DeepVoice

Rémi Hutin, Rémy Sun, Raphaël Truffet
{Remi.Hutin, Remy.Sun, Raphael.Truffet}@ens-rennes.fr
Département informatique, ENS Rennes
Campus de Ker lann, Bruz, France

Guillaume Gravier, Vedran Vukotić
{guig, vedran.vukotic}@irisa.fr
Linkmedia project, IRISA
Campus de Beaulieu, Rennes, France

***Abstract*—To carry out a speaker recognition task, i.e., to identify the speaker in an audio signal, one must first obtain a serviceable representation of said signal. Improvements upon the raw numeric signal have been made over the years, one of which is the creation of a Gaussian mixture model supervector representing the probabilistic distribution of the signal's spectral features. Due to the enormous size of such supervectors, condensed forms such as i-vectors have been created to provide usable data for application tasks. Deep learning techniques have seen significant success in many classification tasks due to their ability to build upon successive layers of data representation. We will strive to explore the possibility of using such techniques to acquire a representation of supervectors that is at least competitive with i-vectors. We will then present the results of our experiments.**

***Index Terms*—Speaker recognition; Deep learning; i-vector; Latent representation**

I. Introduction

Speaker recognition refers to the all too important identification of a speaker from an unlabeled audio signal. However, the resolution of this essential issue is no trivial matter as it raises a number of questions along the way. The most immediate one, is the manner in which the data will be represented: raw numeric signal, Fourier transform, spectral representation?

Years of research into the issue have yielded a solution called *supervectors* which provide a probabilistic representation of the numeric signal's spectral features. Those supervectors however present the significant downside of being enormous and containing information not relevant to the task at hand, which makes them unfit for applications. What recent works have had success with is the extraction of more condensed representations from those supervectors named *i-vectors*.

Deep learning techniques have had tremendous success in a number of fields ranging from computer vision [9] to natural language processing [2] by building upon successive layers of data representation and inferring hierarchical dependencies within the data. It seems to us that such architectures would be especially adapted to the extraction of intermediate representations from *supervectors* and might provide representations that outperform the current state of the art *i-vectors*.

To begin with, we will give a brief exposition of signal representations used up until now II. Next, a brief explanation of the Deep Learning techniques we will need will be provided III. We will then give an overview of the study we plan to conduct IV. Then, we will present the issues we encountered V. Afterwards, we will detail our results VI. Finally, we will present our future work VII.

II. Sound signals

An audio signal used for speaker recognition tasks is technically an analogical signal. To allow computer processing, those signals are sampled into a numeric signal (discrete set of measures that can a discrete set of values). Such a representation is highly vulnerable to noise and transformations however, and more developed means of representation are usually used.

A. Cepstral analysis

The speech signal of a speaker is hardly suitable for statistical modeling or the calculation of a distance. In order to obtain a representation which is more compact and less redundant, we use a cepstral representation of the speech. The cepstrum of a signal $x(t)$ is defined by (1)

$$C(x(t)) = \mathcal{F}^{-1}(\ln(|\mathcal{F}(x(t))|) \tag{1}$$

where $\mathcal{F}$ is the Fourier transform. We can analyze a signal locally by applying a short-term window. Then, we extract a vector of cepstral coefficients of this part of signal. We repeat this operation for several windows, until the end of the signal is reached.

In the cepstral domain, the distance between two speech signals may be easily computed using the Euclidean distance.

B. Gaussian mixture model (GMM) and supervectors

A Gaussian mixture model (GMM) is a probabilistic model used to approximate a distribution of random variables as a sum of normal distributions ([1]). Here, we suppose that cepstral vectors of a signal follow a probability distribution that is specific to the given signal. This distribution is the one that we try to approximate with the GMM using a reference model. In fact, even if the distribution is specific to the given signal, the form of the distribution is universal for the natural language and called the Universal Background Model (UBM). The supervector is the vector that gather means of all the normal distributions of the GMM. Such models are used to represent signals in speaker recognition tasks (Figure 1)

C. I-vectors

A supervector presents a probabilistic representation of a signal's spectral features. However, these supervectors are enormous, which makes them unfit for applications and, since they contain information on the entire signal, contain information of little interest to us. That's why we want a more condensed version a these supervectors.

