Francisco Javier Sáez Maldonado
franciscojavier.saez@estudiante.uam.es

Speaker Recognition
Deep Learning for Audio Signals

2022-04-25

Contents

Introduction

Speaker recognition is the task of identifying a person from his/her voice. In this lab report, we will examine a speaker recognition system that will use *x-vectors* embeddings to classify the speakers. The system has been developed to solve the VoxCeleb challenge.

The system *extracts* the fixed-size embeddings trying to include in these representations as much discriminative information as possible for the speaker recognition task. A few architectures of neural networks can be used with this purpose, as well as different loss functions.

The code that I used to test the model, which was provided by Alicia Lozano, can be found in this link.

Examining the code

In this section, we will examine the provided code in order to find the most relevant parts of it. Let us begin by examining the `trainSpeakerNet.py` file, since it is the main script. The code is properly commented so identifying the different parts is quite simple.

Questions.

- *Which line of the code trains the model?*

  We found `line 169`:

  ```
  loss, traineer = s.train_network(loader=trainLoader);
  ```

  which trains the code. The variable `s` was previously initialized as `s = SpeakerNet(**vars(args));`, creating an instance of the SpeakerNet defined model class. This class has the called `train_network` function, which performs (vaguely explained) a classic forward pass with loss computation and returns the average loss and the `train eer`.

- *Which command loads a pre-trained model?*

  Having a look from lines $112-121$, we find that in lines 116 and 120, the function `s.loadParameters(args)` is called. This function loads the pretrained model from a specific file. As we can see in the code, a distinction is made in our case, being able to load the model from the `save_path` or from the argument `args.initial_model`.

- *Which line evaluates the performance of the neural network?*

  A few lines below the training line, in line 176 we find:

```
sc, lab, _ = s.evaluateFromList(args.test_list, print_interval=100,
                                test_path=args.test_path, eval_frames=args.eval_frames)
```

which evaluates the model in the `test_list`. In this function, all the features for each of the test file audios (*.wav*) files are extracted and then the scores are computed using `pairwise_distance`.

- *Which variable contains the scores of the trials list? In which file are those scores saved?*

  Firstly, we find in line 176 that the scores of the trials are stored in the variable `sc`. Then, in line 177 we observe that the scores are thresholded and stored in the variable `result`. Then, they are saved in the `scorefile` file. This is a variable that, in line 150 opened a file called `result_save_path/scores.txt`, where `result_save_path` came as an argument from the execution of the script.

- *Which variable controls the number of epochs(iterations) that the model is trained?*

  We find in line 198 and 201 that the variable `it` is modified and checks the `max_epoch` indicated as an argument. Then, is the variable `max_epoch` the one that controls the number of epochs that the model is trained.

- *Which parameter determines the loss function?*

  We find that in line 29 the argument `--trainfunc` with description *'Loss function'* is declared. Then, it is passed to the model class in the creation of the instance of the model (passing all the arguments at the same time).

- *What is the purpose of the parameter `test_interval`?*

  Having a look at the description of the parameters, in `line 27` we read that this parameter controls how many train epochs we make before every evaluation. For instance, if $test\_interval = 10$, then every 10 training epoch we perform an evaluation on the `test_list` (`line 176`) and save the results.

- *Which argument has to be modified in order to change the architecture of the model?*

  The architecture of the model is defined in the argument `--model`. As we can see, in the folder `models`, there are a few implementations of `ResNet` and `VGG` that could be used to change the architecture of the model.

- *What is the purpose of the parameter `nClasses`? How would you obtain this value from the dataset? Write the command that you would use.*

  The parameter `nClasses` controls the number of speakers that are used in the softmax layer. Let us explain how we would obtain this from the dataset. As we are explained in the provided jupyter notebook, the training data is stored in the folder `data/voxceleb2`. Inside this folder, we find a large number of subfolders, each one containing the audios from a different speaker, which **represents a class** in our problem. So, to obtain the parameter `nClasses`, we would have to count the number of folders stored in the directory `data/voxceleb2`. We can do this in linux by using a `pipe`, listing the folders in the directory and then counting the number of lines (using the linux command `wc -l`). The execution provides the following result:

  ```
  !ls data/voxceleb2/ | wc -l
  ```

  ```
  4953
  ```

  It has to be remarked that the validation set does not have the same number of unique classes. When we execute this same command in the folder `data/voxceleb1`, we obtain that there are only 40 classes.

Evaluating pretrained models

We are given two pretrained models using the *VoxCeleb2* dataset (the dev partition). In this section, we evaluate the models and compare the results. To evaluate the models, we have to execute the previous

script, using the argument `--eval` as well as the dataset path, the model type and some extra parameters. We show the execution output (removing the `warning` lines).

