3SA: Semantic Search for Speeches in Audio

Syed Ikram **Venkata Sai Pavan Kumar Kosaraju** **Yifan Li**

{sikram, vkosaraj, yla570}@sfu.ca

CMPT - 825 Natural Language Processing
Final Project Report

Abstract

Semantic search is the ability to search for documents by understanding the overall meaning of the query rather than using simple keyword matches. Recent breakthroughs in NLP like Bert, Albert, Roberta, etcetera, paved the way for the development of such powerful semantic search engines. But most of these search algorithms are mainly focused on textual information, i.e., both the document and the query are in natural language. In this project, we aim to develop a semantic search algorithm for arbitrary objects (objects which are not in natural language), specifically for speeches in audio, by leveraging advanced NLP techniques. We propose 3SA, Semantic Search for Speeches in Audio, which can enable the search for audio files, semantically. We perform our experiments on the LibriSpeech dataset and further evaluate our search results using basic information retrieval metrics.

1 Motivation

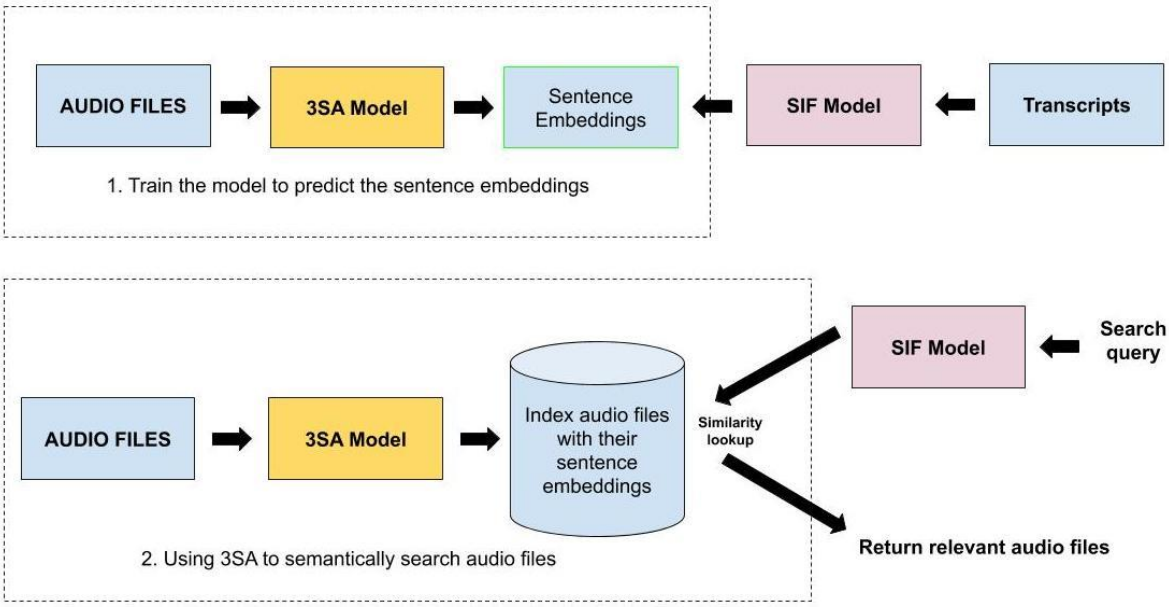
In an age of digital media, many of the modern search engines use keyword searches, which is either curated by experts of that field or is tagged using some algorithm based on the presence of the keyword and meta information of the file. Searching objects which aren't typically text is challenging. Moreover, keyword search engines don't use the semantic similarity to provide optimized search results. But we believe, for any arbitrary object, if there is some text associated with it, then it can be represented in natural language space, thereby enabling semantic search.

The idea to represent objects in natural language space has been fairly proven in the paper DeViSE (Frome et al., 2013). The authors of the paper tried to label objects in an image using the semantic information gleaned from the unannotated text. Using a similar idea, we want to learn embeddings of audio files in the LibriSpeech dataset, which are semantically close their respective transcripts. We can then use those embeddings to represent audio files in natural language space to enable semantic search.

