Available online at www.sciencedirect.com

ScienceDirect

Computer Speech & Language 59 (2020) 22–35

www.elsevier.com/locate/csl

End-to-end DNN based text-independent speaker recognition for long and short utterances

Johan Rohdin*, Anna Silnova, Mireia Diez, Oldřich Plchot, Pavel Matějka, Lukáš Burget, Ondřej Glembek

Brno University of Technology, Speech@FIT and IT4I Center of Excellence, Božetěchova 2, Brno 61266, Czech Republic

Received 16 November 2018; received in revised form 2 April 2019; accepted 2 June 2019
Available online 8 June 2019

Abstract

Recently several end-to-end speaker verification systems based on deep neural networks (DNNs) have been proposed. These systems have been proven to be competitive for text-dependent tasks as well as for text-independent tasks with short utterances. However, for text-independent tasks with longer utterances, end-to-end systems are still outperformed by standard i-vector + PLDA systems. In this work, we present an end-to-end speaker verification system that is initialized to mimic an i-vector + PLDA baseline. The system is then further trained in an end-to-end manner but regularized so that it does not deviate too far from the initial system. In this way we mitigate overfitting which normally limits the performance of end-to-end systems. The proposed system outperforms the i-vector + PLDA baseline on both long and short duration utterances.

Keywords: Speaker verification; DNN; End-to-end; Text-independent; i-vector; PLDA

1. Introduction

In recent years, there have been many attempts to take advantage of Deep Neural Networks (DNNs) in speaker verification. This is motivated by large performance improvements brought by DNNs to many other pattern recognition tasks such as speech recognition (Dahl et al., 2012) and face recognition (Schroff et al., 2015).

The initial attempts of using DNNs in speaker recognition aimed at replacing or improving one of the components of an i-vector + PLDA system (feature extraction, calculation of sufficient statistics, i-vector extraction or PLDA) with a neural network. For example, by using DNN bottleneck features instead of conventional MFCC features (Lozano-Diez et al., 2016), DNN acoustic models instead of Gaussian mixture models for extraction of sufficient statistics (Lei et al., 2014), DNNs for either complementing PLDA (Novoselov et al., 2015; Bhattacharya et al., 2016) or replacing it (Ghahabi and Hernando, 2014).

* Corresponding author.
E-mail address: rohdin@fit.vutbr.cz (J. Rohdin).

https://doi.org/10.1016/j.csl.2019.06.002
0885-2308/

More ambitiously, several DNN based systems that take the frame level features of an utterance as input and directly produce an utterance level representation have been proposed over the last years (Variani et al., 2014; Heigold et al., 2016; Zhang et al., 2016; Snyder et al., 2016; Bhattacharya et al., 2017; Snyder et al., 2017a). The utterance level representation is usually referred to as an *embedding*. The embedding is obtained by means of a *pooling mechanism*, for example taking the mean, over the framewise outputs of one or more layers in the DNN (Variani et al., 2014), or by the use of a recurrent DNN (Heigold et al., 2016). In the initial work (Variani et al., 2014), pooling (mean) was applied only in testing phase. In training the objective was frame level classification of speaker identity. In order to do speaker verification, the embeddings (referred to as *d-vectors*) were extracted and subjected to cosine scoring.

Ideally the DNNs should however be trained directly for the speaker verification task, i.e., binary classification of two utterances as a *target* or a *non-target* trial (Heigold et al., 2016). Moreover, all parameters of the system should ideally be trained jointly. Such systems are known as *end-to-end* systems and have been proven competitive for text-dependent tasks (Heigold et al., 2016; Zhang et al., 2016) as well as text-independent tasks with short test utterances and an abundance of training data (Snyder et al., 2016). On text-independent tasks with longer utterances, direct end-to-end optimization of speaker verification systems has unfortunately proven to be difficult (Snyder et al., 2016). One reason for this could be overfitting on the training data. A second reason could be that end-to-end systems have typically been trained on short segments even when long segments are used in testing. This reduces the memory requirements in training and reduces the risk of overfitting but introduces a mismatch between the training and test conditions.

Due to the difficulties in building end-to-end systems for text-independent speaker verification on longer utterances, much of the recent research on DNN based speaker verification have focused on approaches similar to the d-vector approach. That is, using DNNs only for extracting embeddings and then use these embedding in a standard backend such as PLDA (Bhattacharya et al., 2017; Snyder et al., 2017a). In particular, the Kaldi (Povey et al., 2011b) recipe for producing embeddings (referred to as *x-vectors*) have shown excellent performance in many standard benchmarks (Snyder et al., 2017a; 2018). There are two important differences between the x-vector and d-vector recipe. First, the x-vector DNN is trained on short (2−4s) segments whereas the d-vector recipe, as mentioned above, is trained on frames. Second, the x-vector recipe uses, in addition to the mean, also the standard deviation for producing the utterance level representation.

In our recent work (Rohdin et al., 2018), we showed that it is possible benefit from end-to-end training if we constrain the system to not deviate too much from a standard i-vector + PLDA system. To this end, we developed an end-to-end speaker verification system that is initialized to mimic an i-vector + PLDA baseline. The system consists of a DNN module for extraction of sufficient statistics (**f2s**), an DNN module for extraction of i-vectors (**s2i**) and finally, a discriminative PLDA (DPLDA) model (Burget et al., 2011; Cumani et al., 2013) for producing scores. These three modules are first developed individually so that they mimic the corresponding part of the i-vector + PLDA baseline. After the modules have been trained individually they are combined and the system is further trained in an end-to-end manner on both long and short utterances. During the end-to-end training, we regularize the model parameters towards the initial parameters so that they do not deviate too far from them. In this way the system is prevented from becoming too different from the original i-vector + PLDA baseline which reduces the risk of overfitting.

This paper is an extension of our previous work (Rohdin et al., 2018). Here, we provide a more thorough motivation for why end-to-end training (with binary cross-entropy) is an appealing way to build speaker verification systems, despite the difficulties in building end-to-end systems for text-independent speaker verification, and despite the the excellent performance of the x-vector recipe. Further, we will provide detailed analysis of the design and performance of the individual blocks of our proposed end-to-end architecture aiming to aid future research on end-to-end speaker verification.

The remainder of the paper is organized as follows. In Section 2, we discuss the background and motivation for end-to-end training. We will argue that end-to-end training (using binary cross-entropy) is the most reasonable way to train a speaker verification system and moreover the system should ideally be trained on segments of the same durations as expected in the test data. Then, in Section 3, we present the proposed end-to-end architecture. In Section 4, we present experimental results. In Section 5, we analyze the different parts of the proposed system more in detail. Finally, in Section 6, we concluded the paper. The main contribution in this paper compared to our previous work (Rohdin et al., 2018) is the analysis and discussion in Sections 2 and 5.