First Model:

```
!python ./trainSpeakerNet.py --eval --model ResNetSE34L --log_input True \
          --trainfunc angleproto --save_path exps/test_lite --eval_frames 400 \
          --test_list lists/veri_test.txt \
          --initial_model pretrained_models/baseline_lite_ap.model


Embedding size is 512, encoder SAP.
Initialised AngleProto
Initialised Adam optimizer
Initialised step LR scheduler
Model has 1437080 parameters
Model pretrained_models/baseline_lite_ap.model loaded!
Reading 4700 of 4715: 28.75 Hz, embedding size 512
Computing 37700 of 37720: 1776.69 Hz

EER 2.1792
```

Second Model:

```
!python ./trainSpeakerNet.py --eval --model ResNetSE34V2 --log_input True \
  --encoder_type ASP --n_mels 64 --trainfunc softmaxproto \
  --save_path exps/test_v2 --eval_frames 400 --test_list lists/veri_test.txt \
  --initial_model pretrained_models/baseline_v2_ap.model

Embedding size is 512, encoder ASP.
Initialised Softmax Loss
Initialised AngleProto
Initialised SoftmaxPrototypical Loss
Initialised Adam optimizer
Initialised step LR scheduler
Model has 11103416 parameters
Model pretrained_models/baseline_v2_ap.model loaded!
Reading 4700 of 4715: 4.19 Hz, embedding size 512
Computing 37700 of 37720: 1766.00 Hz

EER 1.1771
```

Questions.

- *How does each of the previous model perform?*

  We obtain the following results:

  - Model 1 EER: 2.1792.
  - Model 2 EER: 1.1771.

  Further explanation on these results will be provided in the following questions.

- *Which one is the best and why?*

  As we will explain in the following question, the lower the EER is, the better the model is. This implies that, in our case, the second model is better than the first one.

- *What is the EER and how can we interpret it?*

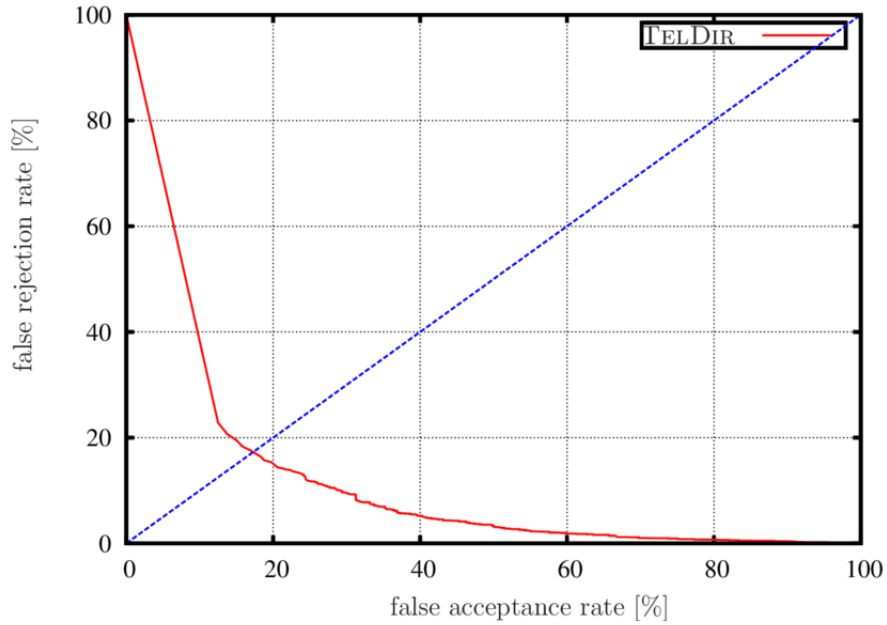


Figure 1: Example of DET curve.

EER stands for **Equal Error Rate**. To understand this concept, we assume that the concepts of *TP, TN, FP, and FN* are known. Using these concepts and considering N as the total number of examples, we can define the **False Rejection Rate (FFR)** and **False Aceptation Rate (FAR)** as the ratios:

$$FFR = \frac{FN}{N}, \quad FAR = \frac{FP}{N}.$$

For each of these two metrics, we have to choose a *threshold* that we would like our model to fulfill, using the number of *allowed False Positives and False Negatives*. The **Dectection Error Tradeoff (DET)** curve represents this tradeoff trying to decrease both FAR and FFR. We can see an example of DET curve in Figure 1.

