The Quote Indexer V: Skyrim

Sebastiaan Alvarez Rodriguez

January 2020

1 Abstract

With the rise in popularity of mindfulness, more and more quote datasets and websites appear on the internet, making inspirational quotes available to everyone interested in reading them. At this point, many websites exist, with hundreds of thousands quotes available. The quotes on these websites are not indexable in a uniform manner. There is no common interface.

In order to address this issue and make quotes indexable, we created a tool named The Quote Indexer V: Skyrim. At first, we wanted this tool to be able to index quotes for the game The Elder Scrolls V: Skyrim. During development, we changed the goals and created support for generic datasets. We created a generic Quote Indexer: A program that allows users to search quotes in a dataset.
Our indexer receives voice as input, and uses this to find a quote. Dataset handlers may provide extra information of the quote, as well as a hyperlink to extra information, or a path to an audiofile. We use DeepSpeech as our quote-to-text engine.

2 Introduction

More and more applications are becoming voice controlled, because this way of interacting with applications is intuitive and fast. To facilitate this, many neural networks have been created and trained, which translate speech to text, which is then further interpreted by voice controlled applications. Such networks include, but are not limited to, Baidu Deepspeech [1], Google Cloud Speech to text [2], and IBM Speech recognition [3].
Some networks are cloud based, enabling them to use high-end hardware to support translation, while others rely on the host device of the user. We used Deepspeech for our framework, which relies on the host device, as we wish our program to be able to work, even in offline environments.
In the following Sections, we will provide necessary background information about neural networks for speech recognition, give an overview of our framework, discuss our design and implementation choices.

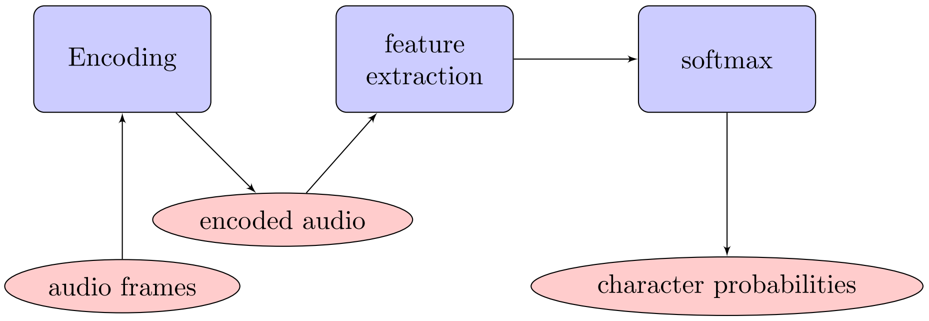

Figure 1: First part of Deepspeech: Audio to character probability distributions

3 Background

In this section, we provide necessary background information to better understand how Voice Audio Detection can be utilized to handle streaming to Deepspeech, and how Deepspeech works internally.

3.1 Voice Audio Detection

3.2 Deepspeech

The Deepspeech speech to text framework uses a neural network to translate audio to text, like many other speech to text frameworks at this date.
As with many neural networks, the Deepspeech neural network is a composition of smaller individual networks which are chained together, forming a single large network to be trained. The network consists of two high level networks: An 'feature extraction network' and a 'temporal classification' network.

3.2.1 Feature Extraction Network

The feature extraction network can be found in Figure 1. It takes audio frames as input, encodes the audio, extracts audio features, and outputs character probabilities for characters (a-z), space, blank, and apostrophe. The most interesting parts are the feature extraction and softmax networks. We show the exact feature extraction network, as presented in the original paper [1], in Figure 2. Deepspeech extracts audio features by applying 3 Fully Connected (FC) networks. Then, it applies a bidirectional Residual Neural Network (RNN) to explore speech context. Lastly, it applies a final FC network to transform the result. In the figure, we also note that the result is transformed into a continuous output of probability distributions for all supported characters.

