report

September 4, 2021

1 VAD

author: steeve LAQUITAINE

date: 04/08/2021

1.1 Abstract

Voice activity detection is critical to reduce the computational cost of continuously monitoring large volume of speech data necessary to swiftly detect command utterances such as wakewords. My objective was to code a Voice Activity Detector (VAD) with reasonable performances (Low false rejection rate) based on a neural network within a week and with low computing resources. The model was trained and tested on labelled data from LibriSpeech prepared by SONOS. I used TensorFlow to develop the model and tested several common VAD modifications such as smoothing, minimum speech time, hangover scheme [TODO]. I relied on a set of diverse development and visualization tools to make sense of the data, clean the data and inspect the inner workings of the model and the pipelines. Future work will integrate additional existing speech and noise datasets (some can be found on http://openslr.org). I strived to go beyond just solving the challenge itself.

1.2 Method

1.2.1 Performance metrics

I used False rejection rate (FRR) as the key measure of model performance. FRR also called False negative rate or miss rate equates to 1 - Recall and is calculated as:

$$FRR = \frac{FN}{FN + TP}$$

where "FN", for false negatives are incorrect rejection of speech and TP, for True positives are correct detections of speech.

My reasoning was that:

- the main goal is not to miss speech periods to maximize users satisfaction:
- it is widely used in most paper in the field

The disadvantage of greedily maximizing that metrics is that it could train a model that detects everything as speech, generating lots of false positive, a poor user experience. It is also very computationally expensive.

1.2.2 Dataset

The dataset contains 1914 files separated in 957 .wav audio files associated with a label file (n=957) with the same name. Label files contained keys-values pairs for each speech interval formatted as start_time: XX secs and end_time: YY secs in secs.

1.2.3 Model implementation

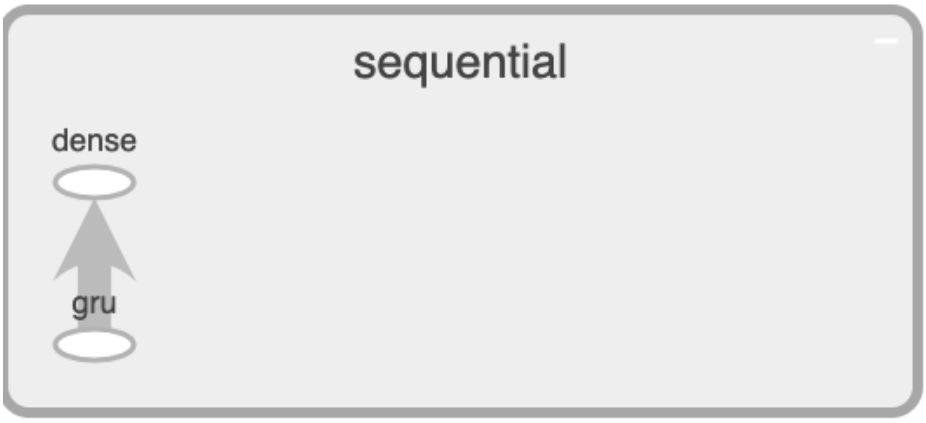
```
/Users/steeve_laquitaine/Desktop/vad/src/vad/pipelines/data_eng/nodes.py:20:
YAMLLoadWarning: calling yaml.load() without Loader=… is deprecated, as the
default Loader is unsafe. Please read https://msg.pyyaml.org/load for full
details.
  config = yaml.load(conf)
```

Neural network architecture: Due to poor computational power, I develop on a laptop, I chose to test minimal network architectures containing up to two hidden layers and a dense binary classification layer. I show the single GRU layer's network's conceptual graph generated by the tensorboard api below.

