The Speaker and Language Recognition Workshop (Odyssey 2018)
26-29 June 2018, Les Sables d'Olonne, France

End-to-End versus Embedding Neural Networks for Language Recognition in Mismatched Conditions

Jesús Villalba[1], *Niko Brümmer*[2], *Najim Dehak*[1]

[1]Center for Language and Speech Processing, Johns Hopkins University, Baltimore, MD, USA
[2]Nuance Communications, Inc., South Africa

{jvillal7,ndehak3}@jhu.edu, niko.brummer@nuance.com

Abstract

Neural network architectures mapping variable-length speech utterances into fixed dimensional embeddings have started to outperform state-of-the-art i-vector systems in speaker and language recognition tasks. However, neural networks are prone to over-fit to the training domain and may be difficult to adapt to new domains with limited development data. A successful solution, used in recent NIST 2017 language recognition evaluation, consists of training the embedding extractor on out-of-domain data and applying a back-end classifier adapted to the target domain. In this paper, we compare the embedding+back-end approach with the end-to-end evaluation of the neural network to obtain language log-likelihoods. Doing careful adaptation of the networks, we show that end-to-end improved detection cost by 6% relative w.r.t. the best embedding system. We compared two embedding architectures. First, we evaluated embeddings using a temporal mean+stddev pooling layer to capture the long-term sequence information (a.k.a. x-vectors). Second, we present a novel probabilistic embedding framework where the embedding is a hidden variable. The network predicts a Gaussian posterior distribution for the embedding given each feature frame. Finally, the frame level posteriors can be combined in a principled way to obtain sequence level posteriors. In this manner, we obtain an uncertainty measure about the embedding value. Language scores are obtained integrating over the embedding posterior distribution. In our experiments, x-vectors outperformed probabilistic embeddings for embedding+back-end systems but both attained comparable results for end-to-end systems.

Index Terms: language recognition, x-vectors, embeddings, q-scoring, domain adaptation, end-to-end.

1. Introduction

Language recognition refers to the process of automatically detecting the language spoken in a speech utterance. Most successful approaches in the last decade have been based on mapping variable length audio sequences into a unique fixed-dimensional vector per recording. These vectors, usually referred as sequence embeddings, are then used as input to some classifier like logistic regression, SVM, or Gaussian classifier. One of the first embeddings were GMM super-vectors [1], later outperformed by GMM i-vectors [2].

With the deep learning revolution, deep neural networks (DNN) were incorporated into the i-vector extraction process. First, a DNN trained to predict senone posteriors for automatic speech recognition (ASR) was used to partition the feature space, instead of using an unsupervised GMM [3, 4]. Later, ASR DNN with a bottleneck layer was used to compute bottleneck features (BNF), which are then used as input to GMM i-vector systems [5, 6, 4]. Both approaches imply a significant improvement w.r.t. MFCC-SDC GMM i-vector systems. However, they are still based on the traditional GMM-UBM paradigm.

In the last years, there have been significant effort to find new approaches based on pure deep learning models. The works in [7, 8], propose an end-to-end network for frame level language classification. At evaluation time, language scores are averaged over time. More recently, the tendency consists in training end-to-end neural networks to make sequence level predictions [9]. This kind of neural networks contains a temporal pooling layer that summarizes the sequence information into a vector embedding. Then, the embedding can be used as feature for other classifiers. Embeddings in [10, 11], known as x-vectors, performed the best in the recent NIST language recognition evaluation (LRE17) [12]. The work in [13] presents a similar approach where embeddings are obtained by an attention mechanism.

In this paper, we propose a novel neural network architecture for sequence level language classification that contains a probabilistic embedding. In this framework, the embedding is a language discriminant latent variable. The neural network estimates a Gaussian posterior distribution for the embedding. Finally, language scores are computed integrating over the embedding posterior. We can use this network for end-to-end evaluation or to compute embeddings for other classifiers, e.g., a Gaussian back-end.