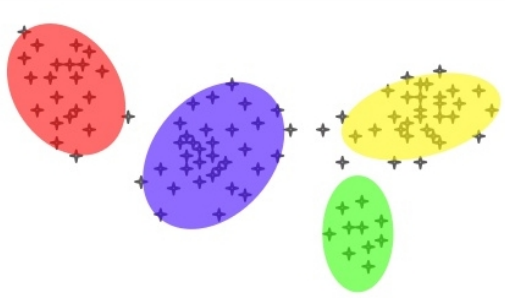


Figure 1: Gaussian Mixture Model with 4 Gaussian distributions (GMM)

Let M be the supervector, obtained as explained previously. M depends on the speaker and on the channel. We will perform a *linear projection*, to express M as

$$M = m + Tw \tag{2}$$

where

- m is a speaker and channel independent supervector
- T is a rectangular matrix, called *Total-variability matrix*
- w is an intermediate vector, or *i-vector*

The typical size of w is 60. These linearly extracted *i-vectors* provide lighter, and therefore more usable, data for application tasks.

D. Previous works

Extracting the i-vector is interesting for speaker recognition because it is easier to apply usual classifications method on it as cosine distance scoring (CDS) and support vector machines (SVM) [13, 3]. I-vectors are good tools for speaker recognition because they are channel independent and compact. We will try to keep these two characteristics in the new representation we will build using neural networks.

III. Use of neural networks

The recent success of deep learning techniques in various fields such as computer vision [9] and natural language processing [2] has sparked many explorations of their usefulness in classification tasks on *i-vectors* such as speaker recognition. Many architectures such as deep belief networks [5],[4], deep neural networks [5],[4], recurrent networks [12] or even a mix of deep neural networks and support vector machines [11] have been tested and yielded better results than previous techniques, though – to the best of our knowledge – none directly tackled the issue of supervectors' intermediate representation. Instead, all of the aforementioned work relied on pre-existing i-vector extraction techniques. We will explore the possibility of extracting a meaningful intermediate representation of supervectors through the use of deep architectures.

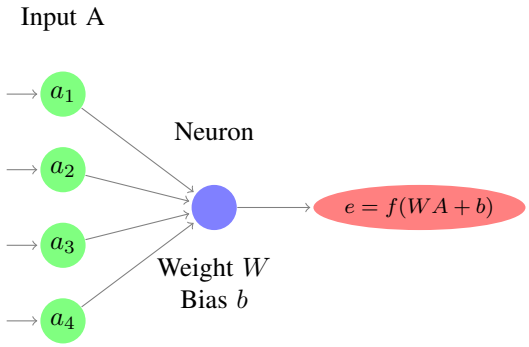


Figure 2: Formal neuron

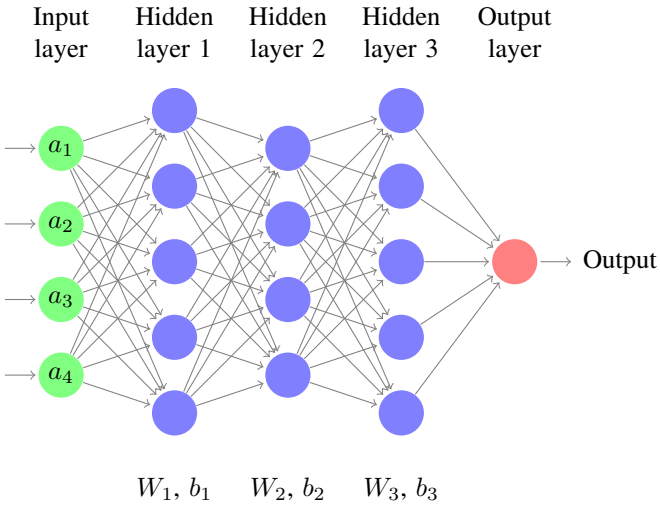


Figure 3: Deep neural network

A. Formal neuron

a) Neuron: Neural networks can be fairly complicated to understand all at once, and therefore we will start by explaining how one of its basic units work. A neuron (Figure 2) can be thought of as a function which takes an n-dimensional vector A as input and returns a scalar e as output. This function typically has two internal parameters which are a bias b and a weight matrix W. The function starts by calculating $WA + b$ before using a non-linear activation function (such as sigmoid or tanh), i.e.,

$$e = f(WA + b). \tag{3}$$

b) Adjusting the function: Our end goal is to have the neuron, and by extension the neural network (which we will introduce later), perform a certain task. The formal neuron "learns" by adjusting its function to perform better on a designated task. For simplicity's sake, we will first explain how a process – called "back-propagation" – works with a single neuron.