2. Motivation for end-to-end training

In this section we aim to motivate the use of end-to-end training with binary cross-entropy as loss function. The term *end-to-end* training is, as far as we know, not precisely defined but it usually refers to the following

1. All the parameters of the system should be trained jointly.
2. The task in training and evaluation is the same.
3. The training objective should be the same as, or at least very similar to, the evaluation metric.

It is easy to motivate why joint training of all the model parameters should be beneficial. Consider for example an embedding extractor and a PLDA backend. A PLDA backend assumes certain properties of the distribution of the embeddings (Gaussianity, independence of speaker and channel effects etc.). If the embedding extractor and backend are trained individually somehow, the embedding extractor has no way to know that it should try to produce embeddings that fulfill the assumptions of the PLDA model even when it might have been easy for it to do so.

It is also fairly easy to motivate why the task in training[1] and testing should be the same. For example, in the case of speaker verification, if the durations of the training utterances are shorter than the durations of the test utterances, the covariance of the training and testing embeddings will be different, leading to suboptimal verification scores.

Why the training objective ideally should be the same as the evaluation metric can be explained in two steps. First, in training we minimize the average cost of the training trials according to some cost function, c. Under reasonable circumstances given by statistical learning theory (Vapnik, 1998), this procedure will lead to an expected cost of the (unseen) test trials, according to the same cost function c, that becomes lower the larger the training set is. In other words, the training will eventually find the parameters, θ that optimize c on test data. Second, the optimal parameters θ may depend on the choice of evaluation metric (cost function). For example, in speaker verification the most common evaluation metric is the *Detection Cost Function* (DCF). This evaluation metric only cares about whether the score from the speaker verification system is on the correct side of a threshold which depends on the parameters of the specific DCF. If we optimize the system to produce scores that are optimal for some specific threshold, they will not necessarily be optimal for another threshold.

Therefore, in general the only way that we can be sure to obtain the model parameters that optimize our desired cost function on the test set, is to use the same cost function in training. Of course, this is not a sufficient condition for finding the optimal model parameters. For example in addition, the training and test data should follow the same distribution, the training algorithm should not get stuck in a local minimum and the amount of training data needs to be large enough. In fact, it should be noted that for smaller amount of training data, it is possible that other training objectives could give better values of the parameters. For example, it has been shown that (if possible) training by generative maximum likelihood results in better model performance than cross-entropy for small amounts of training data (Ng and Jordan, 2002). As far as we know, no one have presented any arguments that other popular training objectives in speaker verification such as triplet loss (Schroff et al., 2015) would be better than cross-entropy for small amounts of training data though. In speaker verification, the cost function of interest is usually the detection cost function (DCF) at some specific operating point (combination of cost of false alarm, cost of false rejection and probability of a target trial). Cross-entropy is however an average of all possible operating points (Brümmer and du Preez, 2006) and may therefore still be suitable as training objective when a DCF is the cost function of interest.

3. Proposed end-to-end DNN architecture

In this section we describe the proposed end-to-end architecture. The system is depicted in Fig. 1. We devote one subsection to the *features to statistics* module, one subsection for the *statistics to i-vectors* module, one subsection to the *DPLDA* module, and finally one subsection for describing the combination of the three modules. In Section 5, we analyse these components more in detail. Note that as in most previous work on end-to-end speaker verification, e.g., Heigold et al. (2016) and Snyder et al. (2016), we do not include feature extraction in the end-to-end pipeline. This is mainly because the computational load of training on the frame-level is high in an end-to-end system. The

[1] By "training" we mean building the non-speaker specific parts of the system (UBM, i-vector extractor, PLDA, DNNs etc.), as opposed to "enrollment" by which we mean building speaker specific models for the speakers we want to recognize.

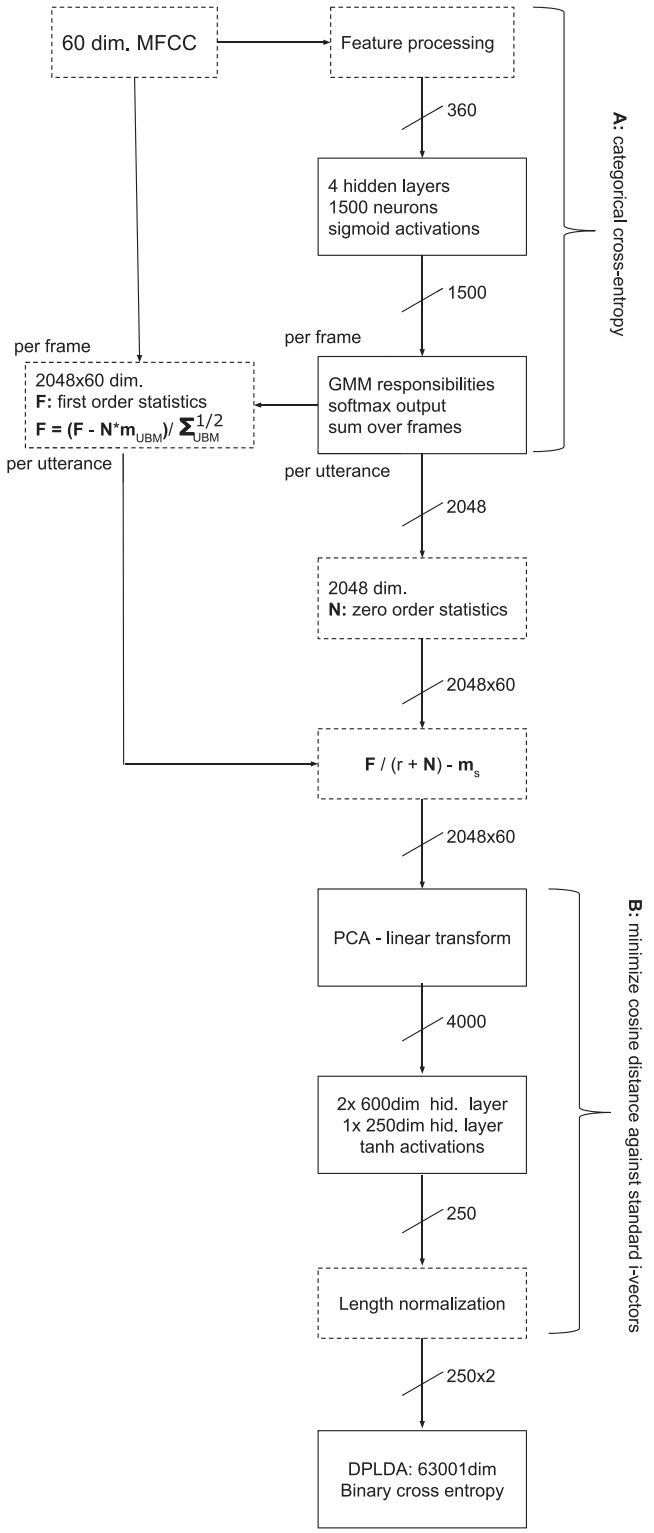

Fig. 1. Block diagram of the End-to-End system. Part **A** corresponds to the UBM that converts features to GMM responsibilities. By adding the next two blocks we obtain first order statistics (**f2s**). Part **B** (**s2i**) simulates the i-vector extraction followed by LDA and length normalization. Parameters in solid line blocks are meant to be trained, while outputs of the dashed blocks are directly computed.