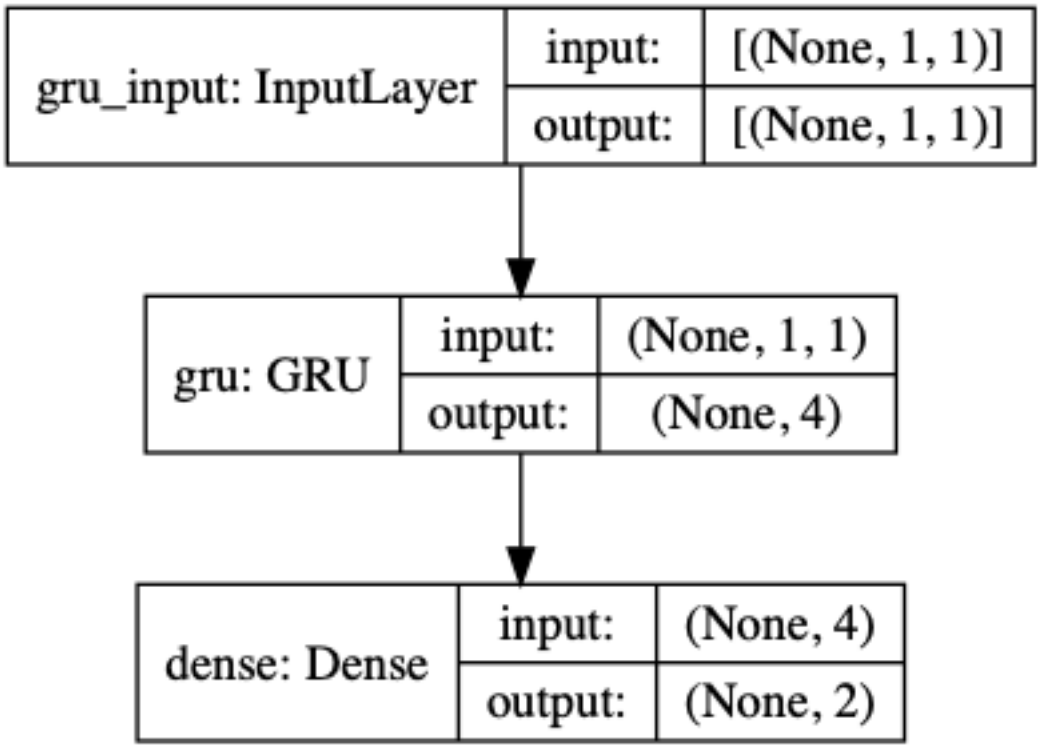
After cloning the project's repository, the graph can be loaded in your web browser by running the command below in a linux (or MacOS unix) terminal:

```
cd ../vad
tensorboard --logdir tbruns
```

where ../vad is the clone repository directory.



I show below a more detailed description of the network layers generated with `tf.keras.utils.plot_model()` api.



I show below the model's number of trainable and non-trainable parameters (`model.summary()`):