We compared x-vectors with our probabilistic embeddings in the recent NIST LRE17 task. In this task, there is significant mismatch between training set and development/evaluation sets. Different adaptation methods were investigated, i.e., back-end MAP adaptation and neural network adaptation.

The rest of the paper is organized as follows. Section 2 describes the state of the art sequence neural embeddings. Section 3 introduces our new proposed probabilistic embeddings. Section 4 describes the methods that we used to adapt the back-end classifiers and the neural embeddings from out-of-domain data to the target domain. Section 5 describes the experimental setup, including NIST LRE17 evaluation, data augmentation and system hyper-parameters. Section 6 presents the results of three kinds of experiments: out-of-domain embeddings with adapted back-end, end-to-end evaluation of adapted networks, and adapted embeddings plus back-end.

2. Deterministic embeddings

Recent works [14, 9] introduced a successful neural network architecture to map sequences into speaker discriminant fixed-

10.21437/Odyssey.2018-16

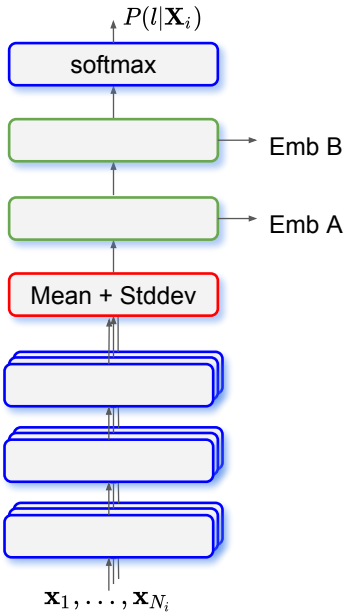

Figure 1: Neural network to compute x-vectors.

length vectors. Authors denominated these embeddings as x-vectors. Figure 1 depicts a generic x-vector neural network. The network receives a sequence of feature frames, which are processed by several layers. The result is summarized by a pooling layer that computes mean and standard deviation over time. Mean and standard deviation are concatenated together and propagated to the output through a series of feed-forward layers. The output is a dense layer with softmax activation predicting the class posteriors (speakers, language, etc.). The network is trained by minimizing a cross-entropy objective. For layers previous to the pooling, we can use whatever type of layer and activation function that we desire. However, time delay neural networks (TDNN) have proven to be very effective (a.k.a. 1D convolutions).

Sequence embeddings are extracted from the feed-forward layers after pooling. They are obtained from the affine transformation of each layer and before applying the non-linear activation function. In [9], the authors show that the embedding from the first affine transform after pooling (x-vector A) is the most discriminant. However, the embedding from the second layer (x-vector B) improves the results using fusion schemes.

The results in [15] indicate that x-vector can outperform i-vectors and be robust across datasets. However, x-vectors is a data greedy approach and only is able to beat i-vectors when we have large amount of training data. For this reason, we need to resort to data augmentation schemes–speed perturbation, noise, reverberation–, to make x-vectors work optimally. x-Vector embeddings have also been effective for language recognition, being the most successful approach in NIST LRE17 evaluation [10, 11, 16].

We used linear Gaussian classifier as back-end to compute language log-likelihoods from the embeddings. We chose this classifier over logistic regression or SVM because it allows easy Bayesian domain adaptation.

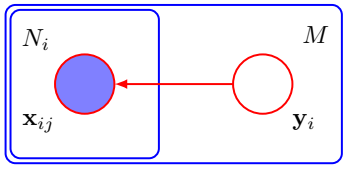

Figure 2: Graphical model for probabilistic embeddings. $\mathbf{y}_i$ is the language i embedding, All the feature frames of language i are independent given $\mathbf{y}_i$. We have M languages with N_i frames per language.