problem of overfitting is most likely smaller at the frame level since the number of frames compared to trainable parameters is high so apart from computational issues including feature extraction in the end-to-end pipelines should be doable in principle.

3.1. Features to sufficient statistics

The first module of the end-to-end system translates a sequence of feature vectors into a vector of zeroth and first order statistics. We will denote this module as **f2s**. This module consists, of a network that predicts a vector of GMM responsibilities (posteriors) for each frame of the input utterance (Block A in Fig. 1), followed by a layer pooling the frames into sufficient statistics. The network that predicts responsibilities consists of four hidden layers with sigmoid activation functions and a softmax output layer. All hidden layers have 1500 neurons while the output layer has 2048 elements which corresponds to the number of components in our baseline GMM-UBM. We train this network with stochastic gradient descent (SGD) to optimize the categorical cross-entropy with the GMM-UBM posteriors as targets.

As input to the network, the acoustic features described in Section 4.2 are preprocessed as follows. For each frame, a window of 31 frames around the current frame (i.e., +/− 15 frames) is considered. In order to reduce the input dimension to the network as well as the amount of redundant information due to correlation between nearby frames, the temporal trajectory of each feature coefficient within the context window is weighted by a Hamming window and projected into first 6 DCT bases (including C_0). This results in a $6 \times 60 = 360$-dimensional input to the network for each frame. This preprocessing step has previously proven to be effective for acoustic modeling (Karafiát et al., 2014). It should be noted that expanding the features should in principle not be necessary in order to predict the GMM-UBM posteriors since these are calculated from original features. However, by using the expanded features, we hope that we can gain further improvements in the end-to-end training.

Once the network predicting responsibilities is trained, we add one more layer that outputs a vector of first order sufficient statistics for the whole utterance. The input to this layer is a matrix of frame-by-frame responsibilities coming from the previous softmax layer and a matrix of original acoustic features without any preprocessing. This layer is not trained but designed in such way that it exactly reproduces the standard calculation of sufficient statistics used in i-vector extraction. The reason for not using the expanded features for the first order statistics is that it would lead to a supervector with very large dimension which would be too computationally demanding to use as input to the next module.

3.2. Sufficient statistics to i-vectors

The second module of the end-to-end system is trained to mimic the i-vector extraction from the sufficient statistics (Block B in Fig. 1). We will denote this module as **s2i**. The input sufficient statistics are first normalized with the UBM mean and variance and then converted into MAP adapted supervectors (Reynolds et al., 2000),

$$F_i = \frac{\widetilde{F}_i}{r + N_i} \tag{1}$$

where $\widetilde{F}_i$ is the normalized first order statistics for Gaussian i in the UBM, N_i is the corresponding zeroth order statistics calculated by the **f2s** module, and r is a *relevance* factor which is set to 16 in this work. To overcome the computational problems that would arise when using 122,880 dimensional supervectors as input to the subsequent DNN, the supervectors were reduced by PCA (estimated on many utterances from the training data) into a 4000 dimensional space. The DNN consists of two 600 dimensional hidden layers, with hyperbolic tangent (tanh) activation functions. The last layer of the DNN is designed to produce length normalized 250 dimensional i-vectors. As training objective, we use the average cosine distance between DNN outputs and LDA reduced and length-normalized reference i-vectors. The DNN is trained with SGD and L1 regularization. Note that we do not update the UBM means and variances since they are just used for feature normalization. Nor do we update the PCA matrix because its many parameters would make this very computationally demanding as well as implicating a high risk of overfitting.

3.3. i-vectors to scores (DPLDA)

The final component of the end-to-end system should, given two i-vectors ϕ_i and ϕ_j, produce a log-likelihood ratio (LLR) score for the *same speaker* and *different speaker* hypotheses. For this, we use a discriminative PLDA (DPLDA) (Burget et al., 2011; Cumani et al., 2013) model. The DPLDA model is based on the fact that, given the two i-vectors, the LLR score for the (generative) PLDA model (Ioffe, 2006) is given by

$$s_{ij} = \boldsymbol{\phi}_i^T \Lambda \boldsymbol{\phi}_j + \boldsymbol{\phi}_j^T \Lambda \boldsymbol{\phi}_i + \boldsymbol{\phi}_i^T \Gamma \boldsymbol{\phi}_i + \boldsymbol{\phi}_j^T \Gamma \boldsymbol{\phi}_j + (\boldsymbol{\phi}_i + \boldsymbol{\phi}_j)^T \boldsymbol{c} + k, \tag{2}$$

where the parameters $\boldsymbol{\Lambda}, \boldsymbol{\Gamma}, \boldsymbol{c}$ and k can be calculated from the parameters of the PLDA model (see Burget et al., 2011 for details). The idea of DPLDA is to train $\boldsymbol{\Lambda}, \boldsymbol{\Gamma}, \boldsymbol{c}$ and k directly for the speaker verification task, i.e., given two i-vectors, tell whether they are from the same speaker or not. This is achieved by forming trials (usually all possible) from the training data and optimizing a loss function that encourages s_{ij} to be high for *same speaker* trials and low for *different speaker* trials. Typically, cross-entropy or the Support Vector Machine (SVM) objective is used. In this work we use cross-entropy, which since s_{ij} is an LLR, takes the form

$$\begin{aligned} xent = & \sum_{i,j:i\neq j,t_{ij}=1} \frac{P_{\text{tar}}}{N_1} \log\left(1 + \exp\left(-\left(s_{ij} + \log\frac{P_{\text{tar}}}{1-P_{\text{tar}}}\right)\right)\right) \\ & + \sum_{i,j:t_{ij}=-1} \frac{1-P_{\text{tar}}}{N_{-1}} \log\left(1 + \exp\left(s_{ij} + \log\frac{P_{\text{tar}}}{1-P_{\text{tar}}}\right)\right) \end{aligned} \tag{3}$$

where t_{ij} equals 1 for *same speaker* trials and –1 for *different speaker* trials, N_1 and N_{-1} is the number of *same speaker* and *different speaker* trials, respectively, and P_{tar} is the prior probability of a *same speaker* trial in our desired application.