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru (GRU)                    (1, 4)                    84
_________________________________________________________________
dense (Dense)                (1, 2)                    10
=================================================================
Total params: 94
Trainable params: 94
Non-trainable params: 0
_________________________________________________________________
```

1.3 Experiments

1.3.1 Approach

I chose to implement the simplest modeling approach in order to ensure quick iterations and debugging. My approach was based on sequential hypothesis testing.

Does a minimal model perform better than chance ?

I first extracted a small sample of the real data to get fast training at the expense of performance. I trained a simple 1-layer GRU model which takes as input a single batch with one look-back timestep and projects to a fully connected binary classification layer with a softmax activation function. I first trained the model with one epoch. I used `Adam`, an adaptive optimizer that is well known to handle the complex training dynamics of recurrent networks better than simple gradient descent optimization. Training the model on a 2.8 secs audio from one speaker (20% of the full audio chunk) took 50 secs.

Is more layers better than less ?

I expected that a 1-layer network would perform better than a 2-layer network a deeper network enable the model to capture more non-linearity in the data, but both models produced had similar performance and a 1-layer GRU did not train faster than a 2-layer net.

Lower resolution better speed?:

I tested two data encoding resolutions: float16 and float32 bits. Both resolution took the same time to train.

Does learning from more timesteps reduce performance ?:

I tested 1, 2, 10 and 320 timesteps which correspond to 0.06 ms, 0.12 ms, 1.2 ms and 20 ms with the current sampling rate of 16 Khz. - Moving from 1 (50 sec training) to 2 timesteps (2 min trainin) produced a 2.5 fold improvement in FRR from 1 timestep (0.34 on average to 0.14) over subjects - FRR improved by 20% from 2 to 10 timesteps which took 5 min to train. - Training on 1 timesteps took about 50 secs, on 10 timesteps 90 secs. - 320 timesteps took too long to train within the given time constrains and was stopped.

Performance vs. train-test split ?:

Performance drastically increased from 0 all over to 0.8-ish train from a 0.5 split to a 0.7. Performance was same for 1 or 2 GRU.

How long to convergence ?:

The loss curve suggests that 7 epoches are enough for convergence (shoulder of the curve).

The more data the better ?:

I trained the model on 20% vs 100% of train set for 7 epochs * same results across nb GRU * metrics: * 100% has:
* 15% better recall (86%) * 10% lower precision (85%) * 4% better f1 (86%)

Does training on 2 secs audio perform well ?:

I trained the model on one 2.8 secs from one speaker's audio (19-198-0003.wav) and tested how well the model generalizes to his other audio chunks and other speakers' audios:

- I trained on speaker 0 (19-198) and predicted on a sample of other test audio chunks from the same speaker. The test performance were about 0.85 f1-score.

I tested how well the model prediction generalize to other speakers audio:

- I trained on speaker 0 (19-198) and predicted on two other persons' audio chunks (103-1240-0001, 118-47824-0000). As a result:

  - 103-1240-0001:
    * I observed an increase in false rejection rate (50% only compared to ~30%)
    * candidate explanation: other person could speak with lower average amplitudeshorter speech periods which would lead the model to miss speeches
  - 118-47824-0000:
    * False rejection rate was the same

- Synthetic vs. person
  - train on person 1 - predict on synthetic (1263-141777-0000)
    * result: FRR was good

- Loud vs. low voice
  - train on person 1 - predict on human (1447-130551-0019)
    * kept good f1 score 0.7 with good recall and reasonable precision (0.6)

- normal vs. high pace
  - I train on speaker 1 - predict on human (1578-6379-0017)
  - kept good f1 score with good recall and reasonable precision

Which activation sigmoid (default) vs. biased Softmax ?

False rejection rate was slightly lower (from 13% to 9% on other subjects test audios) for biased softmax at the dense layer than for the default sigmoid activation function. So I retained Bias softmax. The speed of convergence remained unchanged.

Is Inference fast enough ?

Inference took 12.6 sec on a 15 sec audio that is 840 ms / sec.

1.3.2 A few sanity checks

I tested the inference pipeline:

1. Audio label mapping was shuffled
   - result: FRR worsened to 40% (possibly not significantly different from chance).
2. I tested a dummy model that always predicts "speech":
   - result: FRR worsened to 0.4 (precision was high which is expected if labels are imbalanced toward speech)
3. I tested a dummy model that always predicts "no speech":
   - result: FRR worsened to 100%, together with precision.

I tested the train pipeline by shuffling the audio label-mapping:

- As expected:
  - FRR was 100% and precision 0%
  - there was virtually no reduction in loss over epochs indicating no learning

1.3.3 Final setup

Training dataset:

- 2.8 secs audio from one speaker: this small chunk took 4 min to train while using a full 15 secs audio file (i.e., 5 times more data) took 20 min for a 10% performance improvement.

Architecture:

- 1-layer GRU: Occam's razor principle favors simplicity given the same performance
- Softmax with initial bias to couterbalance the label bias

Training:

- 1 batch
- 10 timesteps

- at least 7 epochs
- split_frac: 0.7 : had the biggest effect on performance (0 at 0.5 to 0.7 at 0.7)

1.4 Results

1.4.1 Description of the audio data

There are several speakers Listening to a sample of the audio files revealed that a variety of speakers

- Humans
  - men
  - women
- Synthetic
  - men
  - women

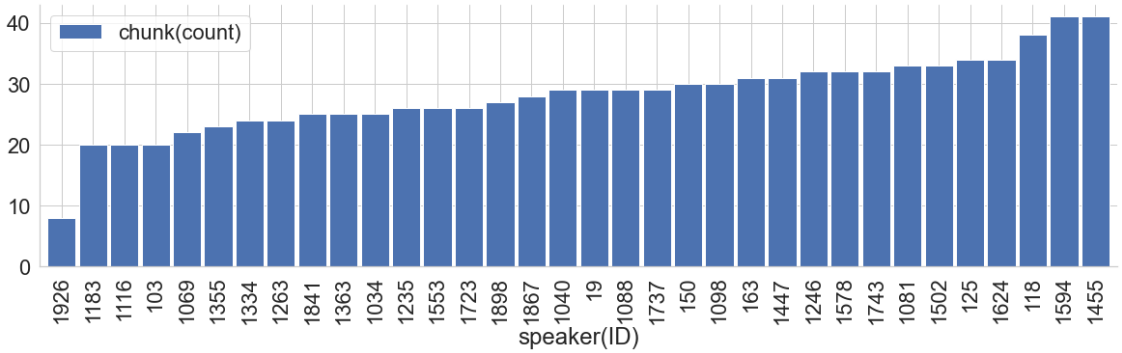
I also characterised speeches by their variety of amplitudes and pace - Normal vs fast pace
- Loud vs, low volume

We show below the best typical example of an audio signal (top panel). and its associated speech labels "1" for speech and "0" for no speech (bottom panel).

All audio signals were 32 bits float single channel time series. We run a few sanity checks:

```
- the 957 label files were correctly mapped with the 957 audio files
```

```
- Number: 34 speakers
- Speakers'ID: ['103' '1034' '1040' '1069' '1081' '1088' '1098' '1116' '118'
'1183'
 '1235' '1246' '125' '1263' '1334' '1355' '1363' '1447' '1455' '150'
 '1502' '1553' '1578' '1594' '1624' '163' '1723' '1737' '1743' '1841'
 '1867' '1898' '19' '1926']
```

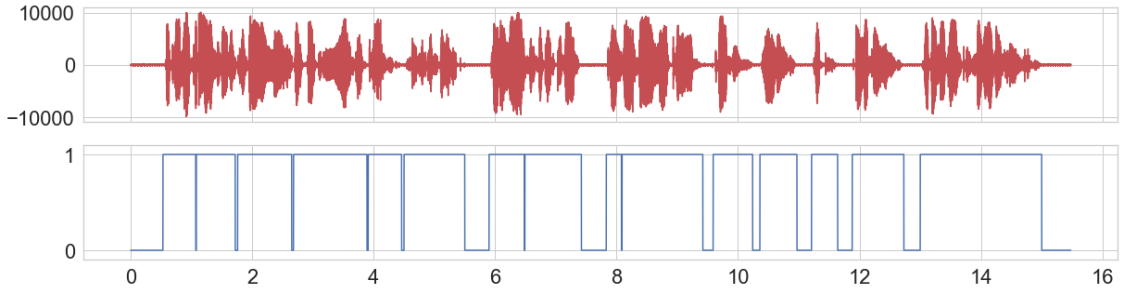


We show below a few interesting example chunks for two speakers. - Audio seem well labelled (see supplementary). - Background noise is low and stationary.

SPEAKER 19

data/01_raw/vad_data/19-198-0003.wav

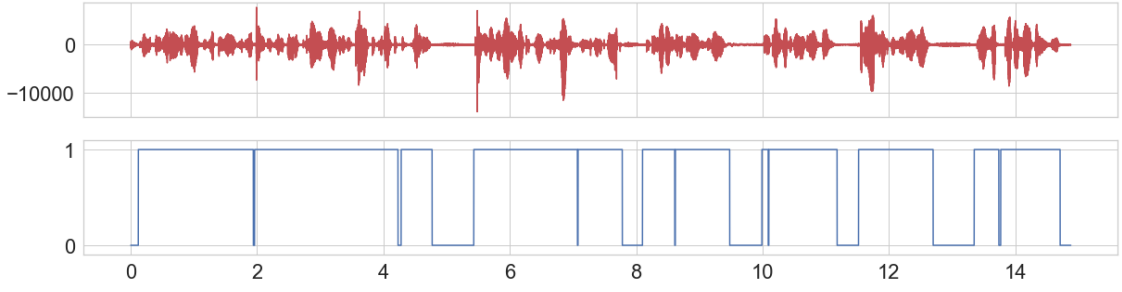
data/01_raw/vad_data/19-198-0003.json



SPEAKER 103

data/01_raw/vad_data/103-1241-0027.wav

data/01_raw/vad_data/103-1241-0027.json



We validated that all audio files were associated with a .json label file.

```
- audio file sample size: 957
- label file sample size: 957
```

The entire sample could be loaded quickly:

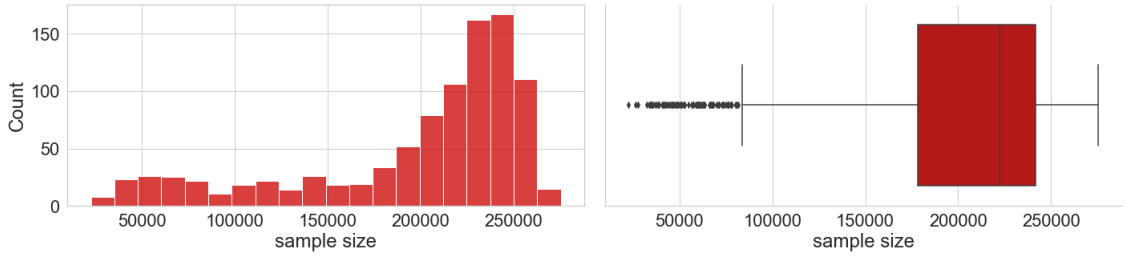
```
- loading duration: 2.74 sec
```

Sample size and sampling rate:

```
Sample rate information:
- 1 sample rate(s)
- rate: 16000 Hz
```

We kept the signal at 16Khz which is enough to cover the frequency range of human speech according to the literature (Human voice b/w `85hz to 8khz` [REF], hearing b/w `20 hz to 20kh`[REF]).