3. Probabilistic embeddings

3.1. Model definition

Let us consider the graphical model in Figure 2. The observed feature frames $\mathbf{x}_{ij}$ of language i are conditioned on the latent variable $\mathbf{y}_i$. The value of $\mathbf{y}_i$ is tied across all the frames of all the recordings of language i. Thus, $\mathbf{y}_i$ contains the language information. We assumed a standard normal prior for $\mathbf{y}_i$. We also assumed that conditional distribution $P(\mathbf{x}_{ij}|\mathbf{y}_i)$ is complex enough to model the feature frames, though its form is unknown to us. As we show below, we don't need to know the form of $P(\mathbf{x}_{ij}|\mathbf{y}_i)$. We just need to be able to compute a good estimation of the latent posterior $P(\mathbf{y}_i|\mathbf{x}_{i1}, \ldots, \mathbf{x}_{iN_i})$.

3.2. Likelihood ratio evaluation: Q-scoring

Let $\mathbf{X}_i = \{\mathbf{x}_{i1}, \ldots, \mathbf{x}_{iN_i}\}$ be the frames of a test recording; and let $\mathbf{X}_l$ be the set of all training frames for language l. We want to compute the likelihood ratio between the hypothesis that $\mathbf{X}_i$ is from language l and the hypothesis that $\mathbf{X}_i$ is not from language l. This is equivalent to the likelihood ratio between the hypothesis that $\mathbf{X}_i$ and $\mathbf{X}_l$ are from the same language, and the hypothesis that they are from different languages,

$$R(\mathbf{X}_i, l) = \frac{P(\mathbf{X}_i|l)}{P(\mathbf{X}_i|\neq l)} = \frac{P(\mathbf{X}_i, \mathbf{X}_l|\text{same})}{P(\mathbf{X}_i, \mathbf{X}_l|\text{diff})} . \quad (1)$$

We can write this likelihood ratio as a function of the posterior distributions of the latent factor $\mathbf{y}$ [17]

$$R(\mathbf{X}_i, l) = \int \frac{P(\mathbf{y}|\mathbf{X}_i)\, P(\mathbf{y}|\mathbf{X}_l)}{P(\mathbf{y})}\, \mathrm{d}\mathbf{y} . \quad (2)$$

We cannot compute the true posteriors, since we don't even know the form of the conditional likelihood. Instead, we propose to use a neural network to compute approximate posteriors $Q(\mathbf{y}|\mathbf{X}_i)$, $Q(\mathbf{y}|\mathbf{X}_l)$. For example, we assumed a Gaussian form for $Q(\mathbf{y}|\mathbf{X}_i) = \mathcal{N}(\mathbf{y}|\boldsymbol{\mu}(\mathbf{X}_i), \boldsymbol{\Sigma}(\mathbf{X}_i))$, where the mean and the covariance are obtained evaluating a neural network. For $Q(\mathbf{y}|\mathbf{X}_l)$, instead of passing all the frames of language l through the network, we assumed that posterior mean and covariance are trainable constants $P(\mathbf{y}|\mathbf{X}_l) \approx Q(\mathbf{y}|l) = \mathcal{N}(\mathbf{y}|\boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)$. We can make this assumption because the training languages are the same as the ones used for evaluation.

The likelihood ratio is thus approximated by

$$R(\mathbf{X}_i, l) = \int \frac{Q(\mathbf{y}|\mathbf{X}_i)\, Q(\mathbf{y}|l)}{P(\mathbf{y})}\, \mathrm{d}\mathbf{y} \quad (3)$$

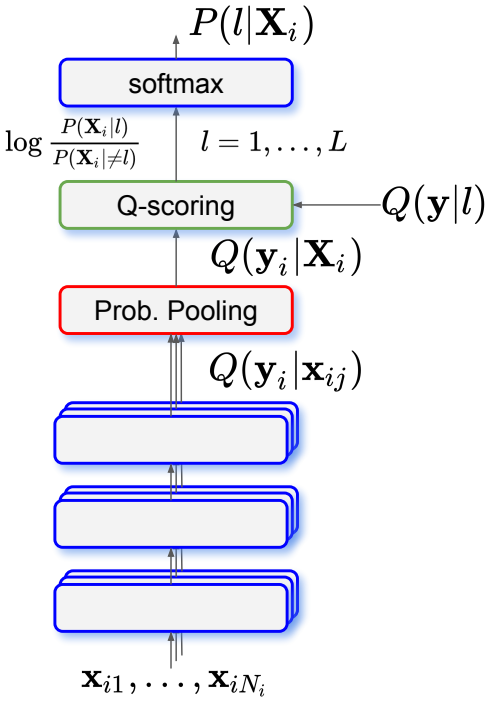

Figure 3: Neural network to compute probabilistic embeddings.