Let the parameters $\boldsymbol{\Lambda}, \boldsymbol{\Gamma}, \boldsymbol{c}$ and k be collected in a vector θ. Let, $\widetilde{\boldsymbol{\theta}}$ be the parameter vector obtained from a generatively trained PLDA model. In our experiments, we initialize θ with $\widetilde{\boldsymbol{\theta}}$ and, during training, regularize it by adding

$$R(\boldsymbol{\theta}) = \rho \| \boldsymbol{\theta} - \widetilde{\boldsymbol{\theta}} \|^2 \tag{4}$$

(except for the parameter k) to the training objective.

Normally, DPLDA is trained iteratively using full batches, i.e., each update of the model parameters is calculated based on all training data. Whenever the DPLDA model is trained individually, we train it in this way. However, for an end-to-end system this would require too much memory and computational time. As is common for neural networks, we therefore calculate each update of the model parameters based on a minibatch, i.e., a randomly selected subset of the training data. Contrary to the individual training of **f2s** and **s2i**, we use the ADAM optimizer (Kingma and Ba, 2014) since it may be more robust to different learning rate requirements of the different modules compared to standard SGD. We half the learning rate whenever we see no improvement in C_{min}^{Prm} on the development set after an epoch (defined to be 200 batches).

Due to the fact that the training trials are formed by combining training utterances, it is not obvious how to optimally select the data for minibatches. In our experiments, we use the following procedure:

1. For each speaker, randomly group his/her utterances into pairs.[2]
2. For each minibatch, randomly select (without replacement) N pairs and use all trials that can be formed from the corresponding utterances. If the last pair is selected, repeat Step 1.

3.4. End-to-end system

After the individual components described in the previous subsections have been trained individually, they are combined to an end-to-end system. During end-to-end training we then regularize model parameter not deviate too far from those of the individually trained components, similarly to how we regularize the DPLDA model not to deviate too far from the generatively trained PLDA model in Eq. (4).

[2] If a speaker has only one utterance, this utterance will be used as a "pair". If a speaker has another uneven number of utterances, one of the pairs will be given three utterances.

Unfortunately, combining the modules as they are leads to large memory requirements of the end-to-end system. This happens mainly for two reasons. First, contrary to the individual training of the modules, the PCA projection now needs to be part of the network in order for the **f2s** and **s2i** modules to be connected. The memory on the GPUs used in the experiments is up to 8GB[3], so the PCA matrix with 122880×4000 parameters uses about 25% of the available memory. Second, the **f2s** now needs to process all frames from many different utterances in one batch to obtain the sufficient number of trials for the DPLDA module. If the three modules are combined as they are, we can use only approximately 2 utterances per minibatch which is not sufficient for effective training (see Section 5.3 for more discussion about this).

To mitigate the problem of the large PCA matrix we will, before doing the complete end-to-end training, train only the **s2i** DNN and the DPLDA model jointly. As for the individual training of **s2i**, we can use pre-calculated input that includes the PCA projection since this input is fixed as long as **f2s** is not updated. To mitigate the large memory requirements of the **f2s** module, we modify the training procedure to keep less intermediate results in memory. Specifically, in usual DNN training, the input is first *forward propagated* through the network to get the output of each layer. These outputs are stored in memory and used during *backpropagation* to obtain the derivative of the loss with respect to each model parameter. For the part of **f2s** that calculates responsibilities (Block A in Fig. 1), this results in $N(1500 + 1500 + 1500 + 1500 + 2048)$ variables to store in memory, where N is the total number of frames. This is much more than in subsequent modules (after pooling the frames into sufficient statistics, $\boldsymbol{F}$ and $\boldsymbol{N}$) because in this part of the network the layer outputs are per frame whereas in subsequent modules the layer outputs correspond to the whole utterance. Thus, in order to reduce the memory usage, we calculate the sufficient statistics for one utterance at the time and discard all the layer outputs from Block A once the sufficient statistics for the utterance have been calculated. When the sufficient statistics for all utterances have been obtained, we continue the forward propagation in the normal way, keeping all outputs in memory. During backpropagation, we recalculate the outputs when needed. This is achieved in a similar way as in scan_checkpoints[4]. This trick allows us to use minibatches of 75 pairs (N) instead of approximately 2.

4. Experiments

4.1. Datasets

We followed the design of the PRISM (Ferrer et al., 2012) dataset in the sense of splitting the data into **training** and test sets. The PRISM set contains data from the following sources: NIST SRE 2004–2010 (also known as MIXER collections), Fisher English and Switchboard. During training of the end-to-end system initialization, we used the female portion of the NIST SRE'10 telephone condition (condition 5) to independently tune the performance of the blocks A and B in Fig. 1.

We report results on three different datasets:

- The female part of the **PRISM language** condition[5] that is based on original (long) telephone recordings from NIST SRE 2005–2010. It contains trials from various languages, including the cross-language trials.
- The **short lang** condition (also containing only female trials) is derived from the PRISM language condition by taking multiple short cuts from each original recording. Durations of the speech in the cuts reflect the evaluation plan for NIST SRE'16 – more precisely we based our cuts on the actual detected speech in the SRE'16 labeled development data. We chose the cuts to follow the uniform distribution:
  - Enrollment between 25 and 50 s of speech
  - Test between 3 and 40 s of speech

  We split the resulting set into two equally large disjoint sets where speakers do not overlap. We used one part as our **dev** set for tuning the performance of the DPLDA and the end-to-end system. The other part was used for

[3] The brand of the GPU varies.
[4] http://www.deeplearning.net/software/theano/library/scan.html
[5] For detailed description, please see section B, paragraph 4 of Ferrer et al. (2012). We gave priority to evaluate the performance for a variety of utterance durations and languages instead of for the two genders because we did not expect the conclusions to differ for different genders.

evaluation only. It should be noted that, for simplicity, we test only on single-enrollment trials unlike in our SRE'16 system description where we include multi-enrollment trials (Plchot et al., 2017).

- Additionally, we report the results on the single-enrollment trials of the NIST SRE'16 evaluation set (both males and females).

4.2. Generative and discriminative baselines

As features we used 60-dimensional spectral features (20 MFCCs, including C_0, augmented with their Δ and $\Delta\Delta$ features). The features were short-term mean and variance normalized over a 3 s sliding window.

