CS 682: Course Project Proposal

Arun Dunna

The project is being done solely by Arun, so no work will be split between any other people. In addition, the project is not being shared with another course, and will be fully for this course with the intention of continued open source development.

1 TASE and TASP

1.1 TASE: Text-Audio Sync Engine

TASE is an engine designed to synchronize words contained in a document or a collection of documents (for example, a PDF file) with speech found in an audio file or a collection of audio files. The engine will be created with modules, allowing it to be easily implemented in a variety of projects. The main application that will be developed to use TASE is TASP.

1.2 TASP: Text-Audio Sync Platform

This is a platform with the primary use case of synchronizing audiobooks with text files. This will allow users to seamlessly switch between listening to an audiobook and reading it, such as from listening in the car to reading in bed. More time could be spent reading and learning if platform switching were effortless for the user. Although software like this exists, it is not widely implemented nor open source, and is used by a small amount of books (see WhisperSync [6] and Learning Ally [3]). Thus, the goals are to develop this as an open source project that can be self-deployed, and maintain enough generality to be applied to a variety of sources (audiobooks, lectures, news broadcasts with transcripts, etc.).

The primary case here will be synchronizing user-supplied books (supported formats are to be determined) with the accompanying audiobook. As a result, the datasets used will consist of public domain works, so that the model can be distributed without copyright issues:

Text: (a): Project Gutenberg [2]

Audio: (a): Project Gutenberg [2]
(b): LibriVox [4]

For these datasets, the complete works will be scraped and then compared to determine which books match. If books match, then they'll be retained and used in the model's dataset.

In order to cross-reference positions between text and audio, but minimize the amount of data stored and processing required, on-the-fly processing will be done using estimates and ranges. Using the average speaker's words per minute (WPM), we can determine where approximately we expect words in audiobooks to appear in the text format, and vice versa. We can then use this as the centerpoint, and span outwards until we've found the particular word. Once we've found the word, we'll ensure that the word is in proper context by using the preceding and following 50 words. If a high enough percentage matches (allows for dialects, model inaccuracies, etc.), then we can report with a very high probability that we've found the right spot. Otherwise, we continue the search until we've found the match.

To minimize the amount of error, we'll use a baseline architecture developed by Hannun et al. [7], with the implementation developed by Mozilla [1] using TensorFlow [5]. We'll use this implementation to develop our own model, and can perform cross-validation to find optimal variable values, namely: the hidden layer widths, dropout rates, beta parameters, the epsilon parameter, the learning rate, and the time slice length for audio segments.

The main reading will consist of both the DeepSpeech architecture [7] and implementation [1], TensorFlow [5], and Adam [8] (stochastic optimization), which will allow for better implementation and background.

Lastly, results will be evaluated based on the accuracy of the model on the held-out test set, and the feasibility of the system through accuracy and speed of text-audio synchronization. The feasibility can be measured through the following metrics (for both text to audio synchronization, and audio to text synchronization):

(a) Speed for finding the correct position (ms)

(b) Accuracy of the reported position vs. the correct position (%)

(c) Number of times re-searching has to be done to find a word (count)

These can be improved through model tweaks, changing the input data to the model, improving the estimation method for synchronization, and selecting neighboring less common words instead of searching for words such as “the” or “and”. We can evaluate each of these metrics visually through scatter plots and CDFs.

References

[1] Deepspeech: A tensorflow implementation of baidu’s deepspeech architecture. `https://github.com/mozilla/DeepSpeech`.

[2] Gutenberg. `https://www.gutenberg.org/wiki/Main_Page`.

[3] Learning ally - audiobook subscription. `https://learningally.org/Adults/Join`.

[4] Librivox | free public domain audiobooks. `https://librivox.org/`.

[5] Tensorflow. `https://www.tensorflow.org/`.

[6] Whispersync for voice | audible.com. `https://www.audible.com/ep/wfs`.

[7] Awni Y. Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, and Andrew Y. Ng. Deep speech: Scaling up end-to-end speech recognition. *CoRR*, abs/1412.5567, 2014.

[8] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.