Deep Learning Approaches for Online Speaker Diarization

Chaitanya Asawa
casawa@stanford.edu

Nikhil Bhattasali
nikhilxb@stanford.edu

Allan Jiang
jiangts@stanford.edu

Abstract

Speaker diarization solves the problem of "who spoke when?". It is a difficult task, especially in the online setting, where speaker identities and the number of speakers are not known. We explore this task using multiple approaches, including temporal classification, speaker change detection, and speaker embedding generation, primarily focusing these approaches on neural architectures.

1 Introduction

The problem of *speaker diarization* answers the question "who spoke when?" in an audio recording that has an unknown number of speakers and an unknown amount of speech (Xavier and et al., 2010). This problem has long been interesting for applications in meeting transcription, speaker indexing, and document content structuring, and we think speaker diarization will be critical to the next generation of voice interfaces in free dialog settings.

We aim to design a model which, in an online fashion, assigns IDs to speakers participating in a conversation and continually determines the ID of the speaker that is currently active. We would like this model to generalize to unseen speakers—which requires us to, rather than learn very well about the speakers in our training set—be able to generally understand speech input and speaker distinction. In addition, the online component of this model introduces complexities such as not being able to see future timesteps, which can be useful information for the model in making decisions.

We try many approaches to tackle this task. One method is to simply detect when a speaker has changed, and using some notion of state (such as average of the features), we can maintain a list of speakers to determine who the speaker is at a timestep. Other approaches we try include developing embeddings that are a sufficiently powerful representation of a speaker's characteristics, with metrics to judge whether a speaker that joins/rejoins the conversation is new or is a previously identified speaker. For these strategies, we are curious to see if neural architectures can sufficiently understand the input in order to perform well on the task set out by the strategy.

2 Related Work

2.1 Traditional Methods

Traditional methods can be categorized into two main groups: top-down and bottom-up. In both cases, neighboring frames are clustered until the frame clusters correspond to different speakers segments. In top-down clustering, the number of clusters starts small and then divided until an appropriate number of clusters are reached. In bottom-up clustering, the number of clusters starts large and then clusters are merged (Xavier and et al., 2010).

Both these approaches are generally based on Gaussian Mixture Models (GMM) and Hidden Markov Models (HMM). Each state in the HMM corresponds to a speaker's unique GMM, which models the features of an audio frame. Transitions between states, then, correspond to transitions between speakers. A recent result for unsupervised speaker diarization using GMMs and HMMs that uses KL divergence for segmentation and Jensen Shanon divergence for clustering achieves 20% accuracy on its output clustering (Madikeri and Bourlard, 2015).

2.2 Deep Learning Methods

Recently, there has been more work applying deep learning to the speaker diarization problem. One piece of related work applies deep neural networks to learn speaker embeddings (Rouvier and et al., 2015). These embeddings are learned from a classification task, in which the network aims to prediction a label for the speaker identity. This approach achieves a 19.25% diarization error rate on audio samples from its corpus of known speakers.

Another standard way to represent speaker identity is using i-vectors (Dehak et al., 2011), which has also been applied to the speaker diarization task (Dupuy et al., 2014).

Unfortunately, treating speaker diarization as a supervised classification task assumes that every speaker appearing in the test set has been seen in the training set. This differs from our task in that it is not online and cannot adapt to new speakers.

Another piece of related work uses triplet loss to train Euclidean embeddings for speaker identity (Bredin, 2016). It applies a BiLSTM RNN to each speaker trained against triplet loss, which, at a high level, aims to create speaker embeddings that maximize the Euclidean distance among any two speakers. We further detail using a triplet loss approach to generate speaker embeddings in the Section 3.

3 Our Approach

We detail the various strategies to tackle the diarization task and the supporting neural architectures. We use Keras (Chollet et al., 2015) for Temporal Classification and Speaker Change Detection, and PyTorch (Various, 2016) for Generating Speaker Embeddings.

3.1 Temporal Classification

One of the earliest approaches we took was, assuming that we have $\leq N$ speakers in a given conversation, seeing if we could classify, for a given timestep, which of at most N speakers was speaking at that timestep, where in this case class 0 corresponds to the first speaker who spoke in that segment, class 1 corresponding to the second speaker who spoke in that segment, and so on. We note that we are not learning to classify *speakers*, but rather class i is the i-th speaker in a particular example. Hence, class i is not the same across all examples (Figure 1). This model tries, to rather than necessarily learn about the speakers, reason temporally about when speaker changes happen.