Both PLDA and DPLDA are based on i-vectors (Dehak et al., 2011) extracted by means of UBM with 2048 diagonal covariance components. Both UBM and i-vector extractor with 600 dimensions are trained on the **training** set. For training our generative (PLDA) and discriminative (DPLDA (Burget et al., 2011)) baseline systems, we used only telephone data from the **training** set and we also included short cuts derived from portion of our training data that come from non-English or non-native-English speakers. The duration of the speech in cuts follows the uniform distribution between 10 and 60 s. The cuts comprise of 22,766 segments out of total 85,858. Finally, we augmented the training data with labeled development data from NIST SRE'16.

PLDA: We used the standard PLDA recipe, when i-vectors are mean (mean is calculated using all training data) and length normalized. Then, the Linear Discriminant Analysis (LDA) is applied prior to PLDA training, decreasing dimensions of i-vectors from 600 to 250. We did not perform any additional domain adaptation or score normalization. We also filtered the training data in such a way that each speaker has at least six utterances which reduces it to the total of 62,994 training segments.

Discriminative PLDA: The DPLDA baseline model was trained on the full batch of i-vectors by means of the Limited-memory Broyden–Fletcher–Goldfarb–Shanno (LBFGS) algorithm, optimizing the binary cross-entropy on the training data. We used the **dev** set to tune the parameter ρ used for L2 regularization given by Eq. (4).

All i-vectors were mean (mean was calculated using all training data available) and length normalized. After the mean normalization, we performed LDA, decreasing the dimensionality of vectors to 250. As an initialization of DPLDA training, we used a corresponding PLDA model. During the DPLDA training, we set the prior probability of target trials to reflect the SRE'16 evaluation operating point (exactly in the middle between the two operating points of SRE'16 DCF (NIST, 2016)).

x-vector: It may also be worth comparing the end-to-end system to x-vectors (Snyder et al., 2017a). However, a systematic comparison of the x-vector recipe and the end-to-end system is unfortunately complicated. The standard x-vector recipe is implemented in the Kaldi toolkit (Povey et al., 2011a). Kaldi uses a special natural gradient-like optimizer as well as several heuristics to find good model parameters (Povey et al., 2015). For a meaningful comparison, we therefore need to exclude differences related to the optimization procedure. In Kaldi it is, as far as we know, not easy to implement the memory tricks discussed in Section 3.4 so we cannot run our end-to-end system in Kaldi. Therefore we implemented the x-vector recipe in Tensorflow. We followed the same data setup as in Novotný et al. (2018) (Embedding III, PLDA N), which like the experiments in this paper, is based on the PRISM dataset.

4.3. Experimental results

We report results in equal error rate (EER) as well as in the average minimum detection cost function for two operating points ($C_{\text{min}}^{\text{Prm}}$). The two operating points are the ones of interest in the NIST SRE'16 (NIST, 2016), namely the probability of target trials being equal to 0.01 and 0.005. Table 1 shows the results for the two baselines, the end-to-end system as well as systems where only some stages of the baseline have been replaced by a DNN. Row 1 and Row 2 show the results for the PLDA and DPLDA baseline, respectively. The DPLDA performs better than generatively trained PLDA on all sets. This is consistent with our previous findings on NIST SRE'16 (Plchot et al., 2017).

Row 3 shows the results when the UBM is replaced with the **f2s** DNN. The i-vector extractor and PLDA model are trained as in the baseline but on the output of the **f2s** DNN. It is noticeable that the **f2s** DNN performs better than the UBM which it is supposed to mimic. The reason for this seems to be that the **f2s** DNN is capable of learning a more robust model that generalizes better to the unseen data than the UBM, mainly because it uses a larger context. See Section 5.1 for details.

Table 1
Overall results, $C_{\text{min}}^{\text{Prm}}$ and EER. Modules marked with a '*' are trained jointly. Other modules are trained sequentially.

					SRE16		Short lang		PRISM lang	
	System name	stats	i-vector	PLDA	$C_{\text{min}}^{\text{Prm}}$	EER	$C_{\text{min}}^{\text{Prm}}$	EER	$C_{\text{min}}^{\text{Prm}}$	EER
0	x-vector	–	–	–	0.999	16.996	0.611	8.353	0.356	3.379
1	Baseline	UBM	i-extractor	Gen.	0.988	17.645	0.699	10.303	0.411	3.902
2	Baseline DPLDA	UBM	i-extractor	Discr.	0.975	16.902	0.616	9.462	0.360	3.461
3	f2s	DNN	i-extractor	Gen.	0.980	16.809	0.687	9.866	0.394	3.713
4	s2i	UBM	DNN	Gen.	0.988	16.686	0.788	11.141	0.430	4.584
5	f2s-s2i	DNN	DNN	Gen.	0.982	16.226	0.780	11.523	0.432	4.616
6	f2s-s2i-DPLDA	DNN	DNN	Discr	0.953	15.091	0.597	9.328	0.300	3.426
7	s2i-DPLDA_joint	DNN	DNN*	Discr.*	0.936	15.166	0.586	8.599	0.287	3.123
8	f2s-s2i-DPLDA_joint	DNN*	DNN*	Discr.*	0.936	15.170	0.587	8.661	0.287	3.125

Row 4 shows the performance when i-vector extractor is replaced by the **s2i** DNN. The input is the original statistics from the UBM and a PLDA model is trained on the output. We can see that, except for SRE'16, the performance degrades to some extent compared to the baseline (Row 1). Row 5 shows the results when we train a **s2i** module on the output from the **f2s** module instead of the statistics from the UBM. Again, we observe a small degradation compared to using a standard i-vector extractor (Row 3). Interestingly, when we further change from generative trained PLDA to DPLDA, the model performs better than both baselines. This suggests that the output from the **s2i** can well discriminate between speakers but may not well fulfill the PLDA model assumptions so that generative training does not work well.

After individual training of all blocks, we proceed with joint training of the **s2i** and DPLDA modules, using L2 regularization (tuned on the **dev** set) towards the parameters of the initial models. For this we use a batch size (N in Section 3.4) of 5000 pairs. As can be seen in the Row 7 of Table 1, the joint training of the two modules improves the performance on all data sets. We notice that in the joint training of the **s2i** and DPLDA modules, the regularization have very little impact on the DPLDA model. This is because it is already optimized for the output of the **s2i** module and at least initially, should not change at all.