```
Sample size information:
- 711 sample size(s)
- max: 275280 samples ( [17.205] secs)
- min: 22560 samples ( [1.41] secs)
- median: 222080.0 samples ( [13.88] secs)
```



Signal amplitudes: the true decibel amplitude of the audio will depend on each speaker's microphone characteristics, the speaker's distance to its microphone, the speaker's volume configuration. Having no acces to these information we did not derive the true decibel amplitude (dB) from the raw audio signal amplitude or compared absolute amplitudes between speakers. Rather we compared the signals' signal-to-noise ratio (SNR).

1.4.2 Speech signals are nearly pure

I made the naive assumption that audio can be linearly decomposed as the sum of independent speech and noise amplitude components. This assumption would not hold in case of reverberation.

$$audio = speech + noise$$

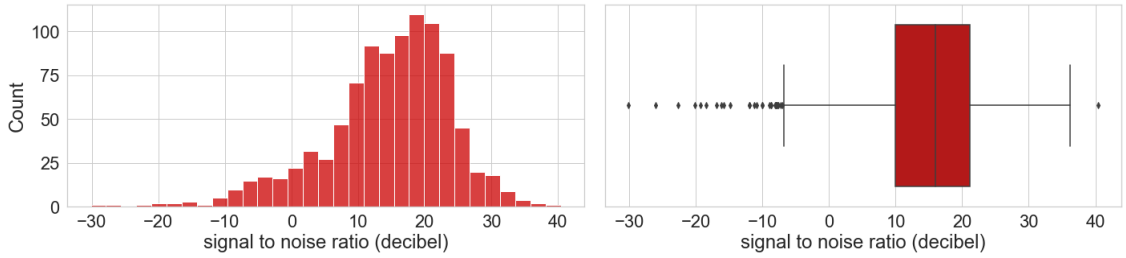
Where i respectively categorize noise as the portion of the audio that is not labelled as speech. Thus:

$$speech = audio - noise$$

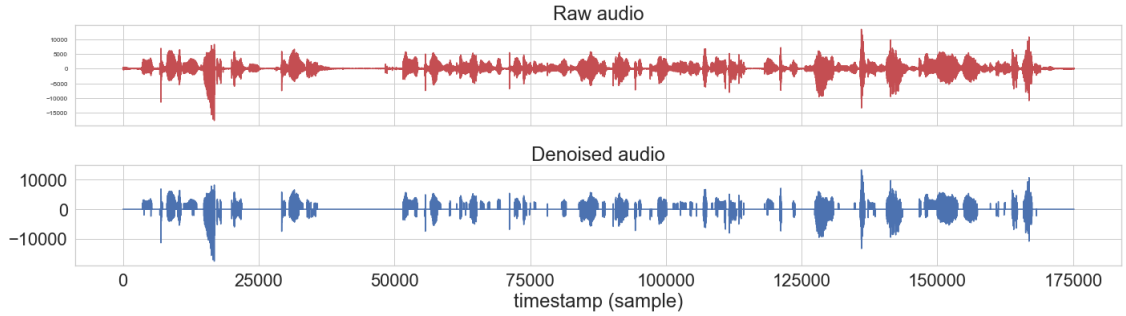
The signal to noise ratio in decibel is given by:

$$SNR = 20.log_{10}(\frac{speech\_rms}{noise\_rms})$$

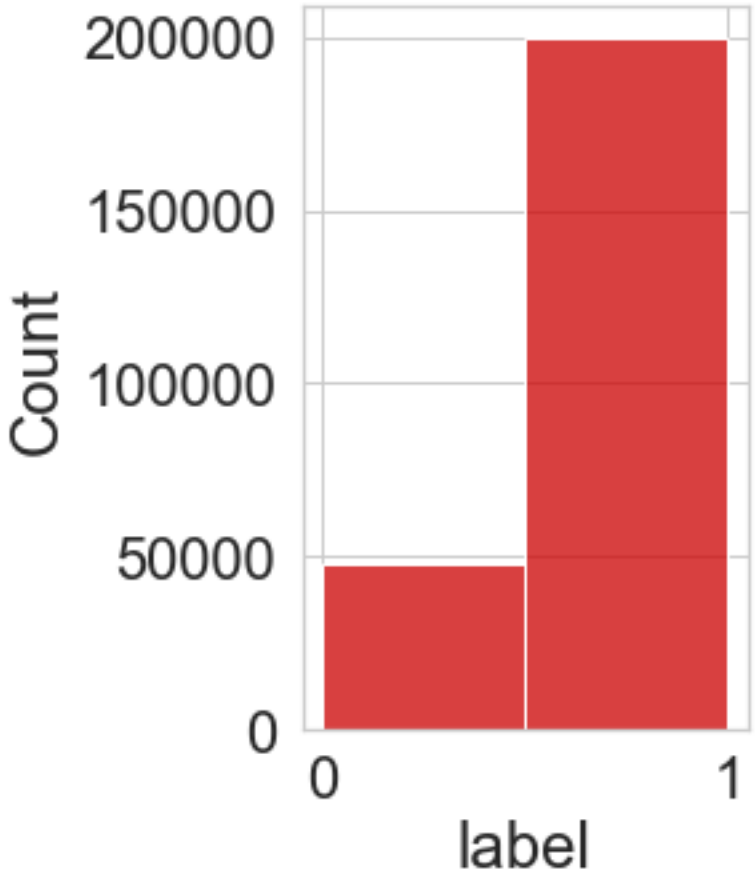
where speech_rms and noise_rms are the root mean square of the speech_signal and of the noise



I show below what noise dampening (blue in bottom panel) does to the raw signal below (red, in top panel) for an example audio file.



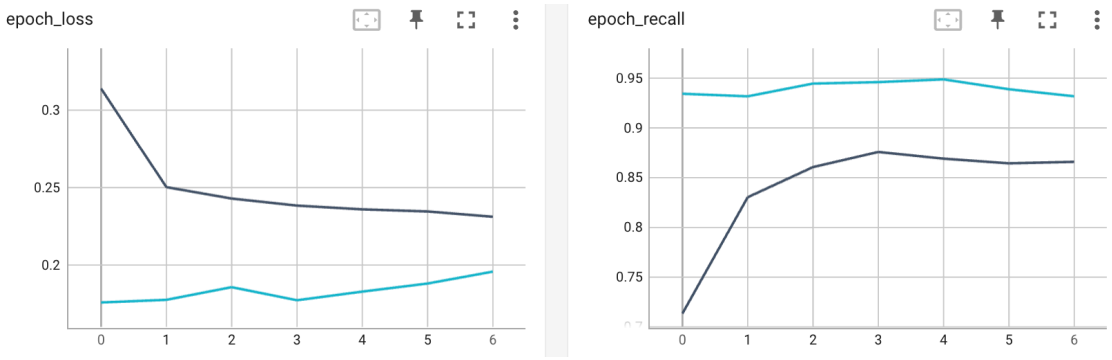
1.4.3 Speech and no-speech classes are imbalanced in the training dataset



1.4.4 The 1-layer GRU net converges within 7 epochs

I monitored training quality in Tensorboard api. I display below the loss curve calculated for 7 training epochs (x-axes ranging from 0 to 6) on training dataset (dark curve) and validation dataset (blue curve). The model was trained with 10 timesteps look-back (see logged runs in tensorboard for vad-1-gru-20210901-131436/train/train, vad-1-gru-20210901-131436/train/validation in tensorboard web api). Categorical cross-entropy on training data was not flat as expected from a model that doesn't learn but goes down until convergence near the 7th epoch.