With all these concept, we can define the EER as the point of the DET curve where $FAR = FRR$, that is, the point where our model rejects the same amount of positive examples than the number of negative samples that it accepts.

At this point, we get to understand why a lower EER is better, since if the EER is lower it means that both our FAR and FFR (which are *negative* quantities) are low.

- *What are the differences between the models?*

Training a model

In this section, we are given a few specifications for a model and we are asked to train a model using these specifications. In particular, the specifications are:

- The **encoder** must be a `ResNetSE34L` with `SAP` (which does not increase the out dimension of the linear layer of the model).
- Loss function **amsoftmax**
- `Scale = 30`
- `margin = 0.3`
- Number of speaker equal to the number of classes (we found before that the number is 4953).
- Fix a number of epochs. In our case, we tested a few different numbers, fixing the final number to 100 epoch (leading to a training that lasts more than one hour).

To train a model matching the required specifications, we must execute the following command:

```
!python ./trainSpeakerNet.py \
  --model ResNetSE34L \
  --encoder_type SAP \
  --trainfunc amsoftmax \
  --save_path mytrained_models/model1 \
  --batch_size 200 \
  --scale 30 \
  --margin 0.3 \
  --nClasses 4953 \
  --train_list lists/train_list.txt \
  --test_list lists/veri_test.txt \
  --max_epoch 10
```

We decided to plot how the **EER** evolves with the increase of the epochs. We can see the evolution in Figure 2.

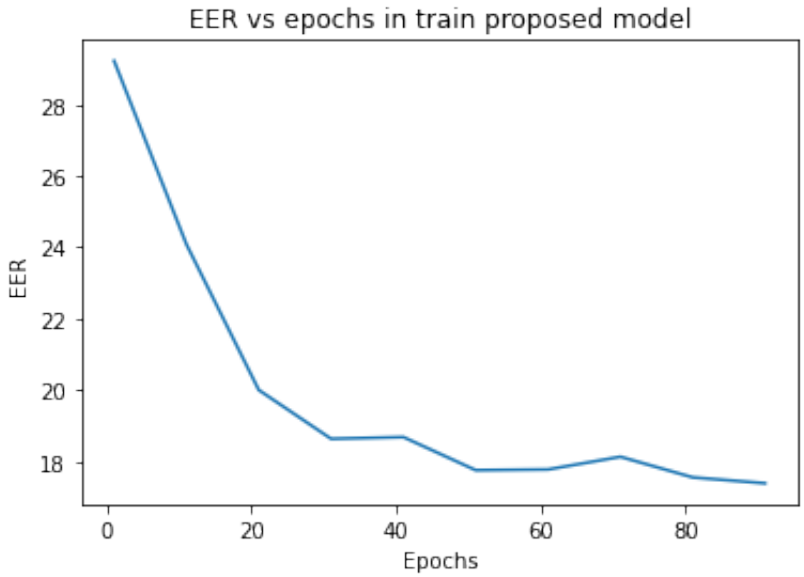
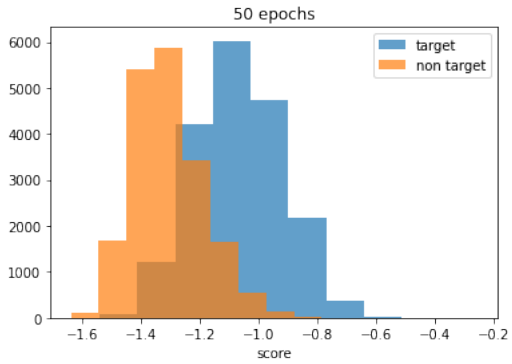


Figure 2: EER of the proposed model in the different epochs.

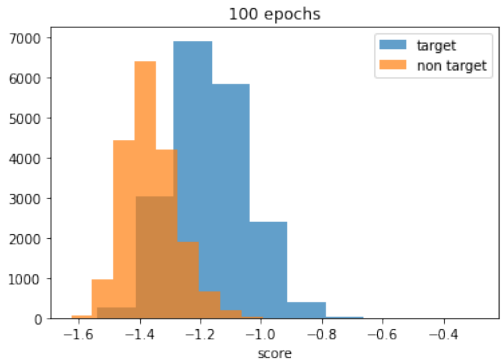
As we can observe, the EER first has a fast decreasement, which is stopped after approximately 30 epochs. After training the model, we have to evaluate the obtained model so that we can plot the histograms of the test set. In our case, the paths to the model folders are a little bit confusing due to saving-loading the data into *Google Drive*. The command to evaluate is the following:

```
!python ./trainSpeakerNet.py --eval --model ResNetSE34L  \
  --encoder_type SAP --trainfunc amsoftmax \
  --save_path exps/mymodel_1 --eval_frames 400 --test_list lists/veri_test.txt \
  --initial_model mytrained_models/mytrained_models/model1/model/model000000100.model
```

Using the evaluation in both 50 and 100 epochs, we can plot both histograms to compare them. We see this in Figure 3.



