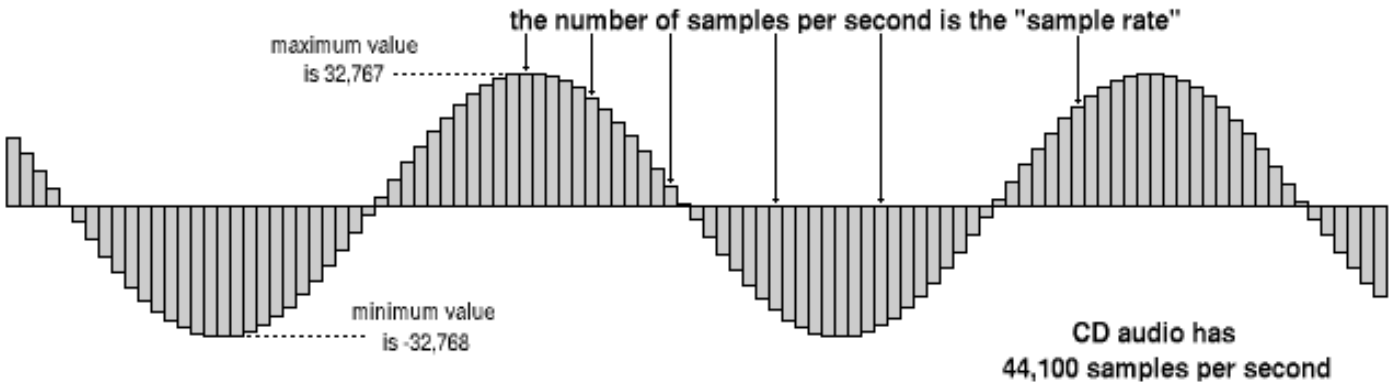
From speech recognition to music generation, a lot of work has been done in the field of Machine Learning applied to Signal Processing. The project I present here is a bit simpler than that but still very useful. Synchronizing audio and subtitles is a tedious and boring task since you have to identify when the human speech happens and modify each subtitle to fill that temporal space. But, what if a machine could do that for us?

To deal with it, in this post I describe how I have trained a Neural Network to detect human speech on videos and use that information to synchronize their subtitles.

Audio preprocessing

The first step in this project is to extract the audio from a video and preprocess it to be readable by an algorithm.

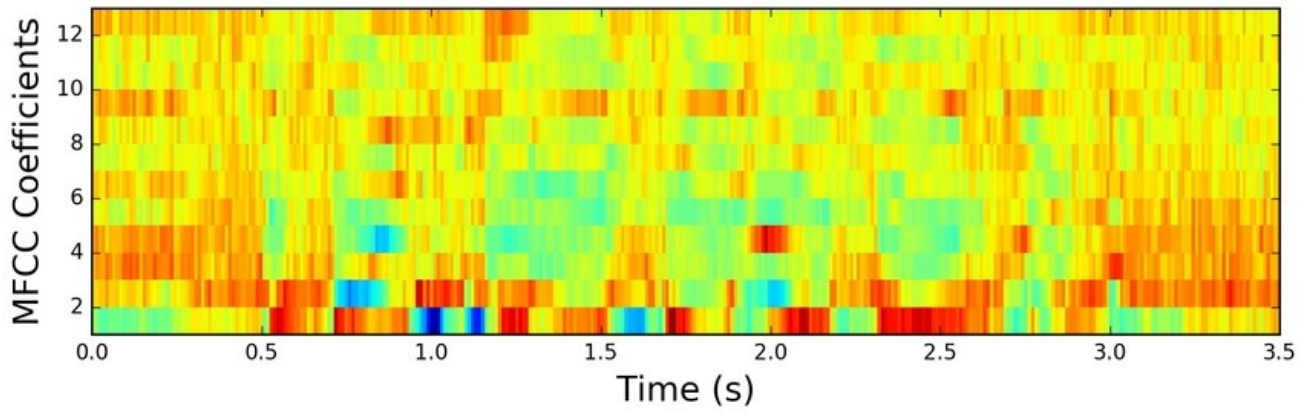
Sound is transmitted as waves, that means, one-dimensional variable which changes its value over time. To make these 'waves' readable by a computer we take samples from them at each time step. Each sample represents the height of the wave (amplitude) at some point in time. For this project, audio has been read with a sampling rate of 16 kHz, enough to cover the human speech frequency range.



