INTERSPEECH 2019
September 15–19, 2019, Graz, Austria

X-vector DNN Refinement with Full-length Recordings for Speaker Recognition

Daniel Garcia-Romero, David Snyder, Gregory Sell, Alan McCree, Daniel Povey, and Sanjeev Khudanpur

Human Language Technology Center of Excellence & Center for Language and Speech Processing
The Johns Hopkins University, Baltimore, MD 21218, USA

dgromero@jhu.edu

Abstract

State-of-the-art text-independent speaker recognition systems for long recordings (a few minutes) are based on deep neural network (DNN) speaker embeddings. Current implementations of this paradigm use short speech segments (a few seconds) to train the DNN. This introduces a mismatch between training and inference when extracting embeddings for long duration recordings. To address this, we present a DNN refinement approach that updates a subset of the DNN parameters with full recordings to reduce this mismatch. At the same time, we also modify the DNN architecture to produce embeddings optimized for cosine distance scoring. This is accomplished using a large-margin strategy with angular softmax. Experimental validation shows that our approach is capable of producing embeddings that achieve record performance on the SITW benchmark.

Index Terms: speaker recognition, x-vectors, deep neural networks, metric learning.

1. Introduction

A key aspect of the success of DNNs is that they can be trained end-to-end to directly solve the task of interest. In this way, the classification and representation stages can be jointly optimized to yield improved performance. For applications like face or speaker recognition, where the set of training classes is most likely disjoint from those at test time (i.e., few-shot-learning), the end-to-end approaches use metric learning loss functions. For example, given two audio recordings, the DNN outputs a similarity score that reflects the likelihood that they belong to the same speaker or not (i.e., same vs different) [1]. Another popular alternative is to use triplet loss [2]. Examples of end-to-end DNNs for text-dependent tasks can be found in [3, 4, 5, 6]. Also, examples for text-independent tasks with short utterances can be found in [1, 4, 6]. Recently, good progress has been presented in [7] for the long duration (e.g., minutes of speech) text-independent case. However, successful implementations of end-to-end DNNs usually require cumbersome sample mining strategies (e.g., hard or semi-hard examples) that strongly influence the performance and convergence speed [6].

One way to avoid this challenge is to train DNNs to produce embeddings using a classification loss (e.g., multiclass cross entropy). In this way, the goal of training the network is to produce embeddings that generalize well to speakers beyond those in the training set. Also, we would like the embeddings to summarize speaker characteristics over the entire recording. Once the DNN is trained, the embeddings are extracted for each recording and compared using a similarity metric. The metric learning process is disjoint from the DNN training and it is typically done using some variant of probabilistic linear discriminant analysis (PLDA) [8, 9, 10, 11].

A successful example of this paradigm is the x-vector system presented in [12, 13], and independently validated in [14, 15]. The x-vector DNN uses a temporal pooling layer that computes the mean and standard deviation of an input sequence to capture the speaker characteristics over the entire recording. Ideally, the DNN embeddings should be trained on speech segments that matches the range of durations we expect to encounter at test time. However, GPU memory limitations and convergence speed force a tradeoff between minibatch size and maximum sequence length. As an engineering compromise, the current x-vector implementation [13] relies on short segments to train the embedding (e.g., up to 10 second segments). Unfortunately, this can introduce an undesirable mismatch between training and inference time.

In this work, we propose a refinement stage of a subset of the DNN parameters (after the pooling layer) to reduce this mismatch. At the same time, we also modify the DNN architecture in a way that facilitates embedding comparisons with cosine distance (i.e., dot product of normalized vectors). We consider this convenient as it enables very fast large scale comparisons [16], easier privacy protection [17], and has been shown to generalize well [6, 18, 19].

Our experimental setup uses VoxCeleb data [20, 21] to train the x-vector DNN and evaluates performance on the speakers in the wild (SITW) core-core task [22]. The results show that our full recording refinement stage is capable of producing embeddings that achieve record performance on the SITW benchmark.