For instance, suppose we have a bi-dimensional vector given as input and that we want our neuron to return 1 if its two coordinates are the identical and -1 if they are not. A natural

way to evaluate how accurate our neuron consists in looking at the distance between its output e and the desired result r :

$$d(e) = |e - r| \tag{4}$$

We want to modify our neuron/function to minimize this distance. That means changing e, typically by gradient descent on function d derivative. Here, $\frac{\partial d}{\partial e} = r - e$, which means we want to "move" e in this direction. To this end, we modify our function's two internal parameters W and b. e, and by extension d, can actually be seen as a function of those two parameters: $d(W, b) = |e(W, b) - r|$. Therefore $\frac{\partial d}{\partial W} = \frac{\partial d}{\partial e}\frac{\partial e}{\partial W}$, $\frac{\partial d}{\partial b} = \frac{\partial d}{\partial e}\frac{\partial e}{\partial b}$. We then only need to compute new internal parameters W' and b' with 5 and 6.

$$W' = W + s\frac{\partial d}{\partial W} = W - s\frac{\partial d}{\partial e}\frac{\partial e}{\partial W} \tag{5}$$

$$b' = b + s\frac{\partial d}{\partial b} = b - s\frac{\partial d}{\partial e}\frac{\partial e}{\partial b} \tag{6}$$

What we just demonstrated was a simple back-propagation algorithm called gradient descent. This method is deeply flawed, but most state of the art back-propagation methods find their origins in this humble algorithm.

c) Neural network: Typically, a neural **network** (Figure 3) is made up of more than a single neuron. A neural layer refers to multiple neurons working on the same input (or parts of the same input) and producing an output that can be construed as some form of concatenation of their respective outputs. This output can be in turn regarded as an alternate representation of the input. Back-propagation for each neuron works the same way as it would if it were the only neuron calculating.

The notion of **deep** learning comes from the fact that the alternate representation computed by one layer A can be fed as input to another layer B. This allows networks to infer multiple levels of representation, same as one first processes simple geometrical forms before recognizing more complex compositions. Back-propagation is straight-forwardly computed on layer B. It is computed on layer A by looking at $\frac{\partial d}{\partial input_B}$ instead of $\frac{\partial d}{\partial e}$ which is simply calculated using the rule of chain derivation:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial x_1} \dots \frac{\partial x_k}{\partial x}. \tag{7}$$

What is of particular interest to us is the intermediate representations the network acquires this way. Indeed, our goal is to obtain a representation of supervectors that outperforms the linear extraction of i-vectors.

B. Autoencoders

An autoencoder [6] is a neural network with a particular architecture which is defined below.

a) A default task for a different goal: Neural networks' ability to extract internal meaningful representation from the original raw data is very enticing. Indeed, providing a meaningful representation of the initial data is one of the biggest hurdles of classical machine learning. However, neural networks need to be trained on a task, and require labeled data to that end.

Since we are no longer interested in the specific task needed to train the network, it is possible to default to a task that is not interesting in itself but might require the network to learn salient features of the data in order to be completed. A very natural one is to match the output to the input, which would not require man made labeling. Hopefully, the learned representation will capture features specific to the input so that it may be reconstructed.

b) Structure: Typically, the **input** is *encoded* (8) into a **latent representation** which is then *decoded* (9) into an **output** that should match the input as closely as possible, hence the autoencoder denomination (Figure 4). The main danger of such an approach is having the networks that does not transform the input at all and yields a latent representation that is no different to the raw input data. A multitude of solutions exist to avoid such a scenario:

- Compress the input into a latent representation of lower dimension
- Put the input through a corrupting input layer: the autoencoder cannot just do nothing and give back the same thing because it never had the actual input to begin with. This particular structure is called denoising auto-encoder.
- Use various regularization methods.

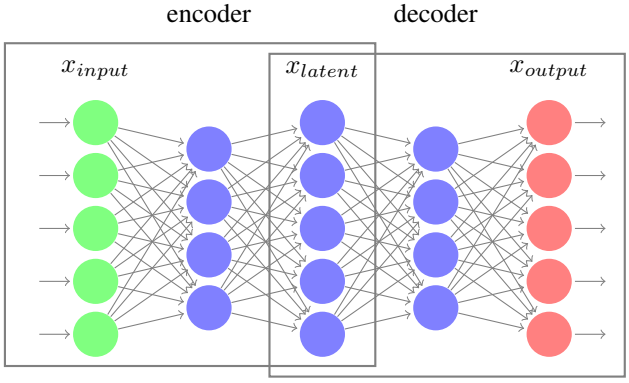


Figure 4: Structure of an autoencoder

$$x_{latent} = encoder(x_{input}) \tag{8}$$

$$x_{output} = decoder(x_{latent}) \simeq x_{input} \tag{9}$$

c) Why use autoencoders: Autoencoders can be used for denoising, using corrupted data as input and the original data as objective. Autoencoders may also be very useful to learn a representation. In fact, the latent representation contains enough information to reconstruct the data.

In our context, autoencoders could be used to find a representation of a speaker, by giving to the autoencoder a supervector from a speaker as input, and an another supervector from the same speaker as objective. This idea is based on considering the within-speaker variability as a noise, and using a denoising autoencoder. Repeating this with several pairs of supervectors, we may learn a representation of the speaker. This new representation, as well as i-vectors, is more condensed than supervectors.