We explore running various recurrent architectures, including RNNs, LSTMs, GRUs, and stacking recurrent layers over the inputs, and then a final softmax layer that generates output probabilities for each of the N potential speakers.

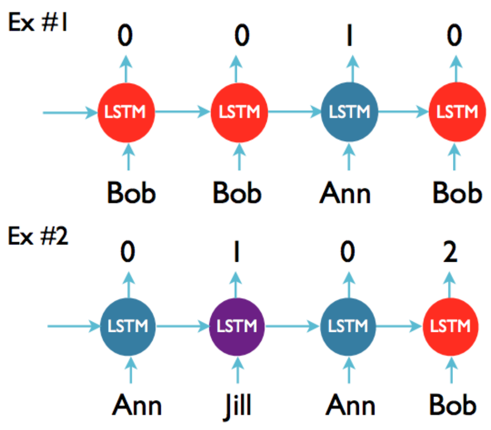

Figure 1: Using a LSTM to temporally segment and classify audio. Note that, across two different examples, the class for Bob is not the same, as we are not identifying the speaker but rather determining what number the speaker is in the context of a conversation.

3.2 Speaker Change Detection

We also consider the approach of simply being able to detect when the speaker has changed. By being able to detect if the speaker has changed effectively, we can segment the audio, and maintain as we segment some notion of the past speakers, and use that to tell if a changed speaker is an entirely new speaker or a past speaker.

Determining if the speaker has changed is a problem that is local with respect to time – a speaker change happens in a matter of a few timesteps. Hence, we experiment with 1D Temporal Convolutions with an affine layer that has a sigmoid activation outputting one value (binary classification of change or not).

3.3 Speaker Embeddings Using Triplet Loss

Finally, we decided to take an approach that explicitly modeled an embedding vector for each speaker. By being able to generate embeddings for different speakers, we can use this to identify both past speakers and when speakers have changed – solving the diarization problem. To generate these embeddings, we were inspired by FaceNet, which produces embeddings for faces using the triplet loss (Schroff et al., 2015).

Our model attempts to train a LSTM that can ef-

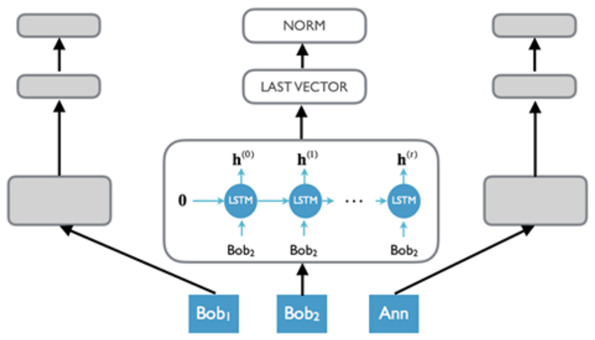

Figure 2: Model using triplet loss to generate speaker embeddings.

fectively encode embeddings for speech segments using the triplet loss. In training, we take segments of audio from different speakers, and construct a triple that consists of (an anchor, a positive example, a negative example) where the anchor and positive example both come from the same speaker but the negative example is from a different speaker. We then want to generate embeddings such that the embedding for the anchor is closer to the positive example embedding by some margin greater than the distance from the embedding for the anchor to the negative example embedding.

The embedding generated by a LSTM is the last hidden step timestep and the embeddings are constrained to live on a unit hypersphere in $\mathbb{R}^n$ (Figure 2).

Our triplet loss is:

$$L = \sum_{i=1}^{N} \max\left(\|\mathbf{h}_x^{(i)} - \mathbf{h}_p^{(i)}\|_2^2 - \|\mathbf{h}_x^{(i)} - \mathbf{h}_n^{(i)}\|_2^2 + \alpha, 0\right)$$

where i is the i-th example, α is a scalar margin, $\mathbf{h}_x^{(i)}$ is the anchor segment embedding, $\mathbf{h}_p^{(i)}$ is the positive segment embedding, and $\mathbf{h}_n^{(i)}$ is the negative segment embedding.