which we called Q-scoring. For Gaussian posteriors, this integral can be computed in closed form,

$$\begin{aligned}\log R(\mathbf{X}_i, l) \approx & \frac{1}{2}\Big(\log|\boldsymbol{\Sigma}_{i,l}| + \boldsymbol{\mu}_{i,l}^{\mathrm{T}}\boldsymbol{\Sigma}_{i,l}^{-1}\boldsymbol{\mu}_{i,l} \\ & - \log|\boldsymbol{\Sigma}(\mathbf{X}_i)| - \boldsymbol{\mu}(\mathbf{X}_i)^{\mathrm{T}}\boldsymbol{\Sigma}(\mathbf{X}_i)^{-1}\boldsymbol{\mu}(\mathbf{X}_i) \\ & - \log|\boldsymbol{\Sigma}_l| - \boldsymbol{\mu}_l^{\mathrm{T}}\boldsymbol{\Sigma}_l^{-1}\boldsymbol{\mu}_l\Big) \end{aligned} \tag{4}$$

where

$$\boldsymbol{\Sigma}_{i,l} = \left(\boldsymbol{\Sigma}(\mathbf{X}_i)^{-1} + \boldsymbol{\Sigma}_l^{-1} - \mathbf{I}\right)^{-1} \tag{5}$$

$$\boldsymbol{\mu}_{i,l} = \boldsymbol{\Sigma}_{i,l}\left[\boldsymbol{\Sigma}(\mathbf{X}_i)^{-1}\boldsymbol{\mu}(\mathbf{X}_i) + \boldsymbol{\Sigma}_l^{-1}\boldsymbol{\mu}_l\right] . \tag{6}$$

We applied a softmax function to the log-likelihood ratio to obtain the language posteriors $P(l|\mathbf{X}_i)$ for $l = 1, \ldots, L$. Thus, we can optimize $\boldsymbol{\mu}_l$, $\boldsymbol{\Sigma}_l$; and the parameters of the networks $\boldsymbol{\mu}(\mathbf{X}_i)$ and $\boldsymbol{\Sigma}(\mathbf{X}_i)$ by minimizing a cross-entropy objective.

To make computations feasible, we used diagonal covariances in this work.

3.3. Evaluation of the posterior

Now, we need to define the architecture for the neural network that computes $Q(\mathbf{y}_i|\mathbf{X}_i)$. Figure 3 depicts the proposed architecture, which is similar to the x-vector one. Each frame is processed by the same chain of layers and a pooling layer combines the information of all the frames to generate the mean and variance of $Q(\mathbf{y}_i|\mathbf{X}_i)$. For pooling, most works use averaging [18, 9] or max pooling [19]. Instead, we propose a pooling method that complies with the rules of probability.

We could prove that the posterior of $\mathbf{y}$ given N frames can be written as a function of the posteriors given each one of the frames $P(\mathbf{y}|\mathbf{x}_j)$ for $j = 1, \ldots, N$,

$$P(\mathbf{y}|\mathbf{x}_1, \ldots, \mathbf{x}_N) \propto \frac{\prod_{j=1}^{N} P(\mathbf{y}|\mathbf{x}_j)}{P(\mathbf{y})^{N-1}} . \tag{7}$$