Since the problem is symmetrical, we can perform the learning in both directions. One way to perform this training is to use an autoencoder that reconstruct each supervector from the other using tied weights.

C. Tied weight autoencoder

We talk about tied weight autoencoder when the same weights appears in two different places in the autoencoder. For example, in the architecture proposed in [15], the weights in light blue in the upper part of the autoencoder are the same as the weights in light blue in the lower part. So, if the function that transform the upper first hidden layer to the upper part of the representation is

$$h_1(x) = f(W \times x + b_1) \tag{10}$$

then the function that transform the lower part of the representation to the next layer is

$$h_2(x) = f(W^T \times x + b_2) \tag{11}$$

In this architecture (see Figure 5), the pink weights are also tied.

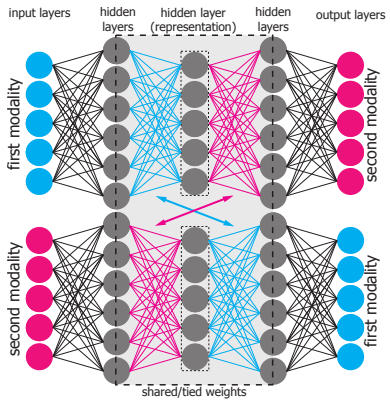


Figure 5: Symmetrical and bidirectional learning architecture using tied weights

This architecture learns a joint representation of the two modalities which gives encouraging results multi-modal query expansion [15]. In our context, the two modalities will be two supervectors from the same speaker.

IV. Method

As explained earlier, supervectors computed from Gaussian mixture models provide useful features of a given signal. i-vectors have built upon this representation to extract a more compact representation that better represent the specificities of the signal. One task that naturally comes to mind is deciding whether two signals have been said by the same person. Our suspicion is that linear extraction of i-vectors provides suboptimal results for such computations. Therefore, we will use neural networks to extract an intermediate representation of each signal that might be better suited to this particular problem.

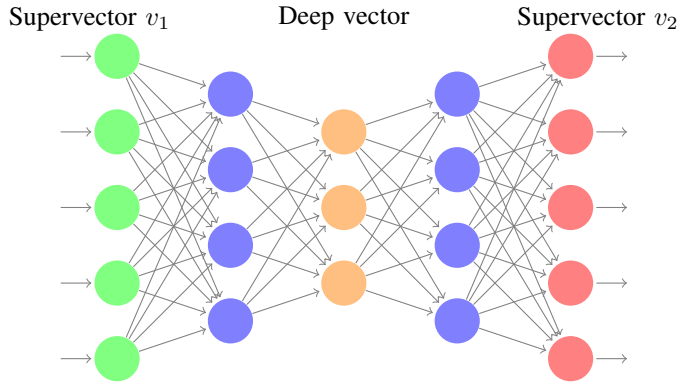


Figure 6: General prospective model

A. Prospective method

a) Phasing out non-speaker dependant noise: In order to extract a representation of supervectors that makes speaker dependent features of the given signal more salient, we use a model derived from the design of a denoising autoencoder. Indeed, we have no need for any information pertaining to what is actually being said or what kind of microphone was used: this is all noise to us. Therefore, if two supervectors v_1 and v_2 originate from the same speaker, it seems reasonable to think of them as the same original "speaker" sound vector which has polluted by non-speaker dependant noise.

The latent representation thusly extracted should present salient features specific to the studied speaker. An autoencoder trained that way could allow for a sort of projection that we wish to study.

b) Evaluating speaker-dependant similarities: We wish to acquire intermediate representations that draws different signals spoken by the same speaker closer together according to some measure of distance, therefore allowing for an easy classification. The distances we will mostly consider during our study will be angular distance and Euclidean distance.

To this end, we will have to carefully chose the distance function used to evaluate the "quality" of our output in order to make sure we will be optimizing this particular aspect of the acquisition.

c) Prospective general model: We will therefore train an autoencoder on supervectors. Given good results of [15], we will tie all but the extreme weights to reflect the symmetrical nature of the task at hand. A rough outline of the general model is shown in Figure 6.

B. First experimentation

In light of the difficulties inherent to the training of large neural networks, a preliminary study on a small dataset will first be conducted.

1) Dataset:

a) Processing raw data: 15308 audio signals were extracted from BFMTV's various programs and labeled with the name of their respective speaker. Those sound files were then processed into 15308 supervectors of length 2304 with the *AudioSeg* tool. We then create 1 839 235 pairs of supervectors that originate from the same speaker.

b) Input: For each computed pair, we feed both of its supervectors to the neural network as input. Therefore, we train the network on 3 678 470 supervectors of size 2304.

c) Output and task: The neural network outputs a vector of size 2304 that we try to match as closely as possible to the supervector the original input was paired with.