2 Related Work

Speech2Vec (Chung & Glass, 2018) is the state-of-the-art approach to learn audio embeddings from an audio signal. The architecture is capable of representing variable-length speech segments as fixed-dimensional speech embeddings that capture the semantics of the segments. The idea is similar to Word2vec, except it is for speeches. The main issue of this approach is that the embeddings are word-level representations since the audio files are segmented during the training process. Moreover, it has been noticed that the same word has different embeddings due to different speakers, channel and background noise. So, if the same speech is read by different readers, they would have different representations and thus, it is not suitable for this task.

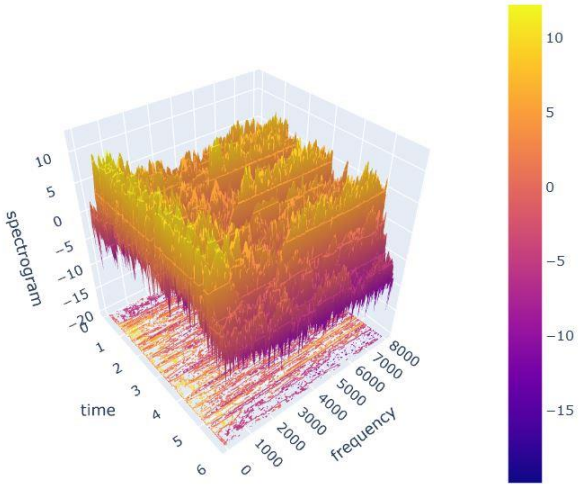
Unlike Speech2Vec, Haque et al., (2019), proposed an approach which can learn sentence-level embeddings for a given audio signal. But since their evaluations are based on extrinsic tasks like speech and emotion recognition, they may not necessarily capture the overall semantic meaning of the transcripts. For instance, audio files with transcripts 'A rabbit is running from an eagle' and 'The eagle tries to hold the rabbit' will have different embeddings even though they are similar in nature. Thus, these embeddings may not fit for our purpose of searching files semantically.



1. Illustration of 3SA project flow.

3 Approach

The goal of this project is to build a model that takes an audio file as an input and predicts the sentence embeddings of the utterance (speech text) in the file. We can then use this model as an algorithm to represent the audio files in natural language space, thereby allowing us to search for these files semantically. The flow of our project is depicted in figure 1. Initially, we preprocess our audio files to extract spectrogram features from the audio signal. Spectrograms of audio signals are



2. Spectrogram of a sample audio file.

proven to be one of the most used feature representation for audio files. As mentioned by Flanagan (1972), trained humans can decode phonemes from spectrogram signals, thus making it easy for any machine learning model to 'learn' to understand these phonemes from spectrograms. A spectrogram of a sample audio file is shown in figure 2. It is the representation of the Fast Fourier transform applied to the sound wave. It separates the frequency and the amplitude of the acoustic speech wave. In the figure, the logarithm of the spectrogram is calculated as it is commonly connected to the way people hear. Here the frequencies are in the range of (0, 8000) according to the Nyquist theorem, and time represents the length of the audio file in seconds. The degree of amplitude from dark to light represents different frequencies at their corresponding time frame.

3.1 Predict Sentence Embeddings

We start by training our model to predict the sentence embedding of a transcript, given an audio file. To learn these sentence embeddings, we follow a supervised learning technique by having pre-trained sentence vectors as 'true' labels. The idea is similar to speech recognition, except here we predict sentence embeddings instead of words.

To map the audio files in the transcripts embedding space, we first need to have sentence vectors for all the transcripts which contain semantic information. We do this using Smooth Inverse Frequency (SIF) technique (Arora et al., 2017). SIF model has proven to manifest good accuracy scores on the STS (Semantic Textual Similarity)

benchmark dataset (Agirre et al., 2017). The idea is to calculate a weighted average of the word embeddings in a sentence and then remove the projections of the average vectors on their first singular vector. The resulted embeddings represent sentence vectors which captures semantic information. We use this method to calculate sentence vectors for all the transcripts in our data. The same is represented as 'SIF Model' in the figure 1. Now that we have the audio files and their respective transcripts transformed into sentence embeddings, we train our model so that when an audio file is fed as an input, our model should predict the respective sentence embedding.