3.2.2 Temporal classification Network

The feature extraction network provides us with individual chances for characters. Intuitively, a single chunk of output may look like "$48\%`a', 47\%`u', \cdots,$

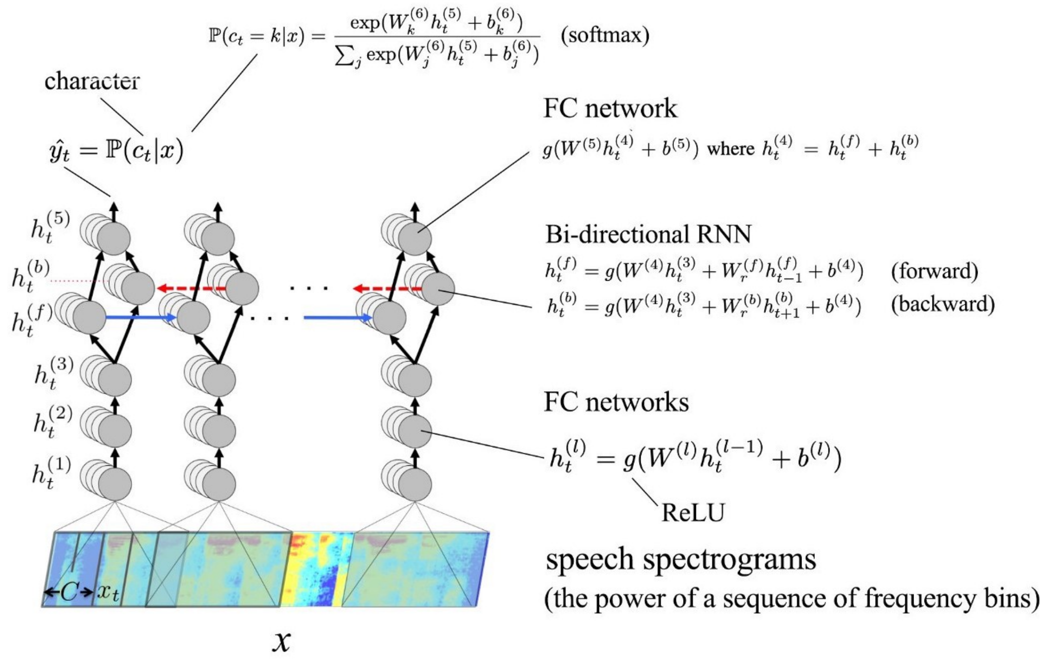
$\mathbb{P}(c_t = k|x) = \frac{\exp(W_k^{(6)} h_t^{(5)} + b_k^{(6)})}{\sum_j \exp(W_j^{(6)} h_t^{(5)} + b_j^{(6)})}$ (softmax)
character
$\hat{y}_t = \mathbb{P}(c_t|x)$
FC network
$g(W^{(5)} h_t^{(4)} + b^{(5)})$ where $h_t^{(4)} = h_t^{(f)} + h_t^{(b)}$
Bi-directional RNN
$h_t^{(f)} = g(W^{(4)} h_t^{(3)} + W_r^{(f)} h_{t-1}^{(f)} + b^{(4)})$ (forward)
$h_t^{(b)} = g(W^{(4)} h_t^{(3)} + W_r^{(b)} h_{t+1}^{(b)} + b^{(4)})$ (backward)
$h_t^{(5)}$
$h_t^{(b)}$
$h_t^{(f)}$
$h_t^{(3)}$
$h_t^{(2)}$
$h_t^{(1)}$
FC networks
$h_t^{(l)} = g(W^{(l)} h_t^{(l-1)} + b^{(l)})$
ReLU
speech spectrograms
(the power of a sequence of frequency bins)
C
x_t
x

Figure 2: Deepspeech audio feature extraction in detail

with percentages of course denoting relative probability.
A naive approach to perform speech to text is to just output the character with the highest probability. This solution does not work well: What if we have a situation like described above, where a few characters have approximately equal chances? Imagine we are gave input ‘audio’, and are translating the second character. ‘audio’ becomes ‘aadio’, using the intuitive idea.
We use a Connectionist Temporal Classification (CTC) Network. The CTC network computes the best path by taking the most likely character per time-step. Between each detected transition of characters, the best path function inserts a ‘blank’ character. After performing best path computations, CTC decodes the output by removing duplicate characters and all blanks from the path. The remaining text is our network output solution.