2. X-vector System

The x-vector system is a DNN that computes speaker embeddings from variable-length speech segments. For this work, we use an extended version of the system in [13], which is the default architecture in the public Kaldi recipes. Table 1 summarizes the extended network architecture and Figure 1(a) shows its diagram. The layers can be grouped into three blocks according to their main functionality. The first group comprises layers 1 to 9 and is a collection of TDNN (1D convolutions) and dense layers. The hierarchical structure of the convolutions provides an efficient way to process an extended input context of 230 ms with a reduced number of parameters. Given a sequence of T input features of dimension F, this block maps patches of $23 \times F$ features into a sequence of T high dimensional vectors (1500 dim). The second group, layer 10, performs a temporal pooling operation over this sequence by computing the mean and standard deviation across each dimension. These two statistics are concatenated together into a fixed-dimensional vector of size 3000, which summarizes the entire input recording. The

http://dx.doi.org/10.21437/Interspeech.2019-2205

Table 1: *Extended TDNN x-vector architecture*

Layer	Layer Type	Context	Size
1	TDNN-ReLU	t-2:t+2	512
2	Dense-ReLU	t	512
3	TDNN-ReLU	t-2, t, t+2	512
4	Dense-ReLU	t	512
5	TDNN-ReLU	t-3, t, t+3	512
6	Dense-ReLU	t	512
7	TDNN-ReLU	t-4, t, t+4	512
8	Dense-ReLU	t	512
9	Dense-ReLU	t	1500
10	Pooling (mean+stddev)	Full-seq	3000
11	Dense(Embedding)-ReLU		512
12	Dense-ReLU		512
13	Dense-Softmax		Num. spks.

third group, layers 12 to 13, is a feed-forward network with a bottleneck layer and serves as a classifier that outputs posterior probabilities for the training speakers. The x-vector is extracted from layer 11 prior to the ReLU non-linearity. The bottleneck structure of the net is used to achieve a dimensionality reduction of the embedding (512 dimensions). The total number of parameters of the DNN used in the experiments (using 7,168 training speakers) is approximately 8 million, of which only 4 million are needed for extracting the x-vector.

3. Angular softmax with additive margin

Since our goal is to obtain embeddings that can be directly compared using cosine distance, it is important to use that metric when training the network. The use of cosine similarity as the logit input to the softmax layer is referred to in the literature as angular softmax [23]. A number of variants have been proposed [24, 25] to reduce the interclass variance by introducing the notion of a margin penalty to the target class logit. Effective applications for speaker recognition have been presented in [18, 19]. In this work, we use the additive margin variant [24] due to its good performance and ease of implementation. The corresponding loss function is

$$\mathcal{L} = -\frac{1}{n}\sum_{i=1}^{n}\log\frac{e^{s(\cos\theta_{y_i}-m)}}{e^{s(\cos\theta_{y_i}-m)}+\sum_{j\neq y_i}e^{s\cos\theta_j}}, \quad (1)$$

where $\cos\theta_{y_i} = \mathbf{w}_{y_i}^T\mathbf{f}_i / \|\mathbf{w}_{y_i}\|\,\|\mathbf{f}_i\|$, $\mathbf{w}_{y_i}$ is the weight vector of class y_i, and $\mathbf{f}_i$ is the input to the layer for example i. Also, s is an adjustable scale factor and m is the penalty margin. To accelerate convergence, we follow the practice of fixing s to a predefined value. Also, during training, m is linearly increased from 0 to the desired margin value following the annealing strategy in [26].

4. Training

4.1. Data preparation

We combined the VoxCeleb1 [20] and VoxCeleb2-dev [21] data (removing the 60 speakers that overlap with SITW) to obtain 155,482 original recordings from 7,183 speakers. We refer to this combination as VoxCeleb. Note that by recording we mean all the labeled audio segments of a particular speaker from a video. After augmentation and removal of recordings with less than 4 seconds of speech (as determined by the Kaldi energy VAD) we obtained a total of 485,385 recordings that we use for training. To augment a recording, we randomly applied either babble noise, music, general noise, or reverberation as described in [13]. A validation set of 1,000 recordings (each from a different speaker) was set aside to monitor DNN training performance.

The audio was processed at 16 KHz sampling rate and parameterized into 60 MFCCs using a 25 ms window every 10 ms with 60 mel bins over the spectral band of 20–7600 Hz.

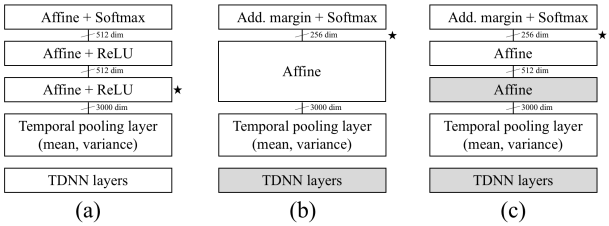

Figure 1: *Block diagrams of the three network architectures used in this work. Gray boxes indicate that the parameters are frozen after pre-training. The position of the star indicates where the embedding is extracted.*

4.2. DNN training

4.2.1. Short segment training

Although we would ideally seek to minimize the mismatch between training and inference, in practice, there are two main reasons why we restrict training to short segments. First, longer batch sizes become feasible in limited GPU memory. Second, longer sequences are easier to classify and the network tends to overfit to them. As a compromise, based on experimentation, we pick examples that range from 2 to 4 seconds (200 to 400 frames) along with a minibatch size of 64. The DNN training speech segments are uniformly sampled per speaker from the 484,385 available recordings. We used 10,000 examples per speaker. The network is trained for 4 epochs using natural-gradient stochastic gradient descent [27]. Figure 2 depicts the accuracy profile for the train and the validation set as a function of the number of days needed to complete the 4 epochs. It is quite remarkable that an accuracy of around 85% can be attained for such a challenging task.