Then, when performing online diarization, we will run windows of speech through the LSTM to create an embedding for this window. If the produced vector is within some distance (using the L2 distance and a tuned threshold) of the stored current speaker vector, we deem that it is the same speaker. Otherwise, we detect that the speaker has changed, and compare the vector with the stored vector for each of the past speakers. The vector for a past speaker may just be the last vector we outputted while we determined that speaker is still speaking. If it is close to any of the past vectors, then we claim the changed speaker is the speaker corresponding to the similar past vector. If none of the past vectors are similar enough, we claim the changed speaker has not been seen before in the conversation, and add it to the learned set of speaker identity vectors.

Our features are MFCC features concatenated with filter bank features.

4 Dataset

4.1 ICSI Meeting Speech dataset

We used two datasets, one of which was the LDC's ICSI Meeting Speech corpus from 2004 (Janin and et al., 2004).

4.1.1 Description

The ICSI Meeting Speech corpus contains 75 meetings from the International Computer Science Institute (ICSI) at Berkeley during 2000-2002. The corpus aimed to to capture the natural conversational dynamics and speaking styles of meetings as best as possible, although each speaker was given a close-talking microphone and was informed of the recording process.

The corpus contains 922 speech files, totaling to about 72 hours of speech. Each meeting's audio is split into multiple audio channels, as each speaker is given a wearable microphone and each meeting room contained 6 table-top microphones.

The entire corpus contains 53 unique speakers. The meetings range from 3 to 10 participants, with an average of about 6 speakers. The corpus also contains many non-native English speakers of varying fluency.

4.1.2 Preprocessing

Each meeting in the ICSI dataset contains a collection of audio channels corresponding to individual speaker microphones during the meeting, which we mixed together into combined meeting audio file.

To generate our labelings for this task, we removed all `NonVocalSounds` segments (such as mic sounds), and `VocalSounds` segments (such as clears throat) from the transcripts. Our final labelings then consist of start and end times for each spoken word segment along with the speaker ID.

4.2 Switchboard dataset

We also used the LDC's Switchboard corpus from 1993 (Godfrey and Holliman., 1993) because we were able to get more samples of higher quality speaker audio. This is desirable as we attempt to create speaker embeddings.

4.2.1 Description

The Switchboard corpus consists of a collection of approximately 2,400 two-sided telephone conversations. These conversations contain 543 speakers from all across the United States, 302 of which were male and 241 female.

The speech is natural, capturing various dialects and speech dynamics found in real-life settings.

4.2.2 Preprocessing

We used the Switchboard corpus to create a dataset of audio clips between 2 to 8 seconds long, tagged by speaker identity. Then, we normalized the volume of each audio clip by dividing each channel of audio by its root-mean square. Finally, we compute the MFCC and filter bank features from the normalized signal and concatenate them.

5 Experiments

5.1 Investigating Clustering Approaches With HMM-GMM

Initially, we explored how well current tools for speaker diarization are able to perform. We specifically work with PyAudioAnalysis (Giannakopoulos, 2015), which performs speaker diarization by using feature extraction, clustering, and smoothing. First, it extracts features for short-term and mid-term segments. For mid-term segments, it uses the average and standard deviations of the MFCC features. It additionally uses probability estimates of whether the speaker is a male or a female using a trained k-Nearest Neighbors (kNN) model.

Once it has its features vectors, it employs k-means clustering to ideally generate a cluster for each speaker with the features (optionally clustering the projections of the features using the Fisher linear semi-discriminant analysis, which can reduce the complexity of the model). If we do not know the number of speakers, the clustering process is repeated for a range of number of speakers, and the Silhouette width criterion (which measures how well objects are in their cluster (Rousseeuw, 1987)) is used to find the best number of speakers/clusters. In addition, HMM smoothing is performed, trying to take into account preceding and succeeding segments, along with the clustering.

To understand the performance of this baseline, we evaluated it on 3 ICSI meetings, each longer than 1 hour. We have evaluated our baseline supplying the model with the number of speakers or without the number of speakers for each meeting.

When the number of speakers is unknown, the model seems to predict that there are only two or three speakers – as indicated by the Silhouette width criterion – whereas the number of speakers actually is closer to 5 or 6. Figure 1 shows the change in the average cluster Silhouette width as we vary the number of clusters, and it seems that the true number of speakers, 5 and 6, got the lowest scores.

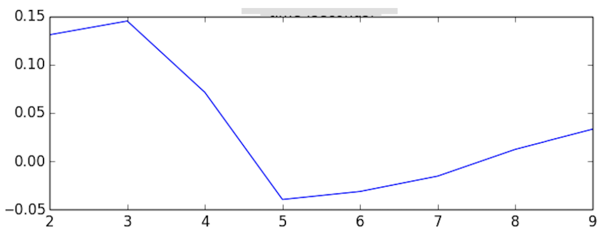

Figure 3: Value of the Average Cluster Silhouette Width (y-axis) as we increase the number of clusters (x-axis).

This seems to indicate more fine grained distinction between speakers is needed, trying to differentiate the few speakers into the separate underlying speakers. To evaluate our baseline results, we use the (average) cluster purity and (average) speaker purity. The average cluster purity provides us an indication of how well a cluster is limited to only one speaker, and the average speaker purity provides a measure of how well a speaker is limited to only one cluster. More formally, as defined by Ajmera et. al, if n_{ij} is the number of frames in cluster i spoken by speaker j, N_s is the total number of speakers, N_c is the total number of clusters, N is the total number of frames, s_j is the total number of frames spoken by speaker j, and c_i is the total number of frames in cluster i, the cluster purity of cluster i is

$$p_i = \sum_{j=1}^{N_s} n_{ij}^2 / c_i^2$$

and the average cluster purity is then

$$\frac{1}{N} \sum_{i=1}^{N_c} p_i c_i$$

Similarly, the speaker purity of speaker j is

$$u_j = \sum_{i=1}^{N_c} n_{ij}^2 / s_j^2$$

and the average speaker purity is

$$\frac{1}{N} \sum_{j=1}^{N_s} u_j s_j$$

(Ajmera et al., 2002).

We summarize our results on the 3 meetings below:

Meeting	Cluster Purity	Speaker Purity
1	61.5%	43.4%
2	53.8%	51.3%
3	55.0%	43.0%

These results give us an understanding of clustering approaches to the problem. Initially we were going to use a clustering based scheme, but we were curious as to whether classification based schemes with neural network architectures. Hence, for our further results, the metric we use is accuracy, where accuracy is what percentage of the timesteps were we able to correctly identify who the speaker is at that timestep.

5.2 Temporal Classification Results

For the Temporal Classification experiments, we use 10 minute meeting segments from the ICSI dataset, classifying up to 10 speakers. We use a little more than approximately 19.2 hours for training, and 4.7 hours for validation. We have a window size of 0.5 seconds, and our windows do not overlap as we want to predict who the speaker is for non-overlapping, discrete timesteps.

5.2.1 Recurrent Architectures

We first experiment with different recurrent architectures for this problem, using softmax loss and filterbank features. We mostly experimented with filter bank features because MFCC features tends to eliminate speaker dependent characteristics (e.g. pitch) which we deem crucial to the task at hand.

We show our results in the table below, including using SVM at each timestep as a baseline.

Model	Train Acc	Val Acc
SVM	23.76 %	19.51%
RNN	30.04 %	22.66%
LSTM	30.17 %	22.62%
Stacked LSTM	30.17%	22.85%
GRU	30.17%	22.88%
Stacked GRU	30.19%	22.78%

We find that, for the most part, the recurrent architectures perform about the same. They all have higher performance than a SVM, which should be expected as a SVM does not have any sense of temporality.

Analyzing what our model does, it seemed that the model did not take into account the features very much when making decisions, and rather, aggregated, for a given timestep t, a sort of notion of what label was likely to be at that timestep across examples. Figure 4 shows the label distribution among our 10 minute segments, and we see that in a given speech segment, most of the talking is done by people who were talking near the start of the segment. This uneven distribution seems to bias our model into, across all timesteps, simply just predicting slightly higher probabilities for the earlier speakers.

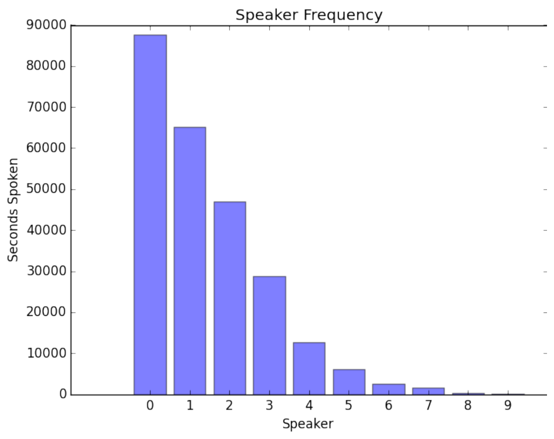

Figure 4: Frequency of speaker position label across the dataset.

5.2.2 Loss Function

We then considered using different loss functions for the problem—perhaps the softmax loss function was not suited to the objective of segmentation we wanted to achieve using classification. With a GRU as the recurrent architecture, which we found had the highest validation accuracy before, we detail some experiments with different classification loss functions:

Loss Function	Train Acc	Val Acc
Softmax	30.17%	22.88%
KL Divergence	30.58%	22.5%
Categorical Hinge	20.31%	20.69%

It seems though, for the most part, we see slightly worse results with different loss functions.

5.2.3 MFCCs

Examining the features, we found that the filter-bank features were highly correlated (as audio signals are temporally)—adjacent filter bank features seemed to have cosine similarities of average of 0.9387, which is maintained across speaker changes. We hypothesized that this high correlation could additionally be contributing to the issue of not really considering the input and rather focusing on a timestep t. Hence, we decided to explore if Mel-frequency cepstral coefficients (MFCCs) could be better features to our Temporal Classification model.

Model	Train Acc	Val Acc
GRU	28.89%	22.23%
Stacked GRU	30.28%	22.75%
Affine ⇒ GRU ⇒ 1D Convolution	31.19%	22.58%

We find that MFCC features also do not improve performance. We conclude that, for speaker diarization, the idea of Temporal Classification is difficult because the model learns more about a given timestep across examples and also it is difficult for such architectures to inherently different states of past speakers—rather they mostly propagate information that is aggregated across all past timesteps.

5.3 Speaker Change Detection Results

We then focus on, instead of necessarily having to both recognize the speaker has changed and remember state, create a model that can simply realize if there has been a speaker change. We can separately maintain a notion of state for each speaker, such as by averaging input features for speakers between changes.

We use the ICSI meeting dataset, segmented into 10 minute chunks using MFCC features, for this task as well. In order to give our model more flexibility, whenever a speaker change happens, we mark the label for speaker change for that timestep, as well as the timestep before and after.

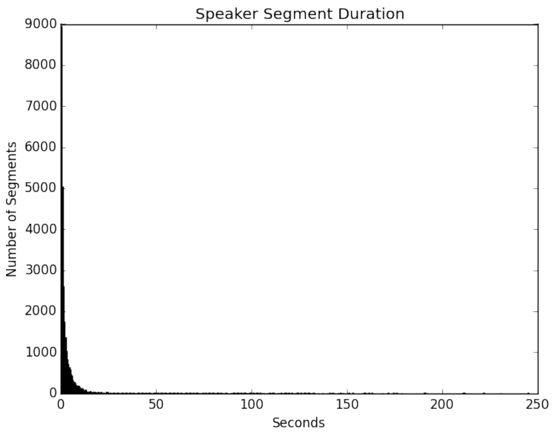

Figure 5: Histogram of duration of speech segments by same speaker.

On the training set, 43.12% of the labels are speaker changes. On the validation set, 52.02% of the labels are speaker changes.

Model	Train Acc	Val Acc
1D Conv	54.69%	48.15%
Affine ⇒ 1D Conv	56.89 %	47.99%
LSTM ⇒ 1D Conv	60.25%	49.21%

Even in this task, we found that our models had difficulty capturing speaker change. As Figure 5 indicates, speakers are mostly speaking for few seconds each time they speak in a conversation – for example, we can imagine a lot of back-and-forth consisting of short segments: "(sentence)" "yeah" "(sentence)" "sure." As a human listener, however, often these short snippets are looked over. This makes the problem of speaker detection very challenging because the model needs to rapidly identify that the speaker has changed and must also do this often. These very quick turns also affect our Temporal Classification model, making it harder for it to learn as well.

5.4 Speaker Embeddings Results

Our dataset consists of a total of 59,357 segments of speech of 2-8 seconds from 216 speakers in the Switchboard dataset. Our embeddings are in $\mathbb{R}^{16}$ and we set our margin $\alpha = 0.2$.

There are two ways we split the dataset for our experiments. In the *same-group* split, all 216 speakers are equally represented across the training and validation sets; only the specific audio segments are different. In the *different-group* split, we split the speakers roughly in an 80:20 ratio (173 speakers in the training set and 43 speakers

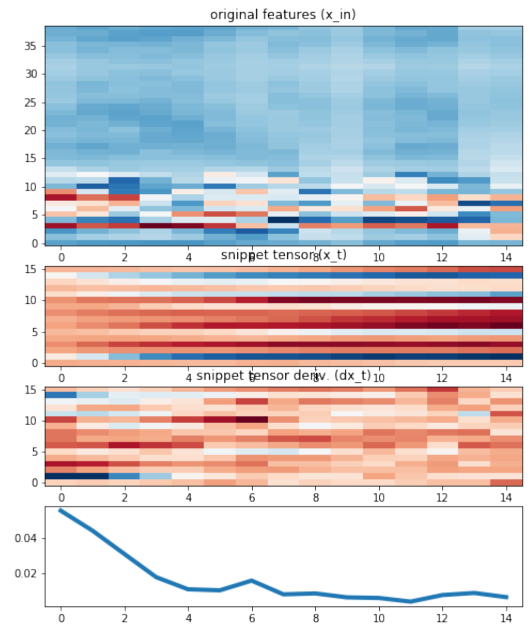

Figure 6: Visualization of speaker embedding. *1st:* Original features (MFCC, log filterbank). *2nd:* Embedding vectors. *3rd:* Delta embedding vectors, i.e. change between successive frames. *4th:* Magnitude of delta embedding vectors.

in the validation set); therefore, on validation the model encounters a completely new set of speakers. We found from our experiments that the same-group split training method (Figure 7) produces better accuracy than the different-group split training method (Figure 8). This is likely because the difference between speech taken from the same speaker is smaller than that for speech taken from different speakers. Thus, the model learns more meaningful information by encountering as many speakers as possible.

Another experiment was how to generate a single speaker embedding vector given the generated embedding vectors at each timestep. One option is to average the vectors over all timesteps. Another option is to use the last vector generated. We found that using the last vector produced significantly smoother trajectories, of the kind seen in (Figure 6). At the beginning of a segment, the embedding vector is changing quickly and then quickly stabilizes. This is in contrast to the dynamics for the averaging option, where we found the embedding vector to oscillate a lot more. This produces a noiser signal that is less useful for diarization.

We detail below the training and validation accuracy of the two different groups, where accu-

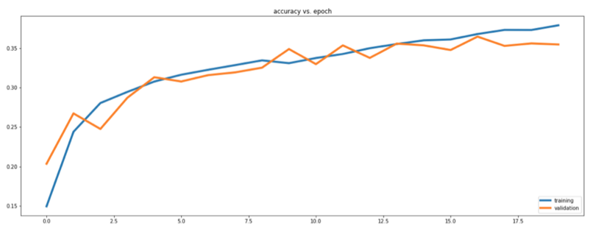

Figure 7: Triplet accuracy for same-group train/val split. Each speaker is equally represented in train and validation sets.

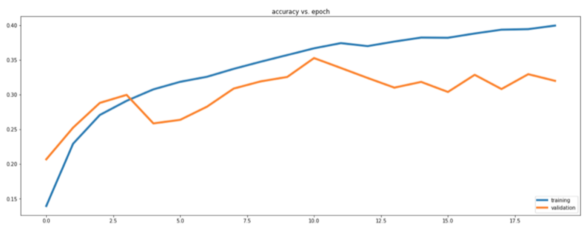

Figure 8: Triplet accuracy for different-group train/val split. Completely different speakers in validation set.

racy represents what percentage of our examples had a squared Euclidean distance between the anchor embedding and positive embeddings that was at least $\alpha = 2$ smaller than the squared Euclidean distance between the anchor embedding and the negative embedding.