This solution is further processed: Beam search is used to find the most likely sequence of words. This output will be the output returned by Deepspeech.

4 Overview

Our framework controls audio recording tools, routes audio to Deepspeech, gathers converted text and queries that to a selected dataset of quotes to find a

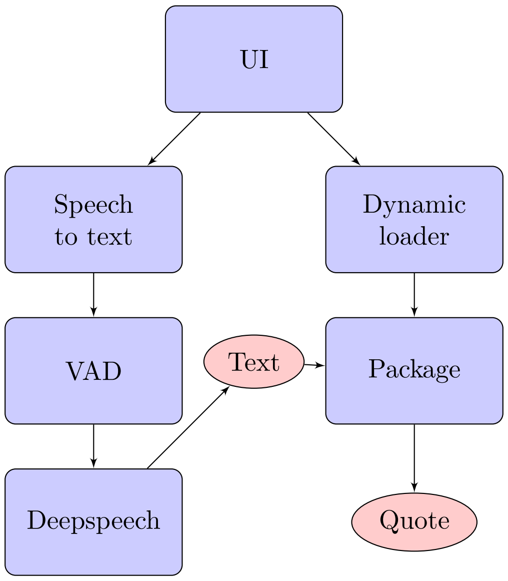

Figure 3: The Quote Indexer V: High level overview

match, as shown in Figure 3. *The Quote Indexer V: Skyrim* is designed as an easy to setup framework, which handles speech to text, and queries found text to a database, with reasonable error correction. The framework automatically handles resource regulations, controlling audio input devices and using voice activity detection.

In order to translate and find a spoken quote, our framework goes through the steps described below. *The Quote Indexer V: Skyrim* first takes control of the main audio input device, and configures Voice Audio Detection to listen for voice activity.
Once the user starts speaking, we stream the audio frames to Deepspeech in the required sampling rate of 16000 samples per second, single channel audio to be interpreted. VAD stops the stream once the user is finished speaking, and Deepspeech will interpret the audio, and return interpreted text to *The Quote Indexer V: Skyrim*.
Our framework then does a dynamic code call to driver code for the currently selected dataset, which will handle querying the dataset.
If a close enough partial match is found, it is returned to our framework, which displays it to the user.
We describe the individual components and design choices in greater detail in Sections 5 and 6

5 Design

Here we discuss our design choices in greater detail.

5.1 Dynamic Loading and Packages

To make extending the framework really simple, we introduced packages for this work. The idea is simple: We add packages to the packages folder. Each package is displayed in the UI. If a package is selected to be loaded, a single dynamic call is made to the package, which then interprets its dataset and loads it into RAM.
Once the user has spoken a query, we take the text input and make a second dynamic call to the currently loaded package, which then should try to find a matching query. The query is asynchronously returned to the UI, which then displays the query, and any additional information as required.

5.2 Voice Activity Detection

In most scenario's, we are not in a completely quiet environment when we want to make a query. To reduce the chances environment noise is interpreted as speech, we implemented a VAD system. This proved fairly effective, and in most tested situations will start and stop audio streaming to Deepspeech correctly.

5.3 Matching quotes

In principle, each package is free to handle matching translated text to any quote in it's dataset. A simple word error rate may be used, or character error rates, or sequence error rates.
For our reference implementations, however, we followed a fairly complex principle. For a given string α, we walk over all strings β in the dictionary. For every β, we very quickly check whether it is somewhat resembling α by comparing string length resemblance ratios. Then, we perform a second check, this time actually comparing characters. In order to keep computations fast, we only look at relative character occurrences, but we do not care about when characters occur at what positions as this would require too much computation at this stage. In the third (final) check, we divide α and β in blocks, and repeat a combination of the first and second check for each block.
The result of each check is a probability for β. After each check, we stop considering β as a target candidate if we do not find a target probability greater than a predefined value $0 < \gamma < 1$.
When experimenting, we found that a γ value of 0.6 proves most optimal, when comparing computation time against result quality: The correct quote is nearly always found using this γ, while it stops considering 'junk' β's quickly to keep computation time low.
This strategy is superior to using word error rate in terms of output quality, but it is a bit slower. Also, it is packaged in a Python library `difflib` we used.