4.2.2. Full recording refinement

The engineering compromise of using 2 to 4 second segments to train the embedding can be relaxed by freezing the DNN components that are responsible for most of the memory, as well as controlling the network capacity so that it does not overfit to longer sequences. In this way, we can use the pre-trained network with short segment as an initialization for a full recording refinement of a subset of the network parameters.

For this purpose, we explore architectures (b) and (c) of Figure 1. Both of them freeze all pre-pooling TDNN layers (as they are responsible for most of the memory) and replace the standard affine+softmax (i.e., multiclass logistic regression) layer with the angular softmax layer described in Section 3. They differ in that architecture (b) replaces layers 12 and 13 from Table 1 with a single affine transformation, whereas (c) freezes the affine component from which the conventional x-vector is extracted, and removes the ReLU components. Both

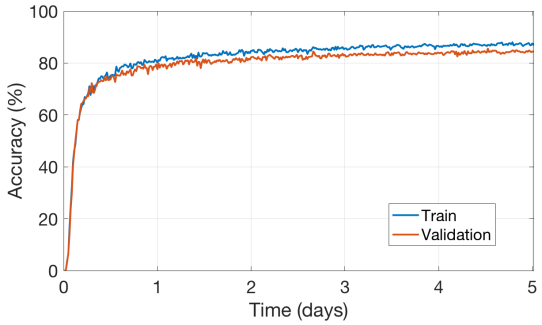

Figure 2: *Accuracy profile for train and validation sets.*

systems produce embeddings of 256 dimensions optimized for a cosine distance metric. The use of an additive margin penalty during training ensures that the learning process is challenging, even for full length sequences of minutes. Also, the reduction of the number of parameters after the pooling layer (about half of those used in architecture (a)) helps mitigate potential overfitting. Using full sequence refinement and angular softmax has the effect of reducing the mismatch between training and inference, and produces embeddings that can be compared using a simple cosine distance.

We used the 485,385 VoxCeleb recordings for refinement and validation. The scale factor s of the angular softmax was first learned by training the net with a margin $m = 0$ and checking the accuracy on the recordings of the validation set. We were mostly interested in understanding a reasonable range for this parameter. Multiple repetitions of this exercise returned values in the range of 18 to 21. The scale factor does not seem very sensitive and we decided to fix it to 20. Following the findings in [24], we explored margin values around $m = 0.35$. Further experimentation using the validation set indicated that a value of 0.5 was slightly better for our setup; although this parameter was also not very sensitive. During training, m was initialized to 0 and increased by 0.025 every other epoch (taking a total of 40 epochs to reach its target value of 0.5). We refined the trainable parameters of the nets for 200 epochs, and both architectures (b) and (c) were able to attain an accuracy around 97% on the validation set.

5. Scoring

5.1. HT-PLDA scoring

For the baseline x-vector (architecture (a)), we used the generative Heavy Tailed PLDA (HT-PLDA) classifier described in [11], as it was shown to outperform a Gaussian PLDA system. The HT-PLDA was trained using the x-vectors from the 485,385 VoxCeleb recordings that we processed by centering and whitening, but no unit-length projection was applied [10]. We used a speaker subspace of dimension 150 and the number of degrees of freedom $\nu = 200$. The evaluation data was also centered and whitened using the parameters learned from the VoxCeleb set.

5.2. Cosine scoring

When using cosine scoring for the embeddings optimized for this metric (architectures (b) and (c)), we simply compare them by evaluating the cosine distance. When we report cosine scoring results for the standard x-vectors (not optimized for this metric), we first center and project them down to 256 dimensions, using LDA learned on VoxCeleb, and then compute the cosine distance.