3.2 Using the model to search semantically

Once we have a trained model, we can use it to represent our audio files as sentence vectors and index them accordingly. To query the files, we first convert the search query into a sentence vector using the SIF model. Since both the audio files and the query are now mapped into the same vector space, nearest neighbors in this space describe the same concept. So, we fetch the K-nearest neighbors of the query sentence vector in the indexed audio files and return them. The returned audio files when played are expected to have either the query words in it or may contain a speech that is semantically similar to the search query.

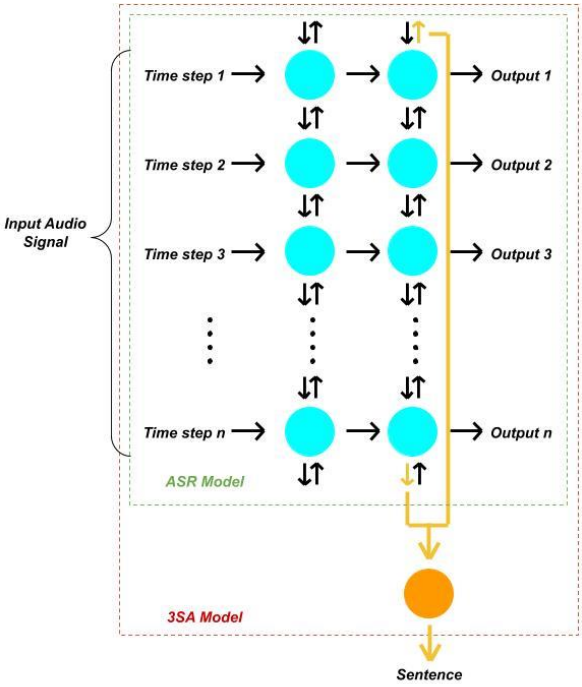
4 Model architecture

Since the idea is quite new, there are no pre-trained models available for this task. There is no right way to build the model, so we designed two model architectures to test our hypothesis. These models are inspired from existing speech recognition models and are fairly simple and intuitive.

4.1 Architecture 1 – ASR + Dense

The first architecture uses pre-trained layers of an ASR (Automatic Speech Recognition) model, which are then coupled with a few dense layers. The model architecture is shown in figure 3.

The ASR model takes an audio signal as input and predicts the output speech. But we discard the output layer and instead use the hidden states of the GRU layer (Bengio et al., 2014). The intuition here is that the hidden state at the final GRU layer represents the learned representation of the audio signal which is in-line with words that are to be predicted. We then pass the hidden output to a few dense layers to predict the sentence vector. The number of dense layers is 2 (300 units each) for the *short* version of this model and 3 (two 500 units + final 300 units) for the *long* version. The idea is that the immediate dense layer accepting GRU output will learn to concatenate the hidden representation and the final dense layers will map the values to respective positions in sentence vector.



3. ASR + Dense model. The blue blob represents bidirectional GRU layer coupled with batch normalization and activation layers. The orange blob represents dense layers.

4.2 Architecture 2 – GRU + Dense

This model is similar to the above architecture but instead of using a pre-trained ASR model, we use our own GRU layers and learn the weights from scratch. Since the pre-trained ASR model might be biased towards the task of speech recognition, we believe it is best to train the model and learn the weights which are more pertaining to our task.

5 Data

We used the LibriSpeech ASR corpus for this project. LibriSpeech ASR corpus (Panayotov et al., 2015) is a large corpus of read English speech, curated for training and evaluating speech recognition models. It is available to download freely. The corpus contains up to 1000 hours of the

recordings from various speakers reading public domain audiobooks. The dataset is divided into training, development, and testing sets by the curators themselves. Each file contains a random sentence or part of a sentence of the public domain audiobooks, read by one of the 1166 speakers. The corpus contains approximately 28K audio files in the training set (train-clean-100) and almost 2.8K audio files in both dev (dev-clean) and test (test-clean) sets respectively. Given the time and resource constraints, we limited our training set to 5000 files, dev and test sets to 10% of training set i.e. 500 files each.

6 Code

We leveraged PyTorch package for designing architecture 2. We implemented a simple loss function for this model since the provided loss functions are not so convenient. The framework also provides LibriSpeech an in-built dataset and some pre-processing tools for processing audio files. For SIF model, we used Fast Sentence Embeddings package. But for dynamic transformation during training, we built our own wrapper functions for SIF model to convert our transcripts into sentence embeddings. On the other hand, for architecture 1, since the ASR pre-trained model is trained in Keras, we wrote our own scripts to pre-process the data. These scripts can feed the data batch-wise on the fly during training of the model. Apart from training, we also wrote a class to search files semantically using NMSlib package. All our code is present in Gitlab[1].

