Voice activity detection for low-resource settings

(Speech recognition)

Abhipray Sahoo
Department of Electrical Engineering
Stanford University
`abhipray@stanford.edu`

1 Introduction

The challenge of voice activity detection (VAD) is to detect the presence of human speech in an audio signal containing speech and noise. Typical VAD methods include feature-engineering based on the power spectral density and periodicity; they are combined with learned statistical models such as gaussian mixture models (GMM) or artificial neural networks for classification. There are several applications of a VAD eg. as a front-end for automatic speech recognition to gate out non-speech sounds or to switch coding schemes in codecs like Opus [1]. These applications are usually real-time and thus require fast low-latency VADs. A popular implementation of a VAD is part of WebRTC which uses a GMM model [2]. It is computationally efficient and can run in low-resource settings but does not have the best accuracy. We improve on existing methods like the WebRTC using a deep learning approach while constraining our implementation to have low computational complexity so that it can run on a microcontroller and be low latency for real-time applications.

2 Related work

A good baseline for performance is the WebRTC VAD implementation [2]. It uses GMM models of speech and non-speech sounds with input features as log energies of six frequency bands between 80Hz-4000Hz. It uses fixed point operations and is optimized for real-time use for web transmission.

The work done by Ko et al. [3] to quantize deep neural network weights demonstrates that it is possible to improve on the WebRTC performance with lower latency. The optimized network has low memory footprint as well as VAD error rate of only 14.1% compared to WebRTC's 20.88%.

The work by Drugman et al. [4] does a mutual information analysis of different features, some source and some filter-derived as well a fusion of features. The features are used as input to a neural network for classification and performs better than state of the art (24% accuracy across all conditions). While feature engineering led to good results here, we believe using low-level features could force a deep network to learn the best features from audio for the task.

3 Dataset and Features

We use the VCTK dataset for the speech samples [5]. It contains English utterances by 109 speakers amounting to roughly 44 hours of audio. From the Noisex-92 database [6], we add four types of noise: babble, car, factory and white noises. The noise is added at two SNR levels: 0dB and 10dB. The total number of hours in the dataset is 396 hours.

The full dataset is split into train, validation and test sets with a 85-10-5% split. This results in roughly 20 hours of audio in the test set with uniform distribution of noise conditions.

The audio is downsampled to 16KHz with 16-bit resolution. 40 log-energies on the mel-scale are extracted from 32ms windows with 16ms overlap and hanning window. These are the features used for the deep neural network. By simple energy thresholding, sequences of pairs of input features and output labels are generated. The threshold for generating positive labels was selected as a value between the average RMS and minimum RMS value across 32ms frames of the audio signal.

We also calculate sample weights for each input sequence, assigning 0 to timesteps that have been padded. Positive labels are given weight corresponding the the number of positive class samples and similarly for the negative class.

4 Baseline

Our baseline algorithm is the WebRTC VAD. Using the WebRTC VAD, we generated predictions on VCTK original and noisy audio versions and compared against the ground truth labels. The optimizing metric is the F1 score for classification. We care about reducing the false positive rate while keeping the true positive rate high. The table below summarizes the results for different noise conditions. FPR and TPR are the false positive rate and true positive rate respectively. F1 score is the harmonic mean of precision and recall.

	F1 score	FPR	TPR	Precision	Recall
babble_0dB	0.617	0.993	0.998	0.447	0.998
babble_10dB	0.737	0.349	0.838	0.657	0.838
factory1_0dB	0.623	0.965	0.990	0.454	0.990
factory1_10dB	0.752	0.239	0.780	0.726	0.780
raw	0.819	0.228	0.894	0.756	0.894
volvo_0dB	0.667	0.735	0.944	0.515	0.944
volvo_10dB	0.804	0.231	0.860	0.755	0.860
white_0dB	0.748	0.198	0.744	0.752	0.744
white_10dB	0.766	0.208	0.777	0.755	0.777

As observed, addition of noise severely degrades performance. It also shows the high false positive rate in the presence of noise.

5 Deep Neural Network

We train a recurrent deep network architectures on sequences of feature and output classification labels using a frame-wise binary cross-entropy loss. An RNN is well suited for this problem since it can model the time dependence of speech in a noisy signal. We use a three-layered GRU network since they are comparable to LSTM units but require fewer computations [7]. Each GRU layer is followed by a batch normalization layer. A final dense layer with a single perceptron with sigmoidal activation outputs the probability of speech.