I also monitored recall to evaluate improvement in recall (decrease in FRR) over training. Recall increased (thus FRR decreased) by 10% within 7 epochs and took about 4 epochs to converge on the training dataset. Recall on validation was already high since the first epoch and remained stable over training (see logged runs in tensorboard for vad-1-gru-20210901-131436/train/train, vad-1-gru-20210901-131436/train/validation).



1.4.5 Weights and biases follow well-behaved multimodal distributions

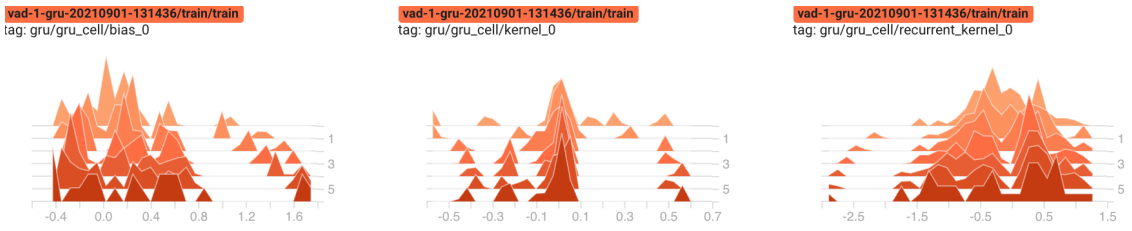
Both the Gru and dense layer show distribution that did not get stuck at 0 values and did not display large outlier values. This suggests that our GRU units successfully circumvented the vanishing gradient problem that is typical of more traditional RNN, thanks to its "Gates" which avoid the long-term information from "vanishing" away.

Gru layer

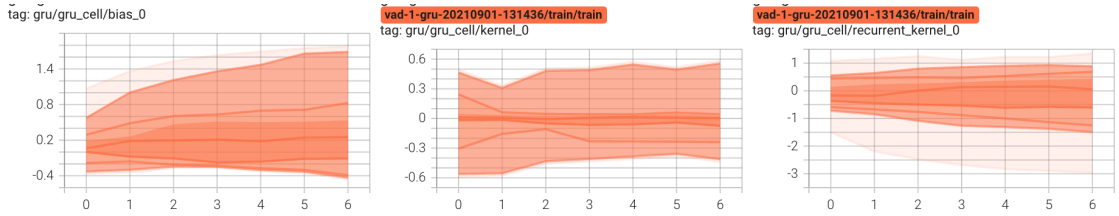
The GRU l layer has 3 types of trainable parameters: - weights ("kernel_0") - recurrent weights ("kernel_0") - biases ("kernel_0")

I will refer to iterations as epochs as for a single batch, iterations equal the number of epochs.

- Weight and bias values
  - The weight distribution was very sparse with the majority of weights being null and some values being spread at negative and positive values.
  - GRU layer's biases and recurrent weights slightly changed over epochs (y-axis), spreading across a larger range of values.

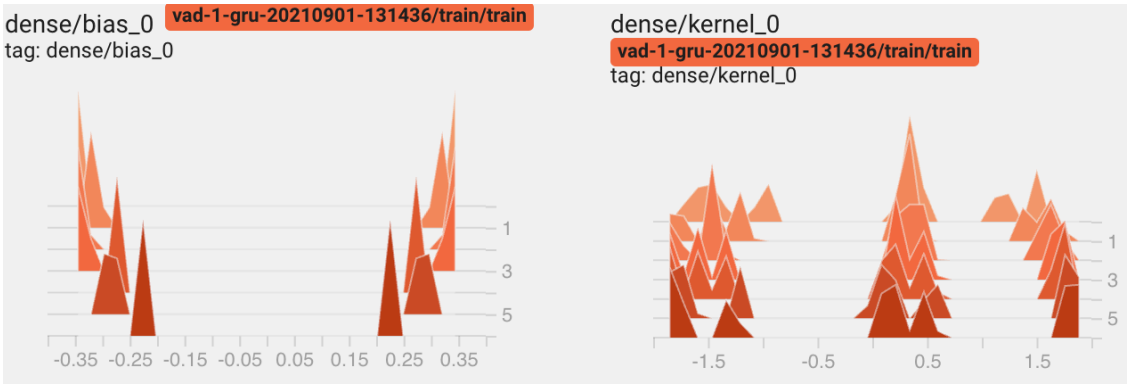


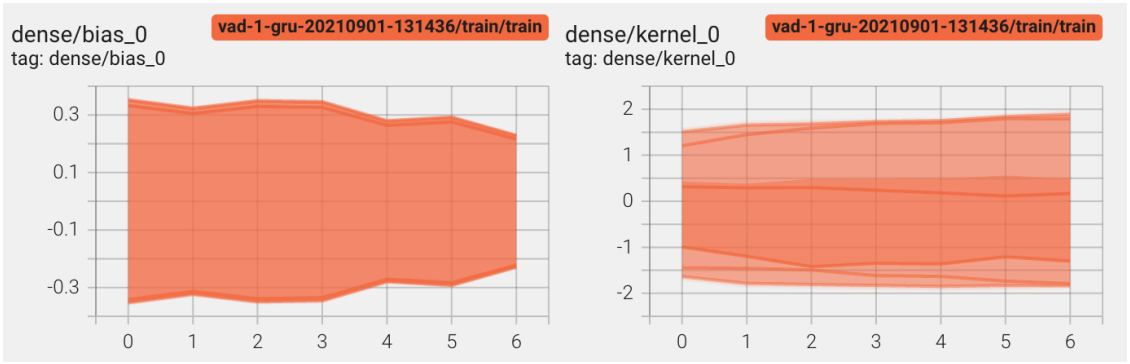
These changes in the bias and weights distributions are more apparent on the plot of the weight values against epochs (different shading columns represent the distributions' 90th, 60th percentiles ...) (x-axis).



Dense layer

- Dense layer's weights and biases did not get not stuck at 0 and did not display large outlier values.
  - Biases followed a bi-modal distributions with some weights at -1, others peaking at -0.3 and other at +0.3. There are no outlier biases.
  - Weights followed a tri-modal distributions with some weights at -1, a small majority peaking at 0 and some peaking at +1.5. There were no outlier bias values.





1.4.6 A 1-layer GRU model predicts well

I first trained the model on a small chunk (2.8 secs) of one speaker data and I tested it on one full audio for each of the 34 speakers' (about 3.5 min, for 15 sec inference per audio file). I chose the