Thus, we assumed that the layers before the pooling compute the mean and variance of the approximate individual posteriors

$$Q(\mathbf{y}_i|\mathbf{x}_{ij}) = \mathcal{N}(\mathbf{y}_i|\boldsymbol{\mu}(\mathbf{x}_{ij}), \boldsymbol{\Sigma}(\mathbf{x}_{ij})) , \tag{8}$$

and applied (7) to compute the pooled posterior. For the case of Gaussian posteriors and standard Normal priors, the pooled posterior is Gaussian with covariance and mean,

$$\boldsymbol{\Sigma}(\mathbf{X}_i) = \left(\mathbf{I} + \sum_{j=1}^{N_i}(\boldsymbol{\Sigma}(\mathbf{x}_{ij})^{-1} - \mathbf{I})\right)^{-1} \tag{9}$$

$$\boldsymbol{\mu}(\mathbf{X}_i) = \boldsymbol{\Sigma}(\mathbf{X}_i)\sum_{j=1}^{N_i}\boldsymbol{\Sigma}(\mathbf{x}_{ij})^{-1}\boldsymbol{\mu}(\mathbf{x}_{ij}) . \tag{10}$$

In our experiments, we estimated $Q(\mathbf{y}_i|\mathbf{x}_{ij})$ using a neural network with two outputs. The first output has linear activation and we assumed that it computes the mean $\boldsymbol{\mu}_{ij}$ or the first natural parameter of the Gaussian $\eta_{ij} = \boldsymbol{\Sigma}_{ij}^{-1}\boldsymbol{\mu}_{ij}$. The second output had sigmoid activation to assure that the predicted variance is lower than 1, since the posterior variance needs to be smaller than the prior variance.

We applied this architecture in two ways. First, to evaluate language scores in an end-to-end fashion. Second, we can use the mean of the latent posterior $Q(\mathbf{y}|\mathbf{X}_i)$ as embedding for a Gaussian classifier, in the same manner as we do with i-vectors. The former can benefit from the uncertainty of $\mathbf{y}$ (covariance), while in the latter that information is lost. We also tried a back-end with uncertainty propagation, but we didn't obtain any improvement. When we use the mean and variance of the embeddings, we will use the term *q-embedding*; while when we use just the mean, we will use the term *q-vector*.

4. Domain adaptation

4.1. Linear Gaussian back-end

We used a linear Gaussian classifier to compute the language log-likelihood scores from the embeddings. This back-end models each class with a Gaussian where the within-class covariance matrix is shared across languages. We equalized the weight of each language in the covariance estimation.

For domain adaptation, we computed the *a priori* back-end means and covariances on out-domain data and applied *Maximum a posteriori* (MAP) adaptation using in-domain data. The adaptation equations for the Gaussian classifier are

$$\boldsymbol{\mu}_l = \alpha_l\boldsymbol{\mu}_{\mathrm{ML}_l} + (1-\alpha_l)\boldsymbol{\mu}_{0_l} \qquad l = 1, \ldots, L \tag{11}$$

$$\begin{aligned}\mathbf{S}_{\mathbf{W}} = \frac{1}{L}\sum_{l=1}^{L}\big[&\beta_l\mathbf{S}_{\mathrm{ML}_l} + (1-\beta_l)\mathbf{S}_0 \\ &+\beta_l(1-\alpha_l)\left(\boldsymbol{\mu}_{\mathrm{ML}_l} - \boldsymbol{\mu}_{0_l}\right)\left(\boldsymbol{\mu}_{\mathrm{ML}_l} - \boldsymbol{\mu}_{0_l}\right)^{\mathrm{T}}\big]\end{aligned} \tag{12}$$

where

$$\alpha_l = \frac{N_l}{N_l + r_{\boldsymbol{\mu}}} \qquad \beta_l = \frac{N_l}{N_l + r_{\mathbf{W}}} ; \tag{13}$$

L is the number of languages, N_l is the number of samples of language l; $\boldsymbol{\mu}_{0_l}$ and $\mathbf{S}_0$ are the prior means and covariance; $\boldsymbol{\mu}_{\mathrm{ML}_l}$ and $\mathbf{S}_{\mathrm{ML}_l}$ are the maximum likelihood means and covariances for language l computed on the in-domain data; and $r_{\boldsymbol{\mu}}$ and $r_{\mathbf{W}}$ are the relevance factors.