The model is trained with binary cross entropy loss using the adam [8] optimizer. During training, sample weights are used as described in the section on dataset and features.

Our goal is to minimize the size of the model while maintaining better-than-baseline performance. A hyperparameter search was done with a Bayesian optimization framework [9] for layer sizes, dropout and learning rate. The GRU model was trained with a hyperparameter search for network size in three stages. In each of the three stages, 30 trials with different combinations of hyperparameters were performed. In the first stage, the three layers were limited to a maximum of 128 nodes. The best performing model gained a validation F1 score of 0.923. In the second stage, the layers were capped to a maximum of 32 units leading to the best model validation F1 score of 0.915. In the last stage, the layer sizes were capped to a maximum of 16 units. The best model among the thirty trials had 13, 10 and 4 GRU units in the three layers. It scored 0.908 F1 score.

The smallest model performs reasonably well. We trained it for another 150 epochs increasing the validation F1 score to 0.921. The final model is shown in figure 1.

The final model uses only 3200 parameters. This amounts to 12,800 bytes using 32-bit floating point weights. We use tensorflow-lite [10] to compress the network to 16-bit floating point weights, reducing the memory footprint in half.

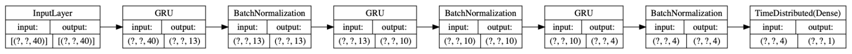

Figure 1: Final model architecture

6 Results

The final model was then evaluated on the test set across the different test conditions.

	F1 score	FPR	TPR	Precision	Recall
babble_0dB	0.880	0.034	0.883	0.876	0.883
babble_10dB	0.932	0.013	0.915	0.949	0.915
factory1_0dB	0.873	0.033	0.873	0.874	0.873
factory1_10dB	0.927	0.014	0.911	0.944	0.911
raw	0.958	0.007	0.942	0.974	0.942
volvo_0dB	0.938	0.010	0.917	0.960	0.917
volvo_10dB	0.947	0.011	0.937	0.958	0.937
white_0dB	0.909	0.018	0.888	0.931	0.888
white_10dB	0.941	0.009	0.918	0.964	0.918

From the table, we can see that the model can work well even in the high noise setting of 0dB. It outperforms the baseline on every single metric for all conditions. Plot 2 shows a comparison of the two on a precision-recall plot. From this, it is clear that the GRU model has higher F1 scores for every noise condition.

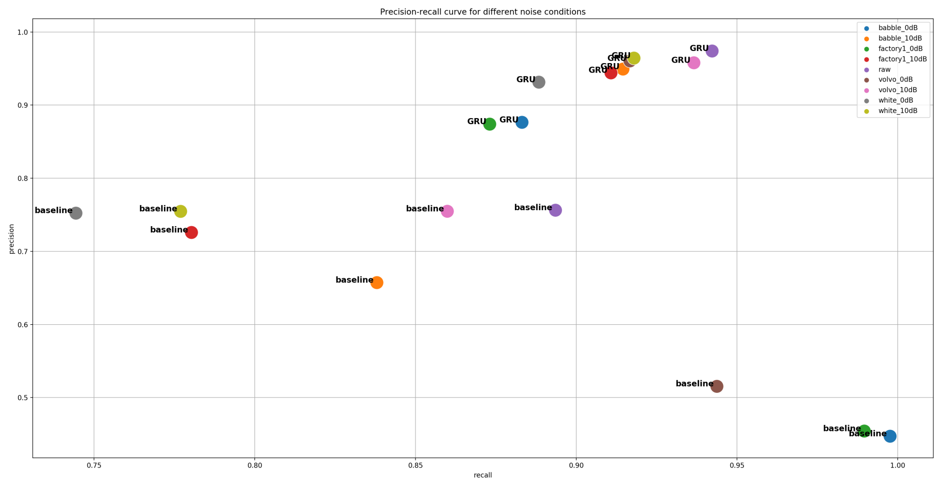

Figure 2: Precision-recall comparison of the baseline vs RNN model

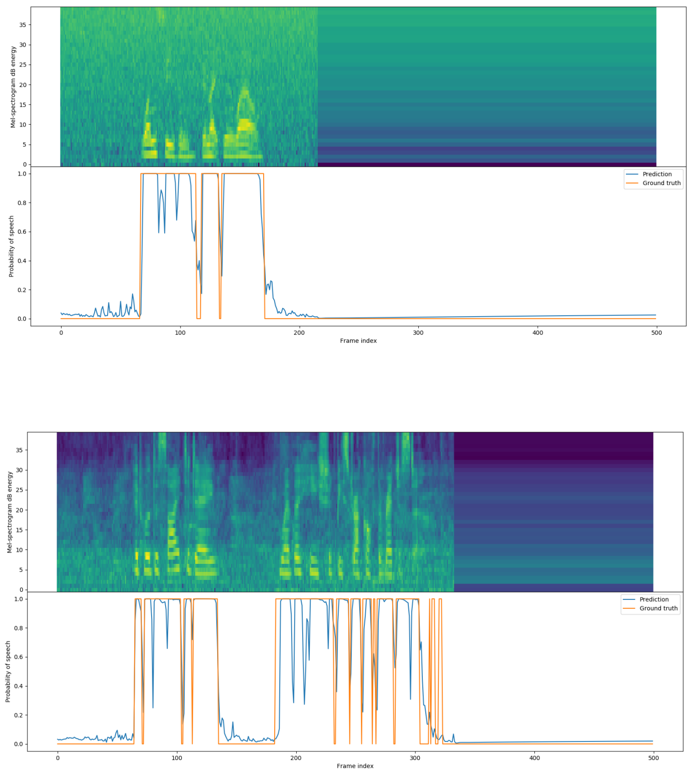

Figure 3: Example speech predictions with noisy inputs using trained GRU network

We also observe through inspecting examples that the ground truth labels are noisy and can be set to positive during silence. In spite of the noisy labels, the network appears to learn robust features that help it accurately detect silences.

We also subjectively evaluate the performance of the network using audio from a laptop microphone. When testing on such out-of-distribution inputs, we notice that the network produces a lot of false positives. This indicates a domain-mismatch problem that will need addressing in the future.

7 Conclusion

We present a neural network based voice VAD that outperforms the popular WebRTC VAD in noisy conditions. We optimized for the F1 score while constraining our network to be sufficiently small to run in a low-resource setting like a microcontroller. We improved on the baseline performance in non-noisy conditions on F1 score from 81.9% to 95.8% and similarly for noisy conditions. The final quantized network footprint was 6.4KB. Our next steps are to re-train the network to reduce domain-mismatch so it can be deployed in different acoustical environments. We would also like to quantize the model to 8-bits to further reduce the memory footprint.

References

[1] Opus 1.3 released. `https://people.xiph.org/~jm/opus/opus-1.3/`. (Accessed on 01/19/2020).

[2] webrtc/common_audio/vad - external/webrtc - git at google. `https://chromium.googlesource.com/external/webrtc/+/branch-heads/43/webrtc/common_audio/vad/`. (Accessed on 01/19/2020).

[3] Jong Hwan Ko, Josh Fromm, Matthai Philipose, Ivan Tashev, and Shuayb Zarar. Limiting numerical precision of neural networks to achieve real-time voice activity detection. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2236–2240. IEEE, 2018.

[4] Thomas Drugman, Yannis Stylianou, Yusuke Kida, and Masami Akamine. Voice activity detection: Merging source and filter-based information. *IEEE Signal Processing Letters*, 23(2):252–256, Feb 2016.

[5] Cstr vctk corpus. `https://homepages.inf.ed.ac.uk/jyamagis/page3/page58/page58.html`. (Accessed on 01/19/2020).

[6] Signal processing information base (spib). `http://spib.linse.ufsc.br/noise.html`. (Accessed on 01/19/2020).

[7] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014.

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

[9] Hyperparameter tuning in cloud machine learning engine using bayesian optimization | google cloud blog. `https://cloud.google.com/blog/products/gcp/hyperparameter-tuning-cloud-machine-learning-engine-using-bayesian-optimization`. (Accessed on 03/16/2020).

[10] Tensorflow lite. `https://www.tensorflow.org/lite`. (Accessed on 03/17/2020).