2) Evaluating results:

a) Evaluation method: We will first try to pinpoint a threshold on the distance between two vectors which separates pairs of vectors labeled to the same speaker and labeled to two separate speakers. To this end, we will compute the distance (discussed in IV-A) between every pair of vectors labeled to the same speaker. The maximum of those distances is then computed and chosen as our **threshold**.

We know by construction that every pair labeled to the same speaker is below the chosen threshold. To evaluate the relevance of the studied representation we then count the number of pairs of unmatched vectors whose corresponding distance is below our threshold. and would be misclassified if our threshold is was used as a classification method.

In order to compare our results to previous work, we have been provided a list of 9049 pairs of sound files that belong to our dataset. We have access to results on this list that come from another method. Firstly, we will not use a training set disjoint from the test set. This allows a sanity check, and it is enough to improve the hyper-parameters.

b) Comparing to i-vectors: To substantiate our claim that i-vector extraction does not lead to a clustering of same speaker vectors, we will apply the evaluation method outlined above to both our extracted intermediate vectors and standard *i-vectors*. Those intermediate representations will be extracted from source files using the tool ALIZE [8].

V. Technical issues

We had to face several technical issues:

- The first dataset we had access to was not good, a occured problem during the extraction. We ran a few experiments, and we began to think that our method doesn't allow the separability of the speakers before finding the origin of the problem.
- As a training proceeds in many optimizations, repeated several times, its execution is very long (several hours).
- A supervector is a big object (vector of size 2304 in our case) and we train on many of them, so we need a lot of memory.
- Because of the two previous points, we needed more powerful machines. We got access to INRIA's servers, but only for two weeks because of administrative delays

Even if we started out with a fairly small training set, and small data, it remains big. All these prevented us from doing many experiments. We will still present some of them in the next section.

VI. Results

A. Parameters

We ran several experiments, using different values for the parameters mentioned previously. Here is an exhaustive list of the parameters that we changed from one experiment to another.

- Number of layer: we used from 3 to 5 hidden layers in our architecture.
- Size of the layers: we used different sizes for the hidden layers of our architecture.
- Tied weights: we tried different configurations, in which we tied or not some layers of our architecture.
- Optimizer: we tried two different optimizing methods. First, we used a gradient descent, in which the partial derivatives of the objective function are locally computed, that provides us the direction in which the function decreases the most. The second optimizer is the Adam optimizer [7], which is also gradient-based, but moreover on an adaptive moment estimation.
- Dropout [14]: in order to avoid over-fitting, it is good to limit the size of the neural networks. This is done by dropping randomly some nodes. At each stage, a node is kept with a probability keep_proba. We tried several values of keep_proba.

After the training, we can compute, for each input, the latent representation $x_{latent} = encoder(x_{input})$. We will call it deep vector, by opposition with i-vectors. Then, Euclidean distances and cosine distances can be computed for every pair of deep vectors.

B. Graphical representation

In order to clearly represent our results, we used Detection Error Tradeoff (DET) graphs. A DET graph plots the false rejection rate vs. the false acceptance rate. Knowing the distances between deep vectors of a list of pairs and a value of threshold, we can compute the classification of the pairs (same speaker or different speakers).Then, as we know the speaker of each deep vector, we can check for every pair whether the result is correct or not. This gives us a false acceptance rate and a false rejection rate. The DET graph can be obtained by doing this for several values of threshold.

C. Results

In order to check if we are on the good way, we plot histograms of the results of the first experiment, to check that pairs from the same speaker don't give results to similar from pairs from two different speakers. We finally reached a repartition in which cosine distances are closer to 1 when

the deep vectors come from the same speaker (Figures 7 and 8, which is what we expected. The results for the Euclidean distance are also coherent.

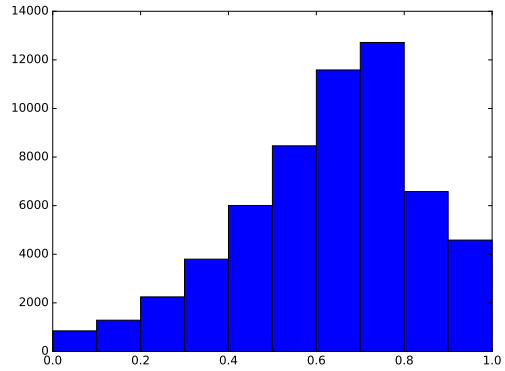


Figure 7: Repartition of the cosine distance between deep vectors from the same speaker