Example of sampling rate in an audio wave

Our audio is now readable by an algorithm, but in order to make it easier to process, we can use *feature engineering* to extract the main components of the audio signal. The feature engineering method applied here is **MFCC** (Mel Frequency Cepstral Coefficient).

This method assumes that an audio signal barely changes in short periods of time (20-40ms) to frame the signal into small frames. For each of these frames, MFCC calculates its periodogram that identifies which frequencies are present in that period of time. However, this periodogram still has some useless information. To get more insight of the periodogram, MFCC applies Mel Filterbank obtaining how much energy exists in various frequency regions. Its result is processed by some logarithms and DCT (Discrete Cosine Transform) and finally reduced to the 2-13 obtained coefficients, discarding the rest. For a deeper understanding of MFCC, I highly recommend this lecture.



Example of MFCC features over time

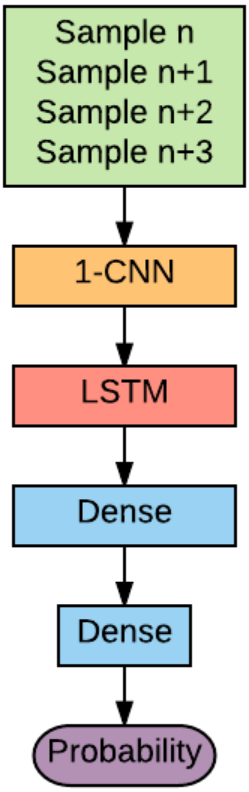
Dataset construction

Since this is a classification problem, it is necessary to have a label for each input data. The input is made of the MFCC features extracted from the audio of a video. For each video there is a synchronized subtitle, so we can know when people are speaking. With this information, we can tag each sample with 1 if there is a subtitle at that point in time (somebody speaking) or 0 if not. This is the output we want to predict.

Neural Network architecture

This problem has been approached with Neural Networks, where its input is one or more samples (MFCC features) and its output will be the probability of having a human voice in the period of time represented by the input samples.

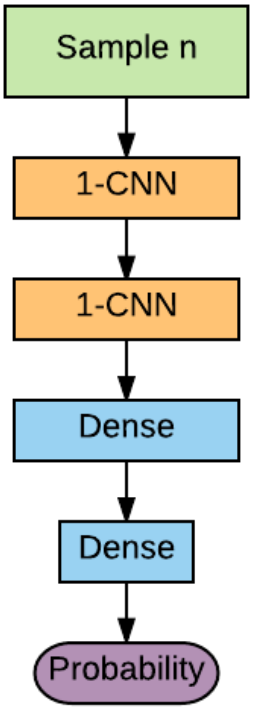
The first approach was to use a Recurrent Neural Network whose input is a bunch of samples and each of these samples is also processed by a one-dimensional Convolution Layer. The idea of this model is to exploit temporal patterns between samples and spatial patterns between frequencies in the samples.



RNN architecture

Although its training results were very good, in practice the model wasn't very accurate. This is because the input data represents a very width space of time. When using LSTMs it is necessary to take as input several samples so, for instance, if a sample represents 0.05 seconds and the input is made with ten samples, the accuracy of the model will be ± 0.5 seconds which is not very useful in practice.

However, one sample is enough to find human speech. So, the final model I used was a Convolutional Neural Network which only takes as input one sample which is processed by several one-dimensional Convolutional Layers.



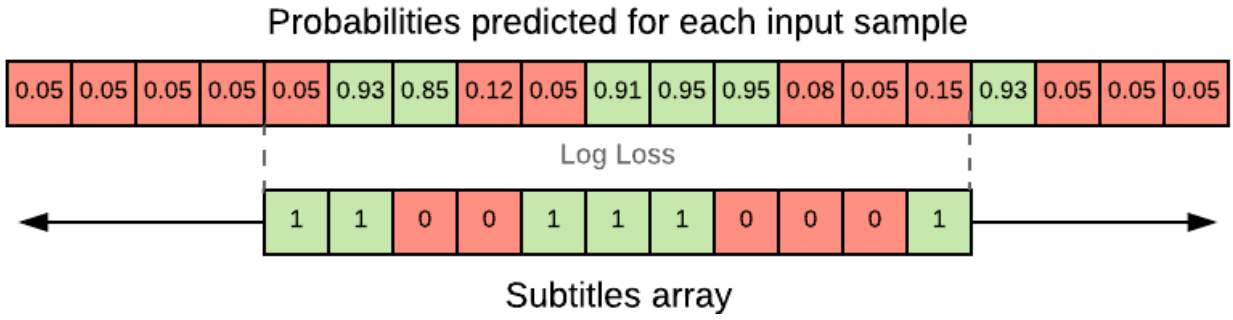
CNN architecture

Subtitle synchronization

Once the Neural Network has been trained, it's time to synchronize subtitles. To do that, we first have to extract audio from a video and its MFCC features to feed the trained Neural Network. As a result, we will have the probability of finding a human voice in each sample. These probabilities are stored in an array.

Besides the audio, we have a not synchronized subtitles in a SRT file, where each subtitle has a timestamp for its beginning and another one for its ending. Knowing the length of each subtitle, we can build an array where each cell represents the same time length as the data from the MFCC features. Each cell has a 1 if there is a subtitle there or 0 if represents the time between subtitles.

Since the video is always longer or equal than its subtitles, the probabilities array will always be greater or equal than the subtitles array. To synchronize the subtitles we take the subtitles array as a window which is moved along the probabilities array to find which is the best position where the subtitles fit the cells with high probabilities of finding a human voice. This position will be that which minimizes the Log Loss function between the subtitles array and the chunk of the probabilities array which is covered by the subtitles array. Once this position has been found, we have to translate it to seconds according to how many cells have been moved. And finally, this is the time we have to delay our subtitles.



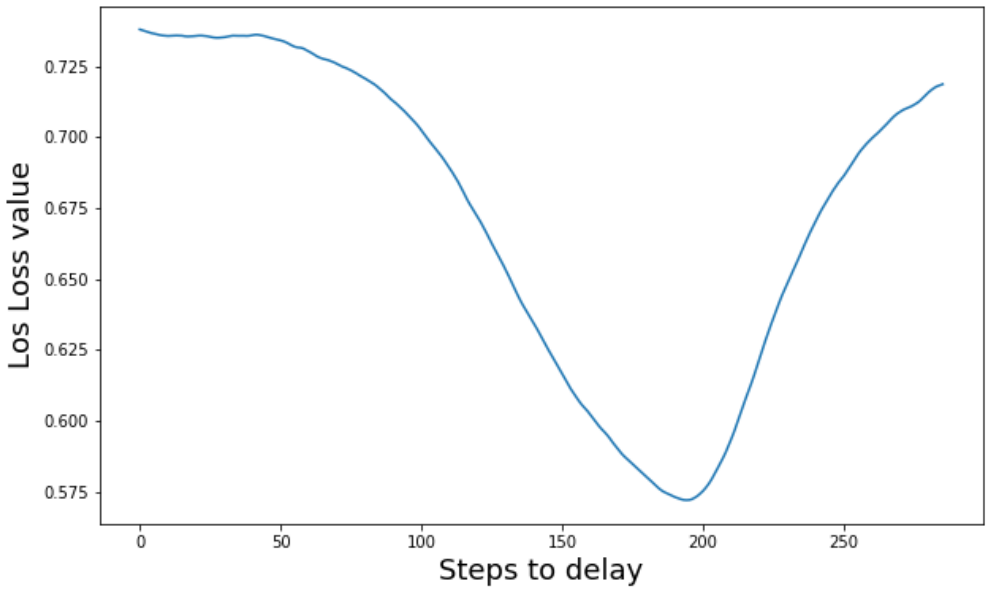
Synchronization schema

The idea of using the Log Loss as the cost function is to maximize the accuracy by penalizing false classifications.