4.2. Neural network

In the case of using the neural network for end-to-end evaluation of the language log-likelihoods, we were able to attain domain adaptation using a very simple technique. First, we train the neural network on large out-of-domain data. Afterwards, we adapted the network doing a few epochs where half of the audios of each mini-batch are from the target domain; and the other half come from the out-of-domain data. In each epoch, the network observed all the out-domain audios once; and the in-domain files were observed repeated times to meet the half/half criterion. In our experiments, we used 4 adaptation epochs; a learning rate $\times 5$ smaller than the training rate; and the batches consisted of 128 audios from each domain.

We can also use the adapted networks to extract adapted embeddings for the Gaussian classifier. The question arises whether the embeddings of the out-of-domain audios–which we use to train the prior back-end– should be obtained using the adapted or the non-adapted network. In our experiments, we observed that combining embeddings obtained from different networks doesn't work well. On the contrary, we obtained improvements using the adapted network on the in-domain and out-of-domain data.

5. Experimental Setup

5.1. NIST LRE17

We evaluated our approach on the NIST language recognition evaluation 2017 (LRE17) task [12]. The LRE17 task consists in closed set language identification between 14 languages from 5 language clusters (Arabic, English, Slavic, Iberian and Chinese).

We focused on the fixed condition where the datasets allowed for system development were constrained by the organizers. NIST provided a training set (TRN17) consisting of narrow-band telephony speech built from previous NIST evaluations (around 2000h). Switchboard and Fisher English telephony corpora were also allowed for training. Additionally, NIST provided a development set (DEV17) containing around 60 hours of speech from a domain similar to the evaluation set. Both, development and evaluation sets contain audio from two sources: narrow-band telephony and broadcast radio (MLS14); and wide-band video (VAST). MLS14 audio files consisted of segments of 3, 10 and 30 seconds while VAST audio files contained the full duration of the original source video file.

Language recognition systems were requested to provide a vector of calibrated log-likelihoods, one for each target language. Performance was measured using a detection cost function which is a weighted average of miss and false alarm rates.

$$C(\gamma) = \frac{1}{L}\sum_{i=0}^{L}\left[P_{\text{Miss}}(i,\gamma) + \frac{\gamma}{L-1}\sum_{j\neq i}P_{\text{FA}}(i,j,\gamma)\right] \tag{14}$$

where $\gamma = (1-P_{\mathcal{T}})/P_{\mathcal{T}}$, $P_{\mathcal{T}}$ is the target language prior, and L the number of languages. $P_{\text{Miss}}(i,\gamma)$ is the miss rate for language i and $P_{\text{FA}}(i,j,\gamma)$ is the probability of detecting language i in an audio containing language j. Miss and false alarms are computed by applying detection thresholds $\log(\gamma)$ to the language log-likelihood ratios (derived from the calibrated log-likelihoods). The primary metric averages (14) for two operating points, $P_{\mathcal{T}} = 0.5$ and $P_{\mathcal{T}} = 0.1$. Also, the counts of each corpus (MLS14 and VAST) are equalized when computing the cost function so both have the same weight in the metric.

5.2. Data augmentation

While the TRN17 and MLS14 datasets contain good quality speech, we observed that VAST audio contained significant number of artifacts, noise and reverberation. Because of this, we decided to augment TRN17 with noise and reverberation. The augmentation setup provided by MIT Lincoln Labs [16] was inspired by the recipe in [20]. Both real isotropic and point-source noises were used together to augment TRN17 $\times 4$. Point source noises were added at a rate of 12 occurrences per minute. Also, since we observed that the VAST data was very noisy and not very reverberant we used low SNR levels of 15, 10, 8, 6, 5, 2 and 0 and small room real room impulse responses.