7 Experiments

We started our experiment by training the second architecture since it might take significant amount of time to learn the weights. The models take spectrograms as input and predicts the embeddings. We trained both the architectures for 80 epochs each. Training time is significantly higher for model 2 even when running on GPU since it has lots of trainable parameters. We used cosine similarity as our loss function as shown in the equation below. Here *x* represents the predicted sentence embedding by the model and *y* represents the actual sentence embedding (given by the SIF model). We then calculate the cosine similarity be-tween the two, represented as *cos(x,y)*, and subtract it from 1. So, when both *x* and *y* are equal, it means that our model predicted the right embedding and the loss will be zero since cosine similarity will be 1. And when the prediction is wrong, our loss will be higher.

$$loss(x, y) = 1 - cos\ (x, y) \quad (1)$$

Evaluation is the trickiest part of this project. Unlike the STS benchmark dataset, we do not have any human-rated audio-sentence pairs for LibriSpeech. Moreover, to assess the performance of our model effectively, we need an objective way to measure the quality of our search results. So, rather than comparing our predicted embeddings simply with the actual embeddings, we find where the predicted embeddings are in the vector space defined by the SIF model. To achieve this, we borrow some techniques from information retrieval.

We map all our test data into the SIF embedding space using our model. Note that these are the audio files in the natural language space. Then, for a selected set of audio files in the test data, we paraphrase the original transcript. The way we paraphrase these transcripts is quite simple. We use back-translation technique where we translate our text to a particular language and translate them back to the original language. Now, for these selected paraphrased transcripts, we search for K neighbors using cosine distance in the SIF embedding space. If we were able to find the corresponding audio file in these neighbors, then we can say that our model successfully predicted and mapped the audio files to the right positions in the vector space.

$$Accuracy = \frac{\sum_P search(p,k)}{|P|} \ where\ p \in P \quad (2)$$

$$search(p_i, k) = \begin{cases} 1\ if\ a_i\ within\ k \\ 0\ otherwise \end{cases} \quad (3)$$

Equations 2 and 3 provide the formula for our evaluation metric. Here *P* is the set of paraphrased transcripts. For each p_i in *P*, we search *k* neighbors. Here a_i denotes the corresponding audio file. If we were able to find a_i within *k* neighbors then we add 1 and 0 otherwise.

[1] GitLab repo here.

8 Results & Analysis

As mentioned earlier, we mapped all the test data (500 audio files) into embedding space using our trained model. Then, we randomly sampled 200 files from our test set and paraphrased their transcripts using back-translation. Further, we mapped these paraphrased transcripts using SIF model into the embedding space. Then for each of these transcripts, we searched for K = 1, 3, 5, and 10 neighbors in the embedding space. The results are shown in table 1.

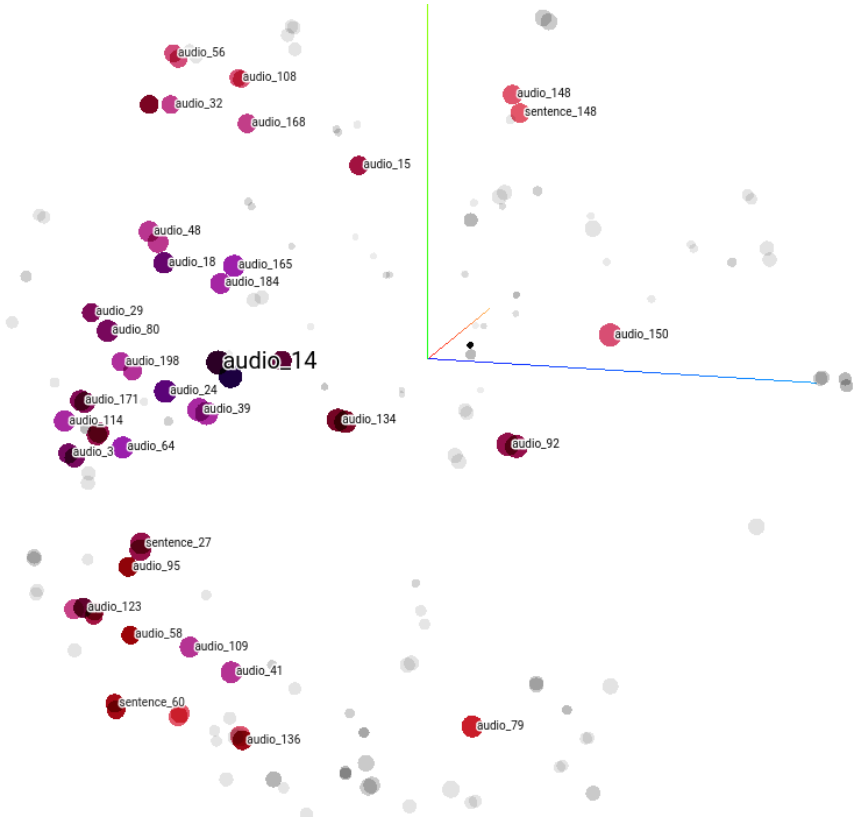
Table 1: Accuracy scores for model architecture 1 with different neighbors.

Model 1 Accuracy	
Neighbors	**Accuracy**
K = 1	0.365
K = 3	0.37
K = 5	0.38
K = 10	0.39

Note that the above results are for architecture 1. To interpret these results, consider the row K = 5 i.e. for each paraphrased transcript we get 5 nearest neighbors in the vector space. If the corresponding audio file is within these neighbors, then we consider it as true positive, hence 1 in equation 3. Here, the accuracy score is 38% which means for 200 paraphrased transcripts only 38% ~ 76 files where successfully mapped by our model in the sentence embedding space. Overall, the accuracy scores are below average for all the K values.

We further analyzed these results by plotting all the vectors in a 3D space as shown in figure 4, using Tensorboard. The 300D vectors are mapped in to 3D space by calculating their respective PCA components. The plot shows both the embeddings predicted by the model (denoted as audio_*i* for audio file *i* in the test set), and paraphrased transcript embeddings generated by the SIF model (denoted as sentence_*i* for the paraphrased transcript of the i^{th} audio file) as points in a 3D space. Notice that the colors are same for same *i*'s. We can see from the plot that our model is considerably working to an extent by predicting the embeddings. For instance, if we consider *audio_148* and *sentence_148* (both represented as

4. Projection of embeddings predicted by the model (*audio_i*) and the paraphrased transcript embeddings generated by SIF model (*sentence_i*) in the same vector space using Tensorboard embedding projector.



orange blobs to the right of *z*-axis), they both are close to each other. So, our model predicted the actual sentence embedding of audio file 148 and hence it is close to the paraphrased transcript.

On the other hand, if we check *audio_14* near the centroid, it is represented as a black blob. This is because, there is an overlap of point here i.e. the model is sometimes predicting same vector for different audio files. Upon inspecting these files, we think the following are the reasons why the model is performing poorly.

Pronunciation effect on spectrograms: The way the words are pronounced differs from person to person. While some people speak fast, others take time to read the speech. Change of pronunciation might completely change phonemes and the resulting spectrogram. For instance, words like *two* and *too* might have similar spectrograms. Change of pitch might also alter the spectrogram of same word. Due to this it is sometimes difficult for the model to learn from the spectrograms alone.

Incomplete transcripts: Most transcripts in our data are part of a sentence and not the whole sentence. Due to lack of subject tags and grammar in such incomplete sentences, it is hard to capture the semantic meaning by the model.

Hidden layer representation: Considering only the hidden layer output at the final GRU layer as sentence representation might not be working as expected. Most state-of-the-art ASR models further use language models and decoders to recognize the speech text. So, the GRU layer may not necessarily capture the required information. Alternatively, we think, capturing the output representation at all timesteps, might have helped the model to produce better results.

Given all the above issues, our model is still able to predict the embeddings for at least some of the files. And if we were given more time to solve these problems, our approach would have worked appropriately. The application of this model is also illustrated in the project notebook evaluation section (refer to our GitLab repo). When we enter a query text, we were able to retrieve the relevant audio files. We cherry picked the results in the notebook to demonstrate the capability of a 3SA model when it is trained properly.