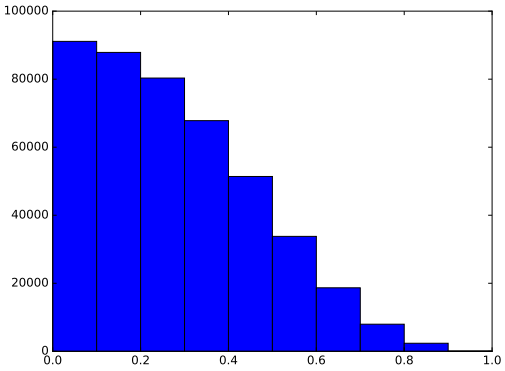


Figure 8: Repartition of the cosine distance between deep vectors from different speakers

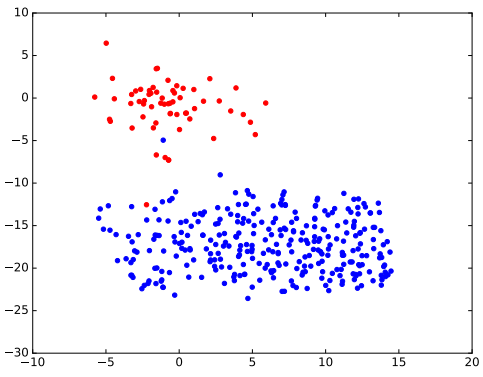


Figure 9: t-SNE of the deep vectors of two different speakers

Figure 9 also encourages us to believe that the deep vectors will provide a good separability. In fact, t-Distributed Stochastic Neighbor Embedding (t-SNE) provides a reduction of the vectors in a 2 dimensions space in order to have a good visualization of high dimensional data [10].

We ran many experiments and therefore cannot present them all. We will only present the most interesting. The configuration and DET graphs of our experiments are presented in the appendix. Every DET graph plots 3 DET curves:

- `g64_norm`: baseline, using a distance computed on the GMMs
- `expi_dist`: our results, using Euclidean distance
- `expi_cos`: our results, using Cosine distance

Here are some details about these results.

1) Experiment 0: This is the first training experiment we ran. We can see than both curves we obtained are mainly above the g64 curve, which means that our results are worse than our reference. However, these results were promising as we knew that many parameters remained to be optimized.

2) Experiment 1: We ran the exact same experiment, using this time the Adam optimizer rather than the gradient descent optimizer. The results we obtained were significantly better, therefore we decided to only use this optimizer.

3) Experiment 2: We changed the size of our hidden layers, and our results were slightly better.

4) Experiment 3: We decided to add two more hidden layers to our architecture. However, the results we obtained were really bad, probably because we didn't train our network long enough.

5) Experiment 4: We used the same experiment as previously, and we tied 2nd layer to the 6th layer. Our results were now much better.

6) Experiment 7: We came back to a simple architecture, as during our first experiment, and we ran an experiment that used different data for the training phase and for the validation phase. Our results were not as good as before, which was predictable, but they were encouraging.

VII. Further work

As mentionned before, the last experiment we ran used different data for the training phase and for the validation phase. In figure 10, we compare these results with the i-vectors, using the same data sets. We can notice that our results are not yet as good as the i-vectors.

We didn't have enough time to run further experiments using these different data sets. Our future work will consist in running additional experiments, using more hidden layers for instance, in order to finally have better results than i-vectors. We are confident that we will soon be able to obtain better results than i-vectors.

VIII. Conclusion

We proposed a new approach in order to use deep learning techniques in a speaker recognition task. Our objective was to obtain a representation of a speaker that is at least competitive with i-vectors. We first ran experiments using the same data

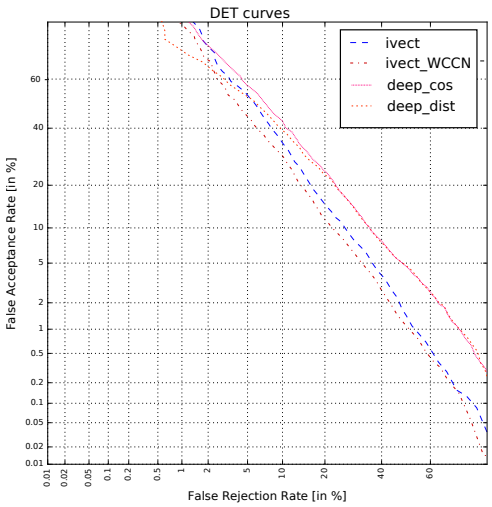


Figure 10: Deep-vectors vs. i-vectors

for the training phase and for the validation phase, as we were more likely to obtain good results. We then ran an experiment with different data for the training phase and for the validation phase, and compared the results with the i-vectors. We unfortunately didn't have time to run additional experiments. Although the i-vectors approach still has better results than our, we are confident that additional work and optimization could make deep-vectors more efficient than i-vectors.