Results

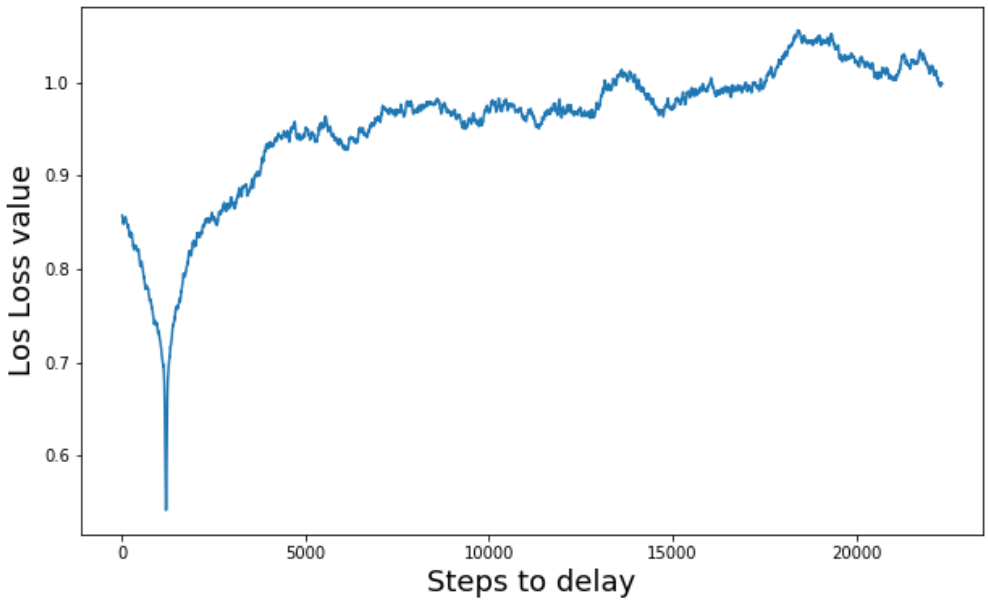
After hyperparameter tuning, the Neural Network has been trained with 'just' 5 hours of audio from two chapters of different TV Shows and two films, in different languages and video formats. To create the training (and test) dataset I use a sampling rate of 16.000 MHz and each MFCC sample takes 512 audio samples, so each MFCC feature represents 0.032 seconds of audio. With these parameters and assuming that the NN is well trained, the worst accuracy we can get is a difference of ± 0.064 seconds with the best synchronization, enough to have a good subtitle synchronization.

The next figure shows the Log Loss minimization of a TV show chapter (50 minutes). Each value in the x-axis represents 0.032 seconds to delay. All the process (from audio extraction to subtitle synchronization) has taken around **45 seconds**.



Log Loss minimization in a TV Show chapter subtitle synchronization

This other figure shows the Log Loss minimization of a film (100 minutes). This subtitle synchronization has taken around **13 minutes**.



Log Loss minimization in a film subtitle synchronization

As you can see in the graphs, the model finds in both cases the exact position where the subtitles match the audio perfectly. However, the number of steps to check (x-axis) is very sensitive to the difference in audio and subtitles length. This problem can be seen in the above example. Therefore, in order to solve this problem, I replace the previously explained subtitle synchronization by a simple **divide and conquer** heuristic. With this solution, there is no need to check every single step to get the same result.

By doing that, the **TV Show** example is synchronized (all process) in **45 seconds** and the **film** example in less than **2 minutes**. Note that, apart from the Neural Network predictions (which uses more CPU cores), all the subtitle synchronization runs on a single CPU core.

Conclusions

This project has proven to be a fast and effective method for automatic subtitle synchronization and it is also robust to noise like those subtitles which represents text in the video or human voices which are not subtitled (like songs or TV Show openings).

So, Mr. Netflix, Mr. YouTube, and Mr. VLC if you need help with your subtitle synchronization, do not hesitate to contact me. ;)

Further work

Some improvements which could be tested are the following:

1. One of the most expensive steps is the audio extraction from a video. This is done with a Linux command line tool. Therefore, the best improvement would be to find another tool to extract this audio faster or at least parallelize this extraction over all the CPU cores.
2. Test another cost function instead of the Log Loss, like Jaccard Coefficient or Dice Coefficient.
3. Use an even simpler Neural Network (without Convolutional Layers) or other algorithms like SVM or RandomForest.
4. Parallelize the final subtitle synchronization over all the CPU cores.
5. Use another feature engineering method instead of MFCC, like spectrograms.
6. Synchronize each subtitle separately instead of taking all subtitles as a unique block.

Full code is available in this Github repo.

Written on September 13, 2017