6 Implementation

In this section, we will discuss our framework implementation in greater detail. Mainly, we discuss package structure and user interface asynchronous communication techniques.

6.1 Language

We wrote our framework in Python 3.7, as it has support for multiple operating systems, great libraries available, and since our target speech to text engine, Deepspeech, has Python library support. Using this language, we were able to deliver results fast and with both great reliability and compatibility.
Also, because of Python's great support for asynchronous communication schemes and dynamic loading protocols, we were able to implement our framework using rather few lines of code.

6.2 User Interface

We implemented a User Interface (UI) in Qt5: A modern Graphical User Interface (GUI) library with support on multiple operating systems, at least including Windows 10 and most Linux distributions.
The UI is pretty basic, making it easy to understand and interact with it. Main features of the UI are the 'Record' button, which activates VAD, prepares audio input devices to be used and prepares Deepspeech.
The 'Hyperlink' button becomes enabled if a package returns a valid URL for a found quote. Upon clicking, it will open the default installed web browser and redirects to given page.
The 'Sound' button becomes enabled if a package returns a valid path to an audio file for a found quote. Upon clicking, it will play the audio file on default device speakers. Paths are considered valid if they point to existing files, and audio files are considered if they are of type WAV, AU, AIFF, MP3, CSL, SD, SMP, or NIST/Sphere.

6.3 Deepspeech

We chose for Deepspeech as our speech-to-text engine, because of the following reasons: It is reasonably fast in translating, even on a mid-end CPU's. Besides that, it is modern, well documented and has a relatively low word error rate (7.5%) on the LibriSpeech clean test corpus [4].
For our implementation, we initially used Deepspeech version 0.5.9. the last day before demo-day, we implemented support for the newly available Deepspeech version 0.6. This version worked on authors voice much better than 0.5.9 and earlier version.

6.4 Packages

As stated in Section 5.1, we allow packages to be loaded at runtime, and we allow them to handle requests for finding closely matching quotes in their dataset. We use Python's `importlib` library to import code from packages at runtime. As we have documented extensively, we expect code to be available in a directory relative to our project root, in a file named `quotelist.py`. This file is loaded at runtime, and it should provide an object with which we can interact. Specifically, it must be able to handle 2 commands: On object creation, it must be able to accept a path to its dataset file, and load strings from this database. Furthermore, the package driver code must be able to answer a `find` requests, which passes interpreted speech as text and expects either a string result containing a match, or `None` if no result is found.
This is all a package has to provide in order to function, making it simple to create a package.

7 Conclusion

We have shown how to construct a powerful speech-to-indexing framework in Python, which allows users to speak parts of quotes as queries for a dataset. We also showed how to make the framework extendable, by introducing dynamic code protocols.
In the future, implementing these 'speech-driven' services will become more important: Keyboards and mouses are getting old, and people want to communicate and interact with programs in a more natural way than with mouse and keyboard. We hope to see much more speech-driven applications in the future.

References

[1] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. "Deep speech: Scaling up end-to-end speech recognition". In: *arXiv preprint arXiv:1412.5567* (2014).

[2] Ashwin P Rao. *Predictive speech-to-text input*. US Patent 7,904,298. Mar. 2011.

[3] Hagen Soltau, George Saon, and Brian Kingsbury. "The IBM Attila speech recognition toolkit". In: *2010 IEEE Spoken Language Technology Workshop*. IEEE. 2010, pp. 97–102.

[4] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. "Librispeech: an ASR corpus based on public domain audio books". In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2015, pp. 5206–5210.