5.3. Neural network architectures

Table 1 summarizes the network architecture for x-vectors and q-embeddings. The first three layers are common to both approaches. They are time delay layers with different temporal context. All of them used exponential linear unit (ELU) activation functions [21].

Afterwards for x-vectors, we used fully connected linear layers followed by temporal pooling. Then, an affine transform computed x-vector A; and ELU + affine transform computed x-vector B. Finally, a dense layer with softmax activation evaluated the language posteriors.

For q-embeddings; TDNN was followed by a dense layer with ELU activation. Then, an affine transform computes either the mean or the natural parameter of the frame level posterior; and another affine transform with sigmoid activation computes the variance. Finally, we applied pooling, scoring and softmax layers. Note that, to compensate for the fact that x-vector A is obtained from an affine transform after the pooling layer, q-embeddings included an extra dense layer before the pooling. In this way, both networks had approximately the same number of trainable parameters before the embedding layer.

5.4. System configuration

The input features for the embedding networks were bottleneck features (BNF) trained on 1800 hours of Fisher English. The bottleneck network was trained with Kaldi NNet2 [22]. It consisted of 7 hidden layers, the 6th layer was an 80 dimension linear bottleneck layer; the rest were TDNN layers with p-norm activations with input/output dimension equal to 3500/350. The output layer was a softmax that classifies 5577 senone acoustic units.

As reference, we show results with two i-vectors systems [23]. The first one trained GMM/i-vector/Back-end on TRN17 without augmentation, and the second one used augmentation in all the stages. They used BNF as input, 2048 component full covariance GMM and 600 dimensional i-vectors.

Embeddings were length normalized prior to be used on the Gaussian classifier.

The Gaussian back-end (GBE) was trained on TRN17 and MAP adapted to the DEV17 set. To obtain the scores for the DEV17 set we used 10-fold cross-validation. To compute the scores of the evaluation set, we used a model adapted to the full DEV17 set. We considered the case where the GBE is adapted to the pooled MLS14 and VAST data; and the case where we adapt a specific back-end for each domain. We calibrated the back-end log-likelihoods using multi-class linear logistic regression with language dependent offset and global scaling parameter [24]. The calibration was trained with 10-fold cross validation on DEV17. We considered condition independent

Table 1: *Embedding networks architectures*

Layer	Layer type	x-vector/q-embedding Size	Context
1	TDNN ELU	512	t-2:t+2
2	TDNN ELU	512	t-4,t-2,t,t+2,t+4
3	TDNN ELU	512	t-6,t-3,t,t+3,t+6

	x-vector		q-embed		
	Layer Type	Size	Layer Type	Size	
4	Dense Linear	400	Dense ELU	512	t
5			Dense Linear+Sigmoid	400+400	t
6	Mean+StdDev	400+400	Prob. Pooling	400+400	sequence
7	Dense Linear (embed A)	256	Q-scoring	14	sequence
8	ELU	256	softmax	14	sequence
9	Dense (embed B)	256			sequence
10	ELU	256			sequence
11	Dense softmax	14			sequence

and dependent calibrations.

For the embedding systems, neural networks were trained on the augmented TRN17 data using Keras toolkit [25]. We took 90% of the audio files for training and 10% for validation. We used Adam optimizer with a batch size of 96 sequences and drop-out rate 0.25. Learning rate was halved each time that the validation loss reached a plateau. For batch generation, in each epoch we sampled a random 5 second segment from each file. In this manner, we have a virtually infinite training set.

For neural embedding adaptation, we iterated for 4 epochs with a small learning rate (0.0001). The batch size was 256 sequences (128 from TRN17 and 128 from DEV17). In each epoch, we used each TRN17 files once and we reused the files from DEV17 as needed. In this case, we generated the batches by sampling random segments with random length between 3-10 seconds. To generate scores for the evaluation set we adapted the networks to the full DEV17. To generate scores for DEV17 we used 5-fold cross-validation. Thus, we needed 6 adapted networks to score the development and evaluation data.

6. Results

6.1. Neural network embeddings

First, we experimented using embeddings trained on TRN17 as feