A Two-Stage LSTM Based Approach for Voice Activity Detection with Sound Event Classification

1st Yarong Feng
Amazon
yarongf@amazon.com

2st Zongyi(Joe) Liu
Amazon
joeliu@amazon.com

3st Yuan Ling
Amazon
yualing@amazon.com

4st Bruce Ferry
Amazon
bferry@amazon.com

***Abstract*—We introduce a two-stage approach using LSTM for voice activity detection with sound event classification. This approach proves to be effective when training data is limited. Moreover, it achieves better performance than pre-trained model using large-scale data set (AudioSet). Apart from clip-level accuracy, we also introduce two metrics for evaluating overall audio segmentation accuracy: mean IoU, and mean front miss. On test set, our method achieves 98% accuracy, 0.95 mean IoU for speech and 0.99 mean IoU for music, and 0.03 mean front miss for both speech and music.**

***Index Terms*—voice activity detection, LSTM, sound event classification, audio signal processing**

I. Introduction

The problem we consider in this paper involves audio recordings of the dialogue between a voice assistant (such as Alexa in Echo Dot) and a "user" collected in controlled lab experiments. [1] The first $20 \sim 30$ seconds of each dialogue is recorded. Fig. 1 shows the lab setup and Fig. 2 shows the waveforms of two example audio recordings. In our use case, such a dialogue usually follows this pattern: wake word(user) $\rightarrow$ question/request(user) $\rightarrow$ acknowledgement of receiving the question(voice assistant) $\rightarrow$ actual answer/content(voice assistant). For example, the first plot in Fig. 2 corresponds to this dialogue:

- User: Alexa, add bananas to my shopping list.
- Alexa: You already have bananas on your shopping list.

and the second one corresponds to the following:

- User: Alexa, play wheels on the bus.
- Alexa: The wheels on the bus go round and round by the countdown Kids on Amazon music.
- Alexa: (music playing).

Given these audios, the task is to locate and classify each component of the dialogue. By locating a component, we mean providing the start and end times of the component, such as $[2.30s, 7.65s]$. By classifying a component, we mean providing a label for it, with value in ["speech", "music"]. With components of a dialogue located and classified, we can evaluate Alexa's performance over a wide range of metrics, such as response latency, music latency, etc.

[1]The user's voice is synthesized. There is no recording of any actual conversation between customer and voice assistant involved.

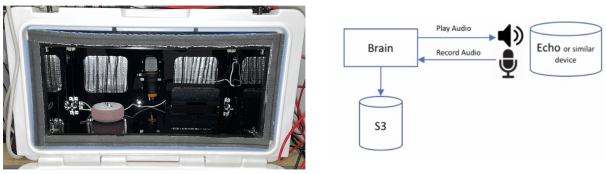

Fig. 1. Left: lab setup. Right: System diagram.

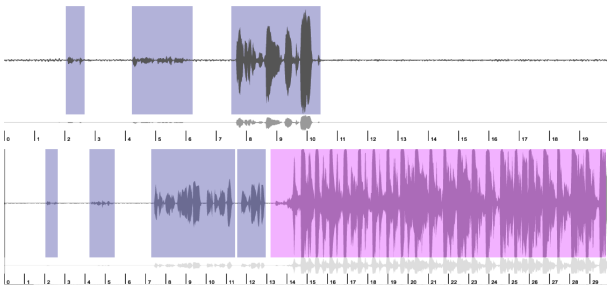

Fig. 2. Waveforms of two example dialogues. Blue blocks are speech components. Pink blocks are music components.

For the task at hand, we do not differentiate between the voice of the user and that of the voice assistant, and consider both as "speech". When the voice assistant acknowledges receiving the user's question by a chime sound, we also classify the chime sound as "music".

As an example, the desired output for the top waveform in Fig. 2 would be:

Listing 1. example output of an audio recording

```
[{class: speech, start: 2.0, end: 2.7},
 {class: speech, start: 4.2, end: 6.2},
 {class: speech, start: 7.5, end: 10.4}]
```

With components of a dialogue located and classified, we can evaluate a voice assistant's performance, or compare across different voice assistants/devices/versions, over a wide range of metrics, such as response latency, music latency, etc. However, in order to draw sound conclusions regarding these latency metrics, measurement with very high accuracy is a prerequisite, because most latency metric values are small. As an example, Table I shows the response latency of a voice assistant from two devices. Roughly speaking, if a 10% margin of error is allowed, then to accurately evaluate

the response latency of device A, we need to locate each component in the audio recording to be within 162ms from the ground truth. If we want to measure the difference in response latency between these devices, it requires each component to be located within 25ms from the ground truth. Such high requirement on measurement accuracy calls for algorithms with high-precision.

TABLE I
RESPONSE LATENCY COLLECTED FROM LAB EXPERIMENTS.

Metric	device A	device B
response latency P50 (second)	1.62	1.87

A. Related work

Voice activity detection(VAD) [4] is a common task performed in many audio processing systems. Sound event localization and detection(SELD) [1] takes one step further, and tries to not only detect voice activities but also to describe them. As SELD result is usually more complex than that of audio tagging, objective evaluation of SELD performance is itself an ongoing research topic [6]. Thanks to the release of AudioSet [3], pre-trained deep learning models on large-scale data set [5] becomes available for audio tasks now. In this paper, we describe an end-to-end algorithm to detect and classify sound events, as well as introduce two metrics to evaluate its performance. We compare the performance of proposed algorithm to a pre-trained model(CNN14_DecisionLevelMax) introduced in [5].

II. DATA

We work with a dataset of roughly 300 raw audios, with sampling rate of 44100Hz and duration of $20 \sim 30$ seconds. For each audio, we manually labeled the ground truth, in the format as given in Listing 1.

The dataset is randomly split into 90% training and 10% validation. A separate test set with ~ 100 audios is held out for final evaluation of model performance in production.

III. METHODOLOGY

A. Data Preprocessing

The raw audios are first filtered by volume. Only audios for which the volume of all components is between $66dB$ and $86dB$ are kept, where the thresholds are determined based on the volume distribution of the training data.

Filtering by volume is necessary for our use case, as the raw audios come from different labs with (potentially) different hardware setup. In some labs, the volume could be so loud that it saturates the recorder, while in other labs so low that it's almost covered by background noise. Moreover, the volume in these labs could change frequently due to software update, hardware setup adjustment, etc. If we rely on a specific volume pattern in the training data, the model's performance will degrade significantly once that pattern changes, which is common as explained.

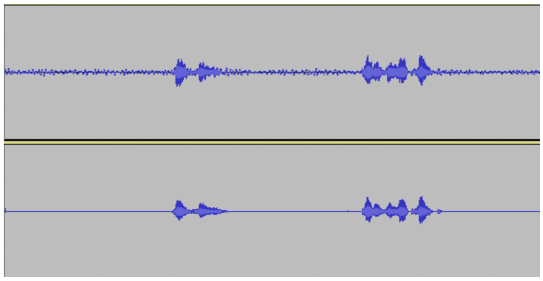

Fig. 3. Effect of highpass filter to a typical audio recording. Top: unfiltered. Bottom: filtered.

The audios with valid volume are then de-noised, as they sometimes contain non-negligible noise. This is necessary as the noise pattern varies by lab and/or hardware, and could change frequently due to the same reason for volume.

Experimental results show that a highpass filter with low frequency threshold at 300Hz works well for our use case. The choice is also supported by [7]. The authors believe the reason is two-fold: 1) most of the noise in our audios occur in the low frequency domain (as shown in Fig. 3), 2) keeping more high frequency signal greatly boosts the model's ability to separate music from speech, as the speech signals we deal with usually contain lower frequencies compared to music.

B. Feature Generation

With preprocessed and filtered audios, we cut each into short clips using a sliding window (hop size = 2205 data points or 0.05 second, window length = 4096 data points or 0.09 second). For each short clip, features and ground truth label are computed as described below.

For each clip, we compute the following frequency domain features: MFCC (Mel-frequency cepstral coefficients) in 80 bins, chroma features in 12 bins, spectral centroid/flatness/rolloff, tonal centroid features, and spectrogram RMS (Root Mean Square). We also compute the following time domain features: zero-crossing rate and RMS.

Annotated ground truth exists for each audio, in the format of Listing 1. To get the ground truth label for a specific clip, we differentiate between the following 3 cases:

1) it is completely covered by a ground truth component;
2) it is partially covered by a ground truth component;
3) it is not covered by any ground truth component.

In case 1, the clip gets the label of the covering component. In case 3, the clip gets the label for background. In case 2, the clip gets the label of the covering component if more than 50% of the clip is covered, otherwise it gets the label for unknown. Therefore, at the clip level, we have 4 classes, namely: "background", "speech", "music", and "unknown". Note that in case 2, the threshold of 50% guarantees that each clip can be covered by at most one ground truth segment, which in turn guarantees the uniqueness of the ground truth label for each clip.

Fig. 4 provides an illustration. For each of the four clips, a vector of features is generated. As to ground truth, clip

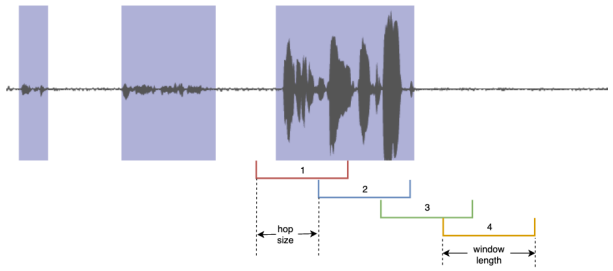

Fig. 4. An illustration of the process of feature and ground truth generation using sliding windows.

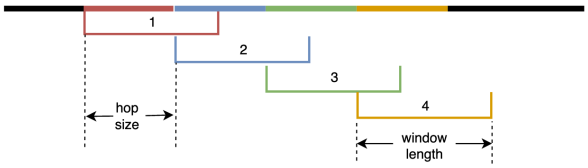

Fig. 5. Generate label for entire audio from overlapping clip labels. Color of each clip represents the predicted label.

1 and 2 gets label "speech" as they are covered by more than 50%, and fully, respectively, by a ground truth speech segment. Clip 3 gets the label "unknown" as it's less than 50% covered by a ground truth segment, whereas clip 4 gets the label "background" as it's not covered by any ground truth segment.

At prediction time, the use of overlapping clips calls for special treatment. With predicted labels generated for each clip, one needs to decide how to combine the labels from overlapping clips to produce a final prediction for the entire audio, in the format of Listing 1. For our use case, the overlap between adjacent clips is less than 50%. So, given the predicted label for a clip i with index in

$$[i * hop\_size, i * hop\_size + window\_length),$$

we apply it only to the beginning portion

$$[i * hop\_size, (i+1) * hop\_size),$$

which is non-overlapping with clip $i+1$. An illustration is provided in Fig. 5. However, if the overlap between adjacent clips is significant, one may need to combines the information from all nearby clips, as in III-D.

C. Stage 1: Dimension reduction via classification

After feature generation, we have converted each raw audio into several short clips, each with a feature vector and a ground truth label. With this, we train the first model(denoted by stage1 classifier) using AutoGluon-Tabular [2]. Before fitting stage1 classifier, we address class imbalance issue by upsampling under-represented classes.

Although AutoGluon-Tabular provides a convenient framework for model training with decent clip-level accuracy, its segmentation result is not great(as shown in Table II). It is because AutoGluon-Tabular treats our dataset as pure tabular. That is, the base learners are not able to capture the temporal dependency among clips that belong to the same audio. For example, if clip i, clip $i+1$, and clip $i+2$ belong to the same audio, and both clip i and clip $i+2$ are predicted as "music", then it should be more likely than usual that clip $i+1$ is also predicted as "music". This is not easy with tabular classifiers, unless information on adjacent clips are used as features, as they usually treat different data points as completely independent.

D. Stage 2: Aggregation

Recurrent neural network is a natural choice to incorporate the temporal dependency. In particular, we choose bi-directional Long Short-Term Memory(LSTM). As LSTM works with sequence data, we convert the array of clips belonging to an audio into temporal sequences, again using the sliding window technique (hop size = 20 clips or 1 second, window length = 60 clips or 3 seconds). We construct ~ 4000 temporal sequences for training.

Due to limitation on training data, instead of working with high-dimensional feature vectors, we use only the output from stage1 classifier as the input to LSTM. In other words, stage 1 classifier is used for dimension reduction. We choose a 2-layer LSTM with 256 hidden states. The model is trained for 100 epochs, with a batch size of 128 and start learning rate of 0.0001. Learning rate is scheduled to decrease by 25% for every 10 epochs.

LSTM generates a label for each clip in each temporal sequence. Note that the way we construct the temporal sequences result in $> 60\%$ overlap. Therefore, due to reason stated in III-B, the final label for a clip is taken to be the majority vote from all temporal sequences covering this clip.

E. Segment construction and post-processing

For each audio, based on the clip-level classification and aggregation results, we simply consider each continuous portion of clips with the same predicted label as one segment. Although we have used LSTM to smooth the clip-level predictions, the output label could still contain very short segments within long ones that are misclassified. We applied a simple post-processing logic to further smooth out such short segments, by 1) merging two adjacent segments if the gap is < 0.1 second, 2) removing isolated segments shorter than 0.3 second, and 3) relabeling segments shorter than 0.3 second within long ones. This is optional, but we find it necessary for our use case to get robust segmentation results. Note that the threshold values are chosen based on performance on our specific test set, and should be re-evaluated (e.g., by cross validation) for other use cases.

IV. Evaluation Metric

As we discretize the raw audio into clips, it's natural to measure clip-level performance as for a classification problem. However, good clip-level performance doesn't always translate

to good segmentation result (i.e., estimating the start and end time of each component accurately). Therefore, it's critical to develop metrics that measure segment-level performance properly as well.

A. Clip-level performance

We use the usual metrics for multi-class classification problem. In particular, we consider overall accuracy, class-level accuracy, precision, and recall.

B. Segment-level performance

We have observed that ground truth data in our dataset contain some level of subjectivity. For instance, in the ground truth data, a component sometimes can contain short gaps between words or sentences longer than the hop size of $0.05s$. In prediction time, these gaps may be treated as background and not included in any detected component. As an example, the 3rd and 4th color blocks in the bottom plot of Fig. 2, may correspond to the same ground truth component. As a result, in order to define a fair and robust metric, it's necessary to first match and group predicted segments with ground truth segments.

1) Match predicted and ground truth segments: For an audio, for each non-background class, given n ground truth segments $[g_1, ..., g_n]$ and m predicted segments $[p_1, ..., p_m]$, do:

noitemsep,topsep=0pt

1) go over each pair of g_i and p_j, and mark them as a match if

$$intersection(g_i, p_j) > t * min(len(g_i), len(p_j)),$$

where t is chosen to be 0.5, and $intersection(\cdot)$ is the usual interval intersection operation. This condition allows one predicted segment to be matched to multiple ground truth segment, and vice versa, as illustrated in Fig. 6. This could help alleviate the subjectivity contained in ground truth label. A special case is when $intersection(g_i, p_j) > 0.5 * len(g_i)$, it implies that g_i can not be matched to any other predicted segment.

2) create groups of ground truth segments and predicted segments based on the matching results in step 1:
noitemsep,topsep=0pt
   - for each g_i, find all its matched predicted segments p_j and add both to the group;
   - for each matched predicted segment p_j, find again all its matched ground truth segments g_k and add it to the group;
   - continue until no new predicted segment or ground truth segment is found.

Using the examples in Fig. 6, in case 1, g_1 and p_1 is a group; in case 2, g_1 and p_1, p_2 is a group; in case 3, no group is formed; and in case 4, g_1, g_2 and p_1 is a group.

2) False positive and false negative: If a predicted segment is not matched to any ground truth segment, it is a false positive. In Fig. 6, p_1 in case 3 is a false positive. If a ground truth segment is not matched to any predicted segment, it is a false negative. In Fig. 6, g_1 in case 3 is a false negative.

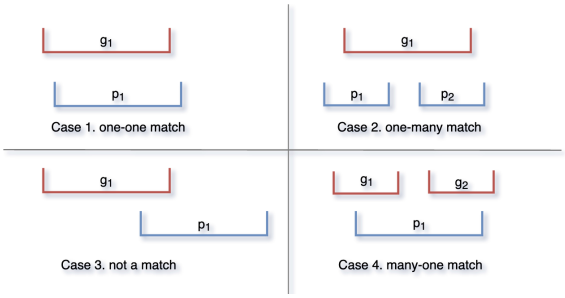

Fig. 6. Illustration of four cases of segment match.

3) Mean segmentation IoU: We borrow the concept of IoU(Intersecion over Union) that's commonly used to evaluate object detection tasks in computer vision, and modify it for our use case. For each matching group of predicted and ground truth segments, compute its segmentation IoU as follows.

noitemsep,topsep=0pt

1) For each pair of (g_i, p_j), compute the pairwise intersection. Then compute the union of all pairwise intersection, and denote it by I;
2) For each pair of (g_i, p_j), compute the pairwise union. Then compute the union of all pairwise union, and denote it by U;
3) the segmentation IoU for this group is taken to be I/U.

The mean segmentation IoU for an audio is simply the average of the group segmentation IoUs across all the matching groups. The mean segmentation IoU for the data set can be computed by averaging across all audios.

4) Mean front miss: Another useful metric, namely the mean front miss, is to measure how close the start time of a ground truth segment is to that of its matched predicted segment(s). As we usually calculate latency metrics, only the start time of a segment matters. For each matching group of ground truth segments and predicted segments, calculate its front miss as follows.

noitemsep,topsep=0pt

1) compute the minimum start time of ground truth segments in the matching group, denoted by A;
2) compute the minimum start time of predicted segments in the matching group, denoted by B;
3) the front miss is taken to be $|A - B|$.

The mean front miss for an audio is simply the average of the group front misses across all the matching groups.

V. Results

We use two datasets to evaluate the model performance: test set and production test set. They differ in the following aspects: 1) the test set consists of audios collected from the same voice assistants, labs, languages, and question list as the training data, while the production test set may contain new voice assistants/lab/language/question; 2) audios in the test set has been filtered by volume in the same way as for training data, but the production test set contains unfiltered

raw audios; 3) audios in the production test set are collected from recent production failures, and usually contain stronger noise than those in the test set. We believe preformance on the production test set can reflect the model's generalization capability and robustness to unseen audios collected in less than ideal environment.

We compare the performance of four methods: 1) two-stage(AG+LSTM), 2) one-stage(AG), 3) one-stage(LSTM), 4) pre-trained(CNN14_DecisionLevelMax). Two-stage(AG+LSTM) is the proposed method. One-stage(AG) uses only the results from stage 1 classifier. One-stage(LSTM) uses a LSTM trained from scratch with raw features in the same setting as in two-stage LSTM. Pre-trained(CNN14_DecisionLevelMax) [5] is a model trained using AudioSet. Results are summarized in Table II. Table III shows the detailed performance by class of the proposed method: two-stage(AG+LSTM).

TABLE II
PERFORMANCE COMPARISON AMONG FOUR METHODS.

Method	**accuracy**		**mean IoU**	
	test	prod test	test	prod test
AG+LSTM	98.2%	93.4%	0.96	0.93
AG	97.8%	88%	0.79	0.87
LSTM	87%	87%	0.88	0.88
CNN14	96.5%	94%	0.61	0.72

TABLE III
DETAILED RESULTS OF TWO-STAGE(AG+LSTM).

Metric	**Test set**		**Prod test set**	
	speech	music	speech	music
Precision	98.2%	99.6%	88.9%	99.5%
Recall	97.4%	99.7%	94.7%	92.6%
Mean IoU	0.95	0.99	0.93	0.92
Mean front miss	0.03	0.03	0.06	0.3
False positives	2	0	22	0
False negatives	12	0	0	4

Comparison. Both AG and the pre-trained model have decent clip-level performance on test set, but poor segment-level results. This is to be expected as both ignore temporal dependency among clips. LSTM, when trained from scratch using raw feature vectors, performs poorly. However, its mean IoU is higher than AG, due to the sequential nature of the architecture. The proposed method, AG+LSTM, achieves the best performance among the four, by adding a dimension reduction step prior to fitting LSTM.

Overall, the proposed method achieves 98.2% accuracy on the test set, and 93.4% on the production test set. Note that the precision for "speech" drops from 98.2% on test set to 88.9% on production test set. Not surprisingly, the recall for "music" drops significantly from test set to production test set, along with its mean front miss increasing by ten times, from 0.03 second to 0.3 second. It is because in the production test set, there are many music segments that start with low prelude. As we mainly rely on frequency domain features to tell the two classes apart, it becomes challenging for the model when music only contains sound with low frequency components, which is similar to speech. In Fig. 7 and 8, we show two examples to illustrate. Fig. 7 contains audio waveform and model prediction for a dialogue where the user asks the voice assistant to play 'Baby Shark'. The beginning part of the song is identified as "speech" as it's drum. On the contrary, Fig. 8 shows a dialogue where the user asks for harpsichord music. There is no misclassification in this case as the frequency of harp sound is much higher than human voice.

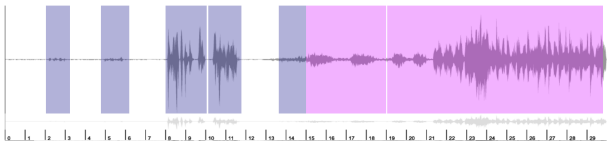

Fig. 7. Audio waveform and model prediction of a dialogue with the last segment being 'Baby Shark'.

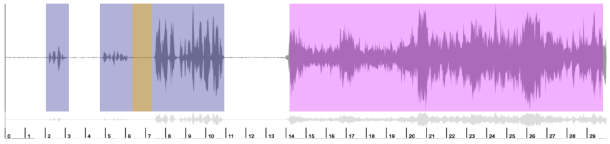

Fig. 8. Audio waveform and model prediction of a dialogue with the last segment being harpsichord music.

VI. FUTURE WORK

Although the proposed method has good performance, there is still room for improvement. In particular, we would like to better differentiate music from speech, especially music with mainly low frequency sound. We think fine tuning the pre-trained models introduced in [5] is a promising path. However, we also believe there will be a limit to this, as certain music genre, such as rapping, naturally contains very little instrument sound but mainly human voice. For such cases, we should rely on the rhythm and beat more, instead of the frequency. We will explore features and model architectures that can take this into consideration.

ACKNOWLEDGMENT

The authors would like to thank the AXA engineers for their help in collecting and labeling data.

REFERENCES

[1] Sharath Adavanne, Archontis Politis, Joonas Nikunen, and Tuomas Virtanen. Sound event localization and detection of overlapping sources using convolutional recurrent neural networks. *IEEE Journal of Selected Topics in Signal Processing*, 13(1):34–48, 2018.

[2] Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. Autogluon-tabular: Robust and accurate automl for structured data. *arXiv:2003.06505*, 2020.

[3] Jort F Gemmeke, Daniel PW Ellis, Dylan Freedman, Aren Jansen, Wade Lawrence, R Channing Moore, Manoj Plakal, and Marvin Ritter. Audio set: An ontology and human-labeled dataset for audio events. *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017.

[4] Jayaprakasha Honnatteppanavar and BG Nagaraja. A review of the recent advances in voice activity detection techniques for speech processing. *JOURNAL OF CRITICAL REVIEWS*, 7, 2020.
[5] Qiuqiang Kong, Yin Cao, Turab Iqbal, Yuxuan Wang, Wenwu Wang, and Mark D Plumbley. Panns: Large-scale pretrained audio neural networks for audio pattern recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 28, 2020.
[6] Annamaria Mesaros, Toni Heittola, and Tuomas Virtanen. Metrics for polyphonic sound event detection. *Applied Sciences*, 6(6), 2016.
[7] Wikipedia contributors. Voice frequency — Wikipedia, the free encyclopedia, 2021.