Failure of Model 2: The model failed to produce sub-optimal audio embeddings (accuracy of 0.05) as they highly rely on the audio signal to train the model, while completely ignoring the transcripts. In any speech recognition model, the audio input file contains silences that don't align with the transcripts (commonly known as the alignment problem), where repeated characters are observed as the phoneme or word durations are either too long or too short. This results in different referenced words from the actual ones as the audio file is essentially a stream of continuous speech in time, thereby increasing the word error rate (WER). Aligning of the audio input with its text transcript can be achieved by implementing Connectionist Temporal Classification (CTC) algorithm, which handles different variations in the audio (audio length per character) and removes duplicates and silences. Since our model lacks such decoding system as we are not recognizing the speech, the hidden layers fail to learn a representation just like in the first model. And since this model is trained from scratch, it is not mature enough to give hidden representations like the first model.

9 Future Work

In future work, we will focus on using a 2D Convolutional Neural Network (CNN) based on Mel-scaled spectrogram acoustic features (Berlage et al., 2020) to generate audio embeddings. We also plan to include acoustic information from audio representing the duration of pauses and pitch frequency, which will help us in substantial error reduction. The audio embeddings generated from the forward pass through the CNN network will be extracted from the pre-final layer of the output layer. CNN will provide a good architecture and compact representation for learning a higher-level structure (feature representation) of the audio input in a higher dimension. This will result in high-quality audio embeddings proving better accuracy on the given task.

The other way to approach this task is by using a Text-to-Speech model. Instead of learning sentence embeddings from audio, we can learn audio embeddings from text. Following this approach might also mitigate the aforementioned issues as we will be in control of generating the audio signals.

Acknowledgments

We would like to thank Frome et al., (2013), the authors of the paper *DeViSE: A Deep Visual-Semantic Embedding Model*, which inspired us to carry out this research.

We would like to thank Olive Borchers for fse (Fast Sentence Embeddings) library. GitHub repo can be found here.

We would like to thank Kartik Chaudhary for the article *Understanding Audio data, Fourier Transform, FFT and Spectrogram features for a Speech Recognition System.* Link here.

Finally, we would like to sincerely thank Prof. Dr. Angel Xuan Chang and Prof. Dr. Anoop Sarkar for CMPT-825 NLP course. The course content and especially the assignments *Robust Phrasal Chunking* and *Attention* helped us to understand the working of RNNs and allowed us to build the model architecture for our project.

References

Agirre E, Cer D, Diab M, Lopez-Gazpio I, Specia L. (2017). *Semeval-2017 Task 1: Semantic Textual Similarity Multilingual and Crosslingual Focused Evaluation*. Proceedings of SemEval 2017.

Arora S, Liang Y, Ma T (2017). *A Simple but Tough-to-Beat Baseline for Sentence Embeddings*. Int. Conf. Learn. Represent. (Toulon, France).

Flanagan J. (1972). *Speech Analysis, Synthesis and Perception*. Springer- Verlag, New York.

Frome A, Corrado G, Shlens J, Bengio S, Dean J, Ranzato M, Mikolov T. (2013). *DeViSE: A Deep Visual-Semantic Embedding Model*. NIPS 2013.

Hannun A, Case C, Casper J, Catanzaro B, Diamos G, Elsen E, Prenger R, Satheesh S, Sengupta S, Coates A, Ng A. (2014). *Deep speech: Scaling up end-to-end speech recognition*. arXiv preprint arXiv:1412.5567, 2014.

Panayotov V, Chen G, Povey D, Khudanpur S. (2015). *LibriSpeech: an ASR corpus based on public domain audio books*. ICASSP 2015.

Berlage O, Shahshahani M, Graus D. (2020). *Improving automated segmentation of radio shows with audio embeddings*. ICASSP 2020.

Chung Y & Glass J. (2018). *Speech2Vec: A Sequence-to-Sequence Framework for Learning Word Embeddings from Speech*. arXiv preprint arXiv:1803.08976.

Haque A, Guo M, Verma P, Fei-Fei L. (2019). *Audio-Linguistic Embeddings for Spoken Sentences*. arXiv preprint arXiv:1902.07817.

Cho K, Merrienbor B V, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, Bengio Y. (2014). *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation.* arXiv preprint arXiv:1406.1078.