Table 2: *Performance on SITW core-core task.*

System	Scoring	mDCF(10^{-3})	mDCF(10^{-2})	EER(%)
(a)	HT-PLDA	0.35	0.21	1.8
(a)	Cosine	0.39	0.25	2.8
(b)-1	Cosine	0.27	0.18	2.4
(b)-2	Cosine	0.32	0.20	2.0
(c)	Cosine	0.29	0.18	2.0

6. Experimental setup

Our evaluation setup uses the Speakers in the Wild (SITW) core-core task [22] with 16 KHz sampling rate. It consists of unconstrained audio from video of English speakers, with naturally occurring noises and reverberation, as well as device and codec variability. Both enroll and test utterances vary in length from 6–240 seconds. The evaluation protocol yields 3658 target trials and 718130 non-target trials.

We report results in terms of equal error-rate (EER) and minimum normalized detection cost (mDCF) at two operating points with $P_{\text{Target}} = 10^{-2}$ and $P_{\text{Target}} = 10^{-3}$. In both cases $C_{FA} = C_{MISS} = 1$.

7. Results

The performance of the baseline x-vector system with the HT-PLDA classifier is shown in the first row of Table 2. To the best of our knowledge, this result represents a new state-of-the-art for the SITW benchmark. We attribute this to the ability of the extended x-vector architecture (Figure 1(a)) to take advantage of the large set of speakers in VoxCeleb, as well as the use of 16 KHz sampling rate. The second row shows that applying the typical LDA followed by a cosine distance classifier (instead of some PLDA variant) lags behind. This was expected, but is shown to facilitate comparison with our proposed approach.

In rows three and four, we present the results of using architecture (b) during the full recording refinement and cosine scoring. System (b)-1 corresponds to a network refined using the margin penalty described in Section 4.2.2. System (b)-2 shows the same architecture but with no margin penalty applied during the refinement. Using the margin penalty seems highly beneficial at operating points that require low false alarms. However, this performance boost is obtained at the expense of some slight degradation at the EER point. Comparing system (b)-1 with the baseline x-vector system, we can observe significant gains (20% and 15% relative improvement at mDCF(10^{-3}) and mDCF(10^{-2}) respectively) and some degradation at EER. To better illustrate the performance trend, Figure 3 show the DET plot for these two systems (as well as an equal weight sum fusion). It is clear that system (b)-1 is tilted in a way that favors the low false alarm region. More importantly, these competitive results validate our approach as a way to obtain embeddings that work well for long recordings with cosine distance.

The final row indicates that an equivalent performance can be achieved with system (c), which also makes use of the margin penalty. It is important to note that this architecture is basically learning an affine projection on top of the conventional x-vector.

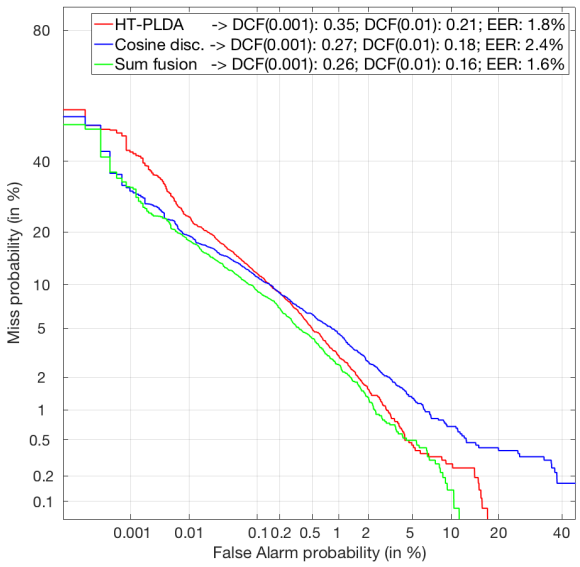

Figure 3: *DET plot*

After applying it, we obtain a new embedding of smaller dimension (256 dimensions) that can be optimally compared using a cosine distance metric. In this way, this can be thought of as an improved replacement over the conventional LDA followed by cosine distance.

8. Conclusions

We have explored a refinement stage of an extended x-vector architecture using full recordings. The goal was to mitigate the duration mismatch between training and inference, and produce embeddings that can be directly compared using cosine scoring. Our approach freezes all the pre-pooling parameters of the x-vector extractor, simplifies the post-pooling classifier, and trains its parameters using the full recordings. We used angular softmax with a margin penalty. Our experimental setup uses VoxCeleb as a training set and the SITW core-core benchmark. Our results indicate that an affine transformation after the pooling layer is capable of transforming the embedding space to directly use cosine scoring. Relative improvements of around 15 to 20% are observed at low false alarm operating points with respect to a very strong state-of-the-art baseline. To the best of our knowledge, these numbers represent the best published results on the SITW benchmark.