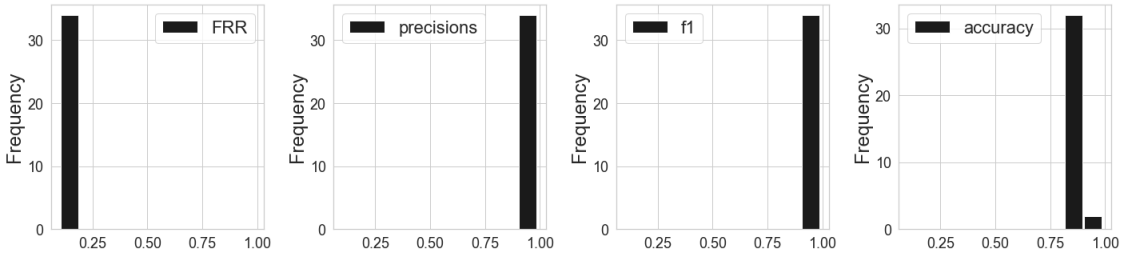
first audio for simplicity. Further analyses must assess whether the first audio is representative of the other audios for each speaker. I chose a sample and not the entire dataset because inference would take more than 3 hours on the 957 files. I show below the distribution of the best model's False rejection rates over the sample test dataset.

I display below the sequential steps performed by my inference pipeline and the False rejection rates for all speaker's first audio recording.

I show below the FRR statistics calculated over the FRR calculated from the first audio of the test subjects with various other usual performance metrics:

	FRR	precision	f1	accuracy
count	34.000000	34.000000	34.000000	34.000000
mean	0.113511	0.977082	0.929584	0.891888
std	0.003565	0.003150	0.003385	0.005164
min	0.099462	0.976307	0.928750	0.890616
25%	0.114389	0.976307	0.928750	0.890616
50%	0.114389	0.976307	0.928750	0.890616
75%	0.114389	0.976307	0.928750	0.890616
max	0.114389	0.989496	0.942923	0.912238

The FRR histogram (shown below) concentrates near 9% with no apparent outlier. This indicates that the model generalize well to all speakers: human men, women and synthetic speakers.



1.4.7 Adding VAD modification

Enforcing a minimum speech time The minimum speech time observed in the dataset was very short. A realistic minimum speech duration would be 1 secs, to say "hi" for example.

`The minimum speech time derived from the data was: 10.0 ms`

I modified the model to predict a sequence of labels. I implemented a Gru layer that outputs a (.., T timesteps, ..) fed to replaced a classification layer. I replaced the Dense classification layer used for the former model by a TimeDistributed Dense layer with a softmax activation function to predict a sequence of (T timesteps, 2 classes) label probabilities. I added a penalty to the loss function such that each occurrence of minimum speech time violation with the choosen a "timesteps" period increases the loss by +1.

I display my custom loss function below (see `src.vad.pipelines.train.nodes.MinSpeechLoss`).

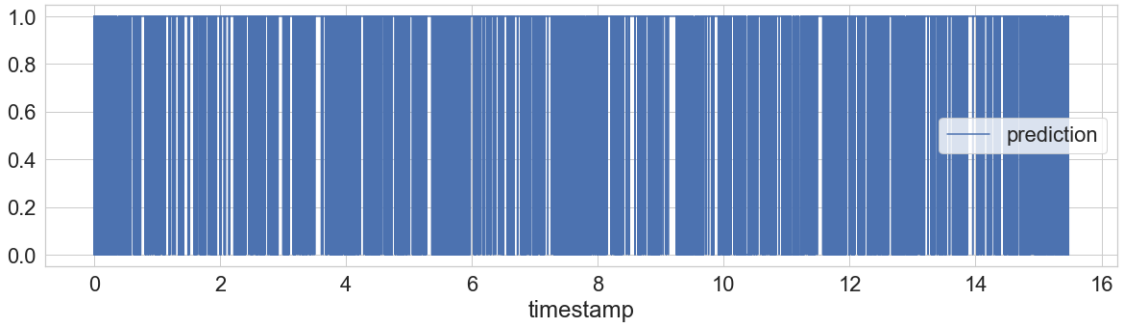
```python
class MinSpeechLoss(tf.keras.losses.Loss):
    """Loss penalized to enforce a minimum speech time"""