Appendix

References

[1] Frédéric Bimbot et al. "A Tutorial on Text-Independent Speaker Verification". In: *EURASIP Journal on Advances in Signal Processing* 2004.4 (2004), p. 101962. ISSN: 1687-6180. DOI: 10.1155/S1110865704310024. URL: http://dx.doi.org/10.1155/S1110865704310024.

[2] Antoine Bordes et al. "Joint Learning of Words and Meaning Representations for Open-Text Semantic Parsing." In: *AISTATS*. Vol. 351. 2012, pp. 423–424.

[3] Najim Dehak et al. "Front-end factor analysis for speaker verification". In: *IEEE Transactions on Audio, Speech, and Language Processing* 19.4 (2011), pp. 788–798.

[4] Omid Ghahabi and Javier Hernando. "Deep belief networks for i-vector based speaker recognition". In: *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2014, pp. 1700–1704.

[5] Omid Ghahabi and Javier Hernando. "Deep Learning for Single and Multi-Session i-Vector Speaker Recognition". In: *CoRR* abs/1512.02560 (2015). URL: http://arxiv.org/abs/1512.02560.

[6] G. E. Hinton and R. R. Salakhutdinov. "Reducing the Dimensionality of Data with Neural Networks". In: *Science* 313.5786 (2006), pp. 504–507. ISSN: 0036-8075. DOI: 10.1126/science.1127647. eprint: http://science.sciencemag.org/content/313/5786/504.full.pdf. URL: http://science.sciencemag.org/content/313/5786/504.

[7] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980 (2014). URL: http://arxiv.org/abs/1412.6980.

[8] Anthony Larcher et al. "ALIZE 3.0-open source toolkit for state-of-the-art speaker recognition." In: *Interspeech*. 2013, pp. 2768–2772.

[9] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[10] Laurens van der Maaten and Geoffrey E. Hinton. "Visualizing High-Dimensional Data Using t-SNE". In: *Journal of Machine Learning Research* 9 (2008), pp. 2579–2605.

[11] Fred Richardson, Douglas Reynolds, and Najim Dehak. "Deep neural network approaches to speaker and language recognition". In: *IEEE Signal Processing Letters* 22.10 (2015), pp. 1671–1675.

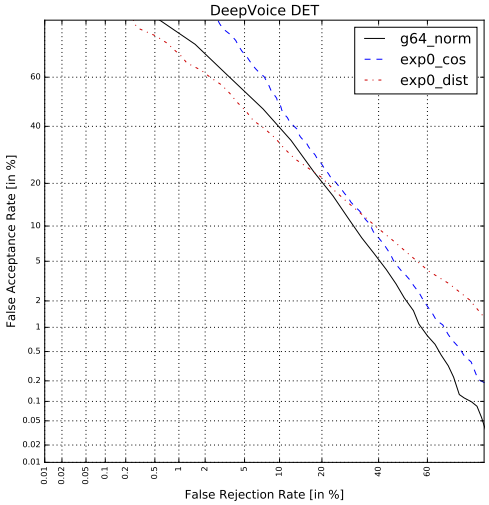
[12] George Saon et al. "The IBM 2016 English Conversational Telephone Speech Recognition System". In: *CoRR* abs/1604.08242 (2016). URL: http://arxiv.org/abs/1604.08242.

[13] Mohammed Senoussaoui et al. *An i-vector Extractor Suitable for Speaker Recognition with both Microphone and Telephone Speech*.

[14] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *J. Mach. Learn. Res.* 15.1 (Jan. 2014), pp. 1929–1958. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=2627435.2670313.

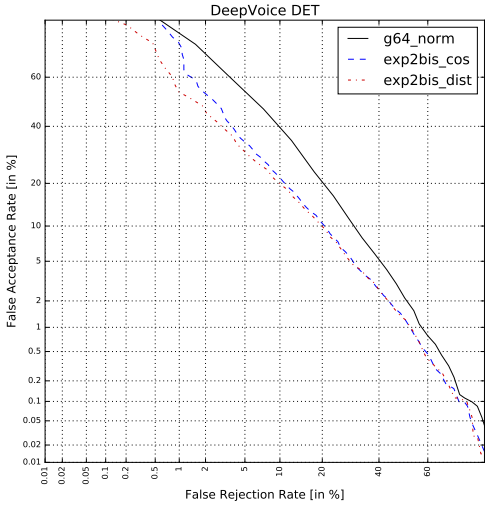
[15] Vedran Vukotic, Christian Raymond, and Guillaume Gravier. "Bidirectional Joint Representation Learning with Symmetrical Deep Neural Networks for Multimodal and Crossmodal Applications". In: *ICMR*. ACM. New York, United States, June 2016. URL: https://hal.inria.fr/hal-01314302.

Number of layers	5				
Size of layers	2304	1000	50	1000	2304
Tied weights	No				
Optimizer	Gradient Descent				
Dropout	0.90				



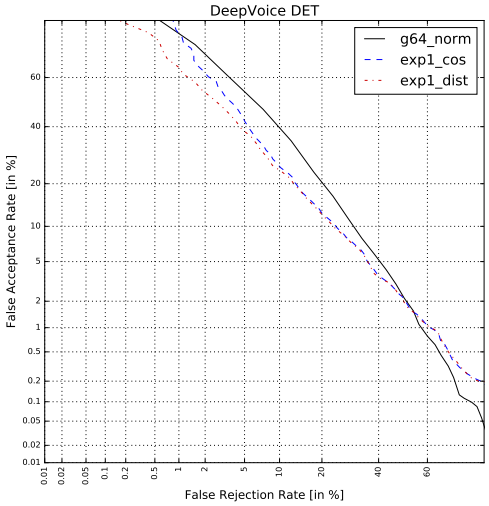
Experiment 0

Number of layers	5				
Size of layers	2304	480	100	480	2304
Tied weights	No				
Optimizer	Adam				
Dropout	0.90				



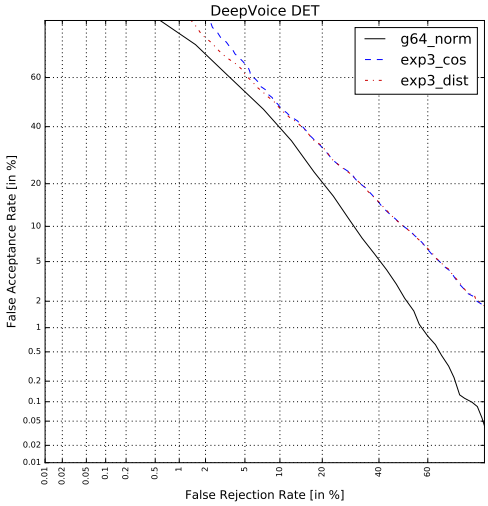
Experiment 2

Number of layers	5				
Size of layers	2304	1000	50	1000	2304
Tied weights	No				
Optimizer	Adam				
Dropout	0.90				



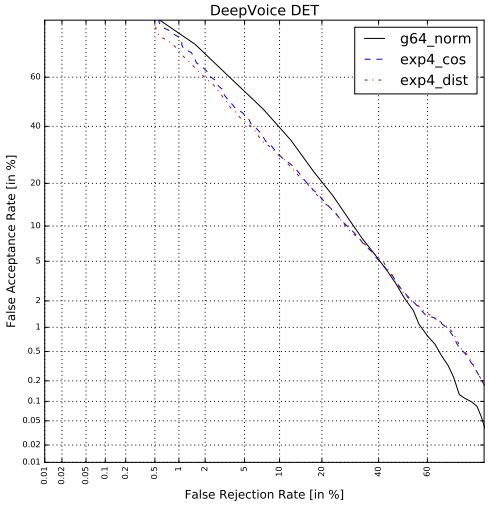
Experiment 1

Number of layers	7						
Size of layers	2304	720	225	70	225	720	2304
Tied weights	No						
Optimizer	Adam						
Dropout	0.90						



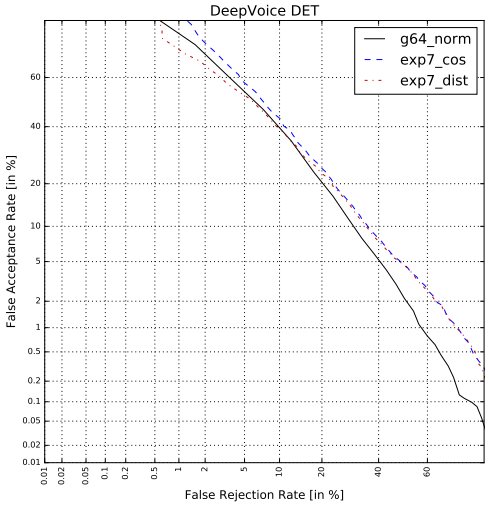
Experiment 3

Number of layers	7						
Size of layers	2304	720	225	70	225	720	2304
Tied weights	Yes						
Optimizer	Adam						
Dropout	0.90						



Experiment 4

Number of layers	5				
Size of layers	2304	500	80	500	2304
Tied weights	No				
Optimizer	Adam				
Dropout	0.80				



Experiment 7