(a) 50 epochs



(b) 100 epochs

Figure 3: Histograms of target and nontarget examples for different number of epochs.

Questions.-

- *How many epochs have you used?*
  As we have mentioned before, we used 100 epochs.
- *What is the score of the model=*
  The final score of our model in epoch 100 is $EER = 17.605$. Also, in epoch 50 the score was $EER_{50} = 18.8229$, which is not a really big difference considering that is half the number of epochs.
- *How is this model compared with the previous pretrained models? Why do you think this is happening?*
  This model is clearly worse than the previous ones. There are a few reasons that we could consider:
  1. The number of epochs of the previous models was much higher
  2. In the previous models, the logarithm is applied to the input. This could be a reason why the results are quite different.
  3. The number of mels used is also different, being the default value in our case (40) and 64 (a higher value) in the previous cases
  4. We are also applying a margin and a scale which we do not know if it has been applied in the previous model.
  5. Lastly, the train function is different, `softmaxproto vs amsoftmax`, which could also make a difference in the results.
- *Include the histograms in this document. Which threshold would you use to classify between target and non-target trials?*
  We can see the histograms in Figure 3. We observe that when the number of epochs increases, the non-target distribution resembles more a gaussian distribution, and the target distribution shifts to the right. The used threshold would be different depending on the number of epochs shown:
  1. For 50 epochs, an appropriate threshold sould be around -1.35.
  2. For 100 epochs, using -1.3 would probably provide good results.

Extra: Training a model of our choice

In this last section, we are asked to train a model of our choice, choosing an **encoder** and a loss function to train a model.
The choice I made was to observe the behaviour of the **triplet loss function** using any encoder architecture. I find triplet loss functions quite interesting since their motivation comes from a very natural and intuitive concept. Let us explain a little bit more about triplet losses.

4.1 Intuitions on triplet losses

x-vectors are representations of the audio signals we are dealing with. Ideally, we would like to obtain intermediate representations in our encoder that, given an input data x, preserve the distance between similar data points close and also makes de distance between different datapoints far on the embedding space [1].
Let us set the notation that we will use first. We will consider sets of triplets (x, x^+, x^-) where:

- The element x is an anchor point,
- The element x^+ is a positive instance,
- The element x^- is a negative instance.

The main idea is to learn a representation of x, say $g(x)$, such that the distance of the representation of the input is closer in distance to the representation of the positive sample x^+ than the representation of the negative sample x^-. Using the norm, we can formally express that as follows:

$$\|g(x) - g(x^+)\|_2 \leq \|g(x) - g(x^-)\|_2,$$

for each triplet in the set.

The loss function that we will use will use this inequality, adding a last term, which we will motivate now: Support-vector machines (SVMs) are supervised learning models used for classification or regression problems. They are one of the most robust prediction methods. They search for a hyperplane h in high or infinite dimensional space that separates the data as much as possible, making use of *support vectors*, the datapoints that are closest to the hyperplane. If the data is linearly separable, we can select two hyperplanes h_1, h_2 that are parallel to h and making the distance from them to h as large as possible. That region is called the *margin*.

Coming back to our triplets problem, we also want to introduce a margin between the distances of the elements of the triplets, in order to separate positive examples from negative examples as much as possible. This way, we introduce a *margin* term α, rewriting our last equation as follows:

$$\|g(x) - g(x^+)\|_2 + \alpha < \|g(x) - g(x^-)\|_2.$$

Using this inequality, we can define a loss function for each triplet in the set:

$$\ell^\alpha(x, x^+, x^-) = \|g(x) - g(x^+)\|_2^2 - \|g(x) - g(x^-)\|_2^2 + \alpha. \tag{1}$$

This loss has been defined for a single triplet. This is the loss function that we have defined in the file `loss/triplet.py`.

4.2 Less parameters

The last task is to train a model using less parameters. In our case, if we do not want to modify the models (ResNet, VGG), we can only explore the arguments given in the training script to try to reduce the number of used parameters. We found that using the parameter `--nOut`, we can specify the output dimension of the `linear layer` of the model, so the goal would be in this case to reduce this `nOut` parameter rather than using the default value.

References

[1] Kihyuk Sohn. “Improved Deep Metric Learning with Multi-class N-pair Loss Objective”. In: *NIPS*. 2016.