9. References

[1] D. Snyder, P. Ghahremani, D. Povey, D. Garcia-Romero, Y. Carmiel, and S. Khudanpur, "Deep neural network-based speaker embeddings for end-to-end speaker verification," in *SLT*, 2016.

[2] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *CVPR*, 2015.

[3] G. Heigold, I. Moreno, S. Bengio, and N. Shazeer, "End-to-end text-dependent speaker verification," in *ICASSP*, 2016.

[4] C. Li, X. Ma, B. Jiang, X. Li, X. Zhang, X. Liu, Y. Cao, A. Kannan, and Z. Zhu, "Deep speaker: an end-to-end neural speaker embedding system," in *CoRR*, vol. abs/1705.02304, 2017.

[5] F. A. R. R. Chowdhury, Q. Wang, I. Lopez-Moreno, and L. Wan, "Attention-based models for text-dependent speaker verification," *ICASSP*, 2018.

[6] L. Wan, Q. Wang, A. Papir, and I. Lopez Moreno, "Generalized end-to-end loss for speaker verification," in *ICASSP*, 2018.

[7] J. Rohdin, A. Silnova, M. Diez, O. Plchot, P. Matejka, and L. Burget, "End-to-end DNN based speaker recognition inspired by i-vector and PLDA," in *ICASSP*, 2018.

[8] S. Ioffe, "Probabilistic linear discriminant analysis," in *ECCV*, 2006.

[9] P. Kenny, "Bayesian speaker verification with heavy-tailed priors." in *Odyssey*, 2010.

[10] D. Garcia-Romero and C. Espy-Wilson, "Analysis of i-vector length normalization in speaker recognition systems." in *Interspeech*, 2011.

[11] A. Silnova, N. Brümmer, D. Garcia-Romero, D. Snyder, and L. Burget, "Fast Variational Bayes for Heavy-tailed PLDA Applied to i-vectors and x-vectors," in *Interspeech*, 2018.

[12] D. Snyder, D. Garcia-Romero, D. Povey, and S. Khudanpur, "Deep neural network embeddings for text-independent speaker verification," in *Interspeech*, 2017.

[13] D. Snyder, D. Garcia-Romero, G. Sell, D. Povey, and S. Khudanpur, "X-vectors: Robust DNN embeddings for speaker recognition," in *ICASSP*, 2018.

[14] O. Novotný, O. Plchot, P. Matejka, L. Mosner, and O. Glembek, "On the use of x-vectors for robust speaker recognition," in *Odyssey*, 2018.

[15] M. McLaren, D. Castán, M. K. Nandwana, L. Ferrer, and E. Yılmaz, "How to train your speaker embeddings extractor," in *Odyssey*, 2018.

[16] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," *arXiv preprint arXiv:1702.08734*, 2017.

[17] A. Nautsch, S. Isadskiy, J. Kolberg, M. Gomez-Barrero, and C. Busch, "Homomorphic encryption for speaker recognition: Protection of biometric templates and vendor model parameters," *Odyssey*, 2018.

[18] W. Cai, J. Chen, and M. Li, "Exploring the encoding layer and loss function in end-to-end speaker and language recognition system," in *Odyssey*, 2018.

[19] S. Novoselov, A. Shulipa, I. Kremnev, A. Kozlov, and V. Shchemelinin, "On deep speaker embeddings for text-independent speaker recognition," in *Odyssey*, 2018.

[20] A. Nagrani, J. S. Chung, and A. Zisserman, "Voxceleb: a large-scale speaker identification dataset," in *Interspeech*, 2017.

[21] J. S. Chung, A. Nagrani, and A. Zisserman, "Voxceleb2: Deep speaker recognition," in *Interspeech*, 2018.

[22] M. McLaren, L. Ferrer, D. Castan, and A. Lawson, "The 2016 speakers in the wild speaker recognition evaluation," in *Interspeech*, 2016.

[23] W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, and L. Song, "Sphereface: Deep hypersphere embedding for face recognition," in *CVPR*, 2017.

[24] F. Wang, J. Cheng, W. Liu, and H. Liu, "Additive margin softmax for face verification," *IEEE Signal Processing Letters*, vol. 25, 2018.

[25] J. Deng, J. Guo, and S. Zafeiriou, "Arcface: Additive angular margin loss for deep face recognition," in *CoRR*, vol. abs/1801.07698, 2018.

[26] W. Liu, Y. Wen, Z. Yu, and M. Yang, "Large-margin softmax loss for convolutional neural networks," in *ICML*, 2016.

[27] D. Povey, X. Zhang, and S. Khudanpur, "Parallel training of deep neural networks with natural gradient and parameter averaging," in *ICLR*, 2015.