Split	**Train Acc**	**Val Acc**
Same group	37.91%	35.47%
Different group	39.97%	32.00%

We show in Figure 9 how embedding vectors would be applied to diarization, in this case, on a synthetically generated audio example. The first plot shows audio features, the second shows the speaker ID at each timestep (e.g. this dialog has

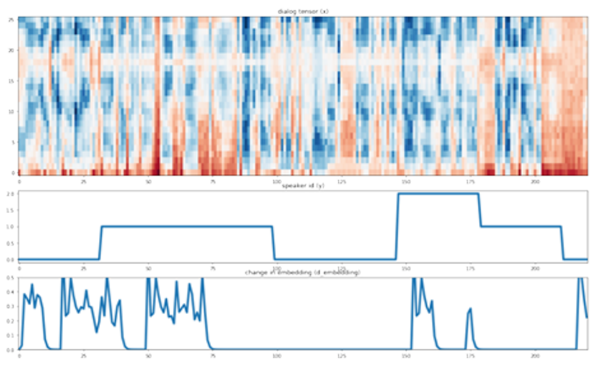

Figure 9: Synthetic conversation example and diarization signal.

3 distinct speakers), and the last plot shows the change in the embedding vectors over time. Notice that the embeddings change most at around the speaker transitions. This is a promising approach for future work.

6 Conclusion

We learned that the neural network architectures we explored were not able to sufficiently capture the objectives of our speaker diarization task. In particular, our models weren't able to capture the higher level, complex features that humans often rely on to distinguish different speakers. While humans do use low level features, such as the distinction between male and female, different accents, and pitch to distinguish speakers, we also use higher-level, context-specific features, such as when a speaker pronounces specific words in a manner we find unusual or interesting, or when we notice a speaker with a pronounced breathing pattern.

We believe our models weren't able to capture these richer, higher-level features in our experiments so far. A critical reason our task was so difficult was its online nature, where the number of speakers and speakers themselves were not known at test time. Hence, temporal classification and our embeddings suffered when interacting with new speakers outside of the training set, while our speaker change detection did not have enough representational power to distinguish speakers with high accuracy.

For future work, we want to further pursue our diarization system via the word embeddings generated via triplet loss training. We also think further innovations in the loss function can help us train models to better understand temporal relationships and speaker states. Similar to how using CTC loss can improve the performance of ASR systems, we posit that a similar mechanism that captures temporal relationships will improve our diarization systems. Lastly, pairing our diarization system with transcripts and a language model can help us further highlight distinctions in pronunciation patterns and let us utilize distinctions in speaking styles, which is an approach we think can lead to much better results.

Acknowledgments

We would like to thank Prof. Andrew Maas and Jiwei Li for instructing the course this quarter and for giving us the opportunity to perform this research, as well constant feedback and support.

References

Jitendra Ajmera, Hervé Bourlard, Itshak Lapidot, and Iain A McCowan. 2002. Unknown-multiple speaker clustering using hmm. Technical report, IDIAP.

Hervé Bredin. 2016. Tristounet: Triplet loss for speaker turn embedding. *arXiv preprint arXiv:1609.04301* .

François Chollet et al. 2015. Keras. https://github.com/fchollet/keras.

Najim Dehak, Patrick J Kenny, Réda Dehak, Pierre Dumouchel, and Pierre Ouellet. 2011. Front-end factor analysis for speaker verification. *IEEE Transactions on Audio, Speech, and Language Processing* 19(4):788–798.

Grégor Dupuy, Sylvain Meignier, Paul Deléglise, and Yannick Esteve. 2014. Recent improvements on ilp-based clustering for broadcast news speaker diarization. In *Proceedings of Odyssey*. pages 187–193.

Theodoros Giannakopoulos. 2015. pyaudioanalysis: An open-source python library for audio signal analysis. *PloS one* 10(12).

John Godfrey and Edward Holliman. 1993. Switchboard-1 release 2 ldc97s62. web download. philadelphia: Linguistic data consortium .

Adam Janin and et al. 2004. Icsi meeting speech ldc2004s02. web download. philadelphia: Linguistic data consortium .

Srikanth Madikeri and Hervé Bourlard. 2015. Kl-hmm based speaker diarization system for meetings. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, pages 4435–4439.

Peter J Rousseeuw. 1987. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics* 20:53–65.

Mickael Rouvier and et al. 2015. Speaker diarization through speaker embeddings. *Signal Processing Conference (EUSIPCO), 2015 23rd European* .

Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. pages 815–823.

Various. 2016. Pytorch. https://github.com/pytorch/pytorch.

Anguera Xavier and et al. 2010. Speaker diarization: A review of recent research. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* .