Finally, the last row shows the performance when all modules are trained jointly. For this training, we can only use $N = 75$ as discussed in Section 3.3 but since this is quite slow, we use $N = 10$. As can be seen, the performance is almost unchanged from the previous row. There are three possible reasons for this. First, the minibatches might be too small for stable training. Second, with the **f2s** being well initialized and the subsequent modules already being trained to fit its output, the model may be stuck in a local minimum. Third, the **f2s** is in its current design quite constrained. It only estimates the responsibilities but cannot modify the features that are used to calculate the statistics. These issues will be studied in future work.

In summary, the final system achieved relative improvements with respect to the DPLDA baseline of 3.9%, 4.7% and 20.4% in $C_{\text{min}}^{\text{Prm}}$ on *SRE16, short lang* and *PRISM lang*, respectively. In EER, the relative improvements were 10.2%, 8.5%, and 9.7%.

Comparing with the x-vector system, we see that the end-to-end system is better than the x-vector in $C_{\text{min}}^{\text{Prm}}$. In terms of EER, the differences between the systems are smaller. The x-vector system is better on *short lang* and the end-to-end system is better on the other two sets. It is reasonable that we see a larger advantage for the end-to-end system in $C_{\text{min}}^{\text{Prm}}$ since this is more close to our training objective than what EER is. In relation to these results, it should be mentioned that for the standard Kaldi SRE16 x-vector recipe (Snyder et al., 2017b), the EER is 14.192% and $C_{\text{min}}^{\text{Prm}}$ is 0.991 for SRE16[6]. This recipe uses data in training that is included in the PRSIM set so we cannot evaluate it on *short lang* or *PRISM lang*. It should also be noted that recently, large improvements (mainly by means of more data augmentation) of the x-vector recipe have been presented (Snyder et al., 2018). However, results for the SRE16 single session trials without backend adaptation were not provided. Our end-to-end system is computationally more demanding than the x-vector recipe and would most likely have to be modified before it can take advantage of substantially larger amounts of training data.

[6] Note that this recipe normally uses the SRE16 *unlabeled* for centering the test data and for adapting the PLDA backend and we do not do that here. Also note that in this study we are considering only the single session trials.

5. Analysis and discussion

In this section, we analyze and describe the design considerations of the different blocks of the model more in detail. When designing the components, we either choose an architecture that we, based on experience, were confident would be good enough for our purpose (e.g. DPLDA for the "i-vector to scores" module) or did some initial experiments with individual component to find an architecture that seemed to be good enough for our purpose (e.g., the f2s and the s2i) module. We did not tune the architectures based on end-to-end training.

5.1. Architectures for f2s network

We did several experiments in order to choose optimal architecture for the **f2s** module. As discussed in Section 3.1, we trained only the part of the module that predicts a vector of responsibilities for each frame. The last layer of the whole **f2s** network is kept fixed for all of our experiments. Table 2 presents the results for some of architectures we tried for the trainable part of the **f2s** unit. In this subnetwork, we varied the number of hidden layers as well as experimented with adding contextual information to the input of the network. The results indicate that all of the tested architectures can successfully mimic the original UBM statistic extraction and that increasing the context window can even provide slight performance improvement compared to the baseline system. Using the 60 dimensional features as input to a 2 layer **f2s** DNN gave similar performance as the UBM ($C_{\text{min}}^{\text{Prm}}$ equal to 0.268 and 0.270, respectively, on SRE'10, condition 5) whereas the large context features gave substantial improvement ($C_{\text{min}}^{\text{Prm}}$ equal to 0.254). For the final end-to-end system we decided to use the most complex architecture out of all tried; it corresponds to the second line of the Table 2. Even though the **f2s** module does not seem to take any advantage from more complex architecture, we hoped that the end-to-end network could utilize the contextual information and complex dependencies in the first module to improve overall performance.

5.2. Architectures for s2i network

We carried out several experiments to find the optimal architecture for the **s2i** module. All experiments presented in this section use two 600 dimensional hidden layers, as this was found to provide optimal performance. The output layer has either 250 or 600 dimensions depending on whether the reference i-vectors have been reduced by LDA or not.

We first analyzed the effect of reducing the MAP adapted supervector to different dimensionalities. Dimensionalities from 4000 to 8000 provided comparable performance, whereas reducing the supervector dimension below 4000 harmed the performance, specially in terms of C_{min}^{Prm}. Therefore, we reduced the supervector dimensionality to 4000 in the remaining experiments.

Next, we explored different preprocessing techniques of the reference i-vectors in combination with different output layers of the module. As preprocessing techniques, we explored length-normalization and within-class-covariance normalization (WCCN). As output layers, we compared whether a length-normalization (LN) output layer was preferable to a linear output layer (LO). When performing these experiments, a linear output layer had to be used first for around 5 iterations and then switched to the LN layer in order to allow convergence. The results are presented in

Table 2
Architectures for **f2s**. Results are on NIST SRE'10, cond. 5, females, C_{min}^{Prm} and EER. The numbers in the *architecture* refers to input dimension, #layers × layer size and output dimension.

Architecture	EER	C_{min}^{Prm}
Baseline (no DNN)	2.37	0.270
DNN (360_4 × 1500_2048)	2.17	0.253
DNN (360_2 × 1500_2048)	2.20	0.254
DNN (60_2 × 1500_2048)	2.27	0.268

Table 3
Combinations of different reference i-vector preprocessing and output layers for **s2i** module. Results on NIST SRE'10, cond. 5, females, C_{min}^{Prm} and EER. LO → LN indicates that the model was initially trained with LO, then with LN.

ivec prep.	Output Layer	EER	C_{min}^{Prm}
Baseline (no DNN)	–	2.40	0.294
LN	LO	2.86	0.318
LN	LO → LN	2.82	0.302
WCCN + LN	LO	2.76	0.299
WCCN + LN	LO → LN	2.59	0.292

Table 3. We can see that using the LN output layer enhances performance. Further, we see that preprocessing the reference i-vectors with WCCN improves the performance. In the remaining experiments we use WCNN+LN for i-vector preprocessing and a LN output layer.

In the final set of experiments we analyze the system performance when using different training objectives: (1) minimizing the mean square error between reference i-vectors and the ones produced by the DNN. (2) Minimizing cosine distance between both sets of i-vectors. (3) Maximizing the PLDA scores obtained by scoring reference and produced vectors with a PLDA system trained on the reference i-vectors. We also experimented with reference i-vectors that were reduced by LDA (to 250 dimensions) before being length-normalized as well as with different regularization methods. The results for these experiments are shown in Table 4. It is clear that using LDA for preprocessing reference i-vectors is helpful. The differences between the different objective functions were marginal. Between regularization methods, L1 provided better overall gains. In the end-to-end system we decided to use the cosine distance objective with LDA and L1 regularization. For simplicity, we did not use the PLDA objective.

5.3. Minibatch design and sizes

As mentioned in Section 3.3, it is not obvious how to optimally select the data for minibatches. Contrary to typical DNN training scenarios, the training data in the speaker verification scenario are statistically dependent due to the fact that the training trials are formed by combining training utterances (Rohdin et al., 2016). Statistically dependent trials provide less information about the optimal model parameters than independent ones, i.e., they are less useful for training. In this aspect, minibatches should therefore ideally consist of independent trials. However, two more aspects need to be considered. First, selecting a set of trials to form a minibatch is not computationally efficient, instead we should select a set of utterances and use all possible trials. For example, instead of using (A−B) and (C−D) in a minibatch, where (X−Y) indicates a trial from the utterances X and Y, we should use all the trials (A−B), (A−C), (A−D), (B−C), (B−D), (C−D). This is because all the utterances A, B, C, and D need to be propagated through the **f2s** and **s2i** module anyway, and because the DPLDA model can very efficiently utilize all possible trials from a set of i-vectors (Cumani et al., 2013).

Table 4
Training objectives for **s2i** module. Results on NIST SRE'10, cond. 5, females, C_{min}^{Prm} and EER.

Obj. Function	LDA	REG	EER	C_{min}^{Prm}
Mean square error	No	–	2.59	0.292
Mean square error	Yes	–	2.57	0.283
Cosine distance	No	–	2.56	0.290
Cosine distance	Yes	–	2.55	0.284
Cosine distance	Yes	L1	2.43	0.281
Cosine distance	Yes	L2	2.50	0.284
PLDA	Yes	L1	2.36	0.284

Table 5
Effect of minibatch size. Results in C_{min}^{Prm} on development set and training lost of the final model on the training set, L.

Training method	Batch size	L2: 1.0		L2: 0.1	
		C_{min}^{Prm}	L	C_{min}^{Prm}	L
LBFGS	∼85k	0.554	0.203	0.566	0.128
ADAM	10	0.576	0.287	0.670	0.301
”.	20	0.558	0.282	0.558	0.258
”.	50	0.540	0.253	0.537	0.236
”.	100	0.546	0.244	0.549	0.206
”.	500	0.544	0.231	0.568	0.173
”.	1000	0.546	0.225	0.562	0.160
”.	2500	0.552	0.217	0.568	0.146
”.	5000	0.557	0.213	0.565	0.136

The second aspect to consider is that target and non-target trials should be reasonably balanced in the minibatch. The method we used in the experiments (described in Section 3.3) results in minibatches that have few target trials per minibatch. An approach that results in more target trials per minibatch is to let the minibatches consist of all utterances from a few number of speaker. Although this results in more target trials, these trials are statistically dependent since the same speaker and utterances are used in several of the trials. Our initial experiments suggest that this approach is worse than the one we used in this work.

In Table 5, we present an analysis on how the training is affected by the minibatch size. We consider two metrics. First, C_{min}^{Prm} on the **dev** set. Second, the training loss on the whole training set for the final model. The results are given for the case of L2 regularization 1.0 (the optimal) and 0.1. We can see that the minibatch size needs to be several thousand in order for the resulting L to be comparable to the L obtained by training with full batches (LBFGS). This holds especially for the weaker regularization. Fortunately, C_{min}^{Prm} (on the **dev** set) converges faster than L which suggest that even though we do not manage to optimize the model as well as with LBFGS, it is sufficient from the performance point of view. However, this may not be the case for more complicated models or data sets. In these experiments we checked L after each epoch (defined to be 20 batches, i.e., less than in the end-to-end experiments) and halved the learning rate if it did not improve. The reason we used L as halving criteria instead of C_{min}^{Prm} was that these experiments were intended to show how well minibatch training can optimize the model compared to using full batches.

6. Conclusions

We have developed an end-to-end speaker verification system that outperforms an i-vector+PLDA baseline on three different datasets having utterances from many different languages and of both long and short durations. The system was constrained to behave similar to an i-vector + PLDA system. In this way we mitigated overfitting which normally limits the performance of end-to-end systems. This was a conservative approach and it is possible that less constrained systems can perform better. However, our main motivation for this work was to show that it is possible to take advantage from end-to-end training in text-independent speaker recognition on both long and short utterances. In this paper we present a more detailed motivation and analysis of the system compared to our previous publications. We hope that this work can serve as motivation for future research on end-to-end training of text-independent speaker recognition systems. For the system presented in this study, we found that joint training of all its three modules is difficult due to large memory requirements. However, joint training of two modules was effective. In future work we therefore want to develop more effective strategies for joint training of all three modules or apply end-to-end training on lighter architectures. It should also be noted that the proposed system is designed single enrollment sessions, and extending it to deal with multiple enrollment sessions is an important future work.

Acknowledgments

The work was supported by European Union’s Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No. 748097, the Marie Sklodowska-Curie cofinanced by the South Moravian

Region under grant agreement No. 665860, Google Faculty Research Award program, Czech National Science Foundation (GACR) projects “NEUREM3” No. 19-26934X and No. GJ17-23870Y, Technology Agency of the Czech Republic project No. TJ01000208 “NOSICI”, by a contract with NTT Corporation and by Czech Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project “IT4 Innovations excellence in science - LQ1602”.

References

Bhattacharya, G., Alam, J., Kenny, P., 2017. Deep speaker embeddings for short-duration speaker verification. Interspeech 2017, pp. 1517–1521.

Bhattacharya, G., Alam, J., Kenny, P., Gupta, V., 2016. Modelling speaker and channel variability using deep neural networks for robust speaker verification. In: Proceedings of the 2016 IEEE Spoken Language Technology Workshop, SLT 2016, San Diego, CA, USA, December 13−16, pp. 192–198.

Brümmer, N., du Preez, J., 2006. Application-independent evaluation of speaker detection. Comput. Speech Lang. 20 (2−3), 230–275.

Burget, L., Plchot, O., Cumani, S., Glembek, O., Matějka, P., Brümmer, N., 2011. Discriminatively trained probabilistic linear discriminant analysis for speaker verification. In: Proceedings of the 2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 4832–4835. doi: 10.1109/ICASSP.2011.5947437.

Cumani, S., Brümmer, N., Burget, L., Laface, P., Plchot, O., Vasilakis, V., 2013. Pairwise discriminative speaker verification in the i−vector space. ieeetr_aslp 21 (6), 1217–1227. doi: 10.1109/ICASSP.2011.5947442.

Dahl, G.E., Yu, D., Deng, L., Acero, A., 2012. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. IEEE Trans. Audio Speech Lang. Process. 20 (1), 30–42. doi: 10.1109/TASL.2011.2134090.

Dehak, N., Kenny, P., Dehak, R., Dumouchel, P., Ouellet, P., 2011. Front-end factor analysis for speaker verification. IEEE Trans. Audio Speech Lang. Process. PP (99). doi: 10.1109/TASL.2010.2064307.

Ferrer, L., Bratt, H., Burget, L., Cernocky, H., Glembek, O., Graciarena, M., Lawson, A., Lei, Y., Matejka, P., Plchot, O., et al., 2012. Promoting robustness for speaker modeling in the community: the prism evaluation set. https://code.google.com/p/prism-set/.

Ghahabi, O., Hernando, J., 2014. Deep belief networks for i-vector based speaker recognition. In: Proceedings of the 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 1700–1704. doi: 10.1109/ICASSP.2014.6853888.

Heigold, G., Moreno, I., Bengio, S., Shazeer, N., 2016. End-to-end text-dependent speaker verification. In: Proceedings of the 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 5115–5119. doi: 10.1109/ICASSP.2016.7472652.

Ioffe, S., 2006. Probabilistic linear discriminant analysis. ECCV (4), pp. 531–542.

Karafiát, M., Grézl, F., Veselý, K., Hannemann, M., Szőke, I., Černocký, J., 2014. But 2014 babel system: analysis of adaptation in nn based systems. In: Proceedings of Interspeech 2014. International Speech Communication Association, pp. 3002–3006.

Kingma, D.P., Ba, J., 2015. Adam: a method for stochastic optimization. 3rd International Conference on Learning Representations, (ICLR) 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings http://arxiv.org/abs/1412.6980, https://dblp.org/rec/bib/journals/corr/KingmaB14 .

Lei, Y., Scheffer, N., Ferrer, L., McLaren, M., 2014. A novel scheme for speaker recognition using a phonetically-aware deep neural network. In: Proceedings of the 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 1695–1699. doi: 10.1109/ICASSP.2014.6853887.

Lozano-Diez, A., Silnova, A., Matějka, P., Glembek, O., Plchot, O., Pešán, J., Burget, L., Gonzalez-Rodriguez, J., 2016. Analysis and optimization of bottleneck features for speaker recognition. In: Proceedings of Odyssey 2016. International Speech Communication Association, pp. 352–357.

Ng, A., Jordan, M., 2002. On Discriminative vs. Generative classifiers: a comparison of logistic regression and naive Bayes. In: Dietterich, T., Becker, S., Ghahramani, Z. (Eds.), Advances in Neural Information Processing Systems (NIPS). MIT Press, pp. 841–848.

NIST, 2016. The 2016 NIST speaker recognition evaluation plan (sre16). https://www.nist.gov/file/325336.

Novoselov, S., Pekhovsky, T., Kudashev, O., Mendelev, V.S., Prudnikov, A., 2015. Non-linear PLDA for i-vector speaker verification. In: Proceedings of the 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 214–218.

Novotný, O., Plchot, O., Matejka, P., Mosner, L., Glembek, O., 2018. On the use of x-vectors for robust speaker recognition.

Plchot, O., Matějka, P., Silnova, A., Novotný, O., Diez, M., Rohdin, J., Glembek, O., Brümmer, N., Swart, A., Jorrín-Prieto, J., García, P., Buera, L., Kenny, P., Alam, J., Bhattacharya, G., 2017. Analysis and description of ABC submission to NIST SRE 2016. Interspeech 2017, pp. 1348–1351. Stockholm, Sweden.

Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., Hannemann, M., Motlicek, P., Qian, Y., Schwarz, P., Silovsky, J., Stemmer, G., Vesely, K., 2011. The kaldi speech recognition toolkit. In: Proceedings of the IEEE 2011 Workshop on Automatic Speech Recognition and Understanding. IEEE Signal Processing Society.IEEE Catalog No.: CFP11SRW-USB.

Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., Hannemann, M., Motlicek, P., Qian, Y., Schwarz, P., et al., 2011. The kaldi speech recognition toolkit. In: Proceedings of the IEEE 2011 workshop on automatic speech recognition and understanding. IEEE Signal Processing Society.

Povey, D., Zhang, X., Khudanpur, S., 2015. Parallel training of deep neural networks with natural gradient and parameter averaging. In: Proceedings of the 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7−9, 2015, Workshop Track.

Reynolds, D.A., Quatieri, T.F., Dunn, R.B., 2000. Speaker verification using adapted gaussian mixture models. Dig. Signal Process. 10, 19–41.

Rohdin, J., Biswas, S., Shinoda, K., 2016. Robust discriminative training against data insufficiency in PLDA-based speaker verification. Comput. Speech Lang. 35, 32–57.

Rohdin, J., Silnova, A., Diez, M., Plchot, O., Matejka, P., Burget, L., 2018. End-to-end DNN based speaker recognition inspired by i-vector and PLDA. In: Proceedings of the 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 4874–4878.

Schroff, F., Kalenichenko, D., Philbin, J., 2015. Facenet: a unified embedding for face recognition and clustering. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

Snyder, D., Garcia-Romero, D., Povey, D., Khudanpur, S., 2017. Deep neural network embeddings for text-independent speaker verification. Interspeech 2017.

Snyder, D., Garcia-Romero, D., Sell, G., Povey, D., Khudanpur, S., 2018. X-vectors: Robust dnn embeddings for speaker recognition.

Snyder, D., Ghahremani, P., Povey, D., Garcia-Romero, D., Carmiel, Y., Khudanpur, S., 2016. Deep neural network-based speaker embeddings for end-to-end speaker verification. In: Proceedings of the 2016 IEEE Spoken Language Technology Workshop (SLT), pp. 165–170. doi: 10.1109/SLT.2016.7846260.

Snyder, D., et al., 2017. Kaldi sre16 x-vector recipe. https://github.com/kaldi-asr/kaldi/tree/master/egs/sre16/v2. Accesed: 2017-11.

Vapnik, V.N., 1998. Statistical Learning Theory. Wiley-Interscience.

Variani, E., Lei, X., McDermott, E., Moreno, I.L., Gonzalez-Dominguez, J., 2014. Deep neural networks for small footprint text-dependent speaker verification. In: Proceedings of the 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 4052–4056. doi: 10.1109/ICASSP.2014.6854363.

Zhang, S.X., Chen, Z., Zhao, Y., Li, J., Gong, Y., 2016. End-to-end attention based text-dependent speaker verification. In: Proceedings of the 2016 IEEE Spoken Language Technology Workshop (SLT), pp. 171–178. doi: 10.1109/SLT.2016.7846261.