    def __init__(self):
        """Instantiate loss constrained to enforce a minimum speech time"""
        super().__init__()

    def call(self, y_true, y_pred):
        # replicate y_pred over timestep axis and calculate loss
        cce = CategoricalCrossentropy()
        cce_loss = cce(y_true, y_pred)
        cce_loss = tf.convert_to_tensor(cce_loss)

        # penalize minimum speech violations
        minspeech_loss = get_penalty(y_pred)
        return cce_loss + minspeech_loss

    @classmethod
    def from_config(cls, config):
        return cls(**config)
```

Applying label smoothing I used categorical cross entropy loss and applied label smoothing (0.5). Label values are smoothed, relaxing the confidence on their values. For example, i applied a `label_smoothing=0.5` which corresponds to 0.5 for the non-speech label and 1 for the speech label.

```
model.compile(
    loss=CategoricalCrossentropy(label_smoothing=0.5), ...
)
```

Before smoothing Train the model and predict in the terminal (linux):

```
kedro run --pipeline train --env train
kedro run --pipeline predict --env predict
```



The performance metrics of the model on this example subject are:

```
                          0
accuracy           0.795272
precision          0.911992
recall             0.849165
f1                 0.879458
false_rejection_rate  0.150835
```

After smoothing To train the model and predict run in terminal:

```
kedro run --pipeline train --env train_smooth
kedro run --pipeline predict --env predict_smooth
```

```
                          0
accuracy           0.795272
precision          0.911992
recall             0.849165
f1                 0.879458
false_rejection_rate  0.150835
```

1.5 Conclusion & Discussions

I have developped a 1-layer GRU neural network that detects speech activity with about 9% false rejection rate on test dataset different speakers' voices including male and female voices and synthetic voices, with a resource-constrained device (a laptop).

I have implemented two voice activity modifications: - mimimun speech time - label smoothing

1.6 Perspectives

To improve the model, i could apply a few preprocessing steps:

- data minmax rescaling between 0 and 1
- z-scoring

to speed up learning.

To improve the model the next steps would be to try a more thorough hyperparameter search:

- different learning rates
- more model architectures

The model should also be tested for online speech detection with streaming audio data from a microphone: - Add a streaming inference pipeline that makes online predictions

Deployment a constrained resource device such as an iphone or BlackBerry Pi could be tested and would require, for example:

- model quantization
- serializing the model to a lighter file format such as tensorflow lite

1.7 References

1.8 Supplementary results

1.9 Speakers' first audio

1.10 Supplementary methods

1.10.1 Experimental run tracking

Each run parameters was tracked with mlflow api.

1.10.2 Training pipeline

we display below the directed acyclic graph of our training pipeline, plotted with the `kedro-viz` api.

1.10.3 Inference pipeline

1.10.4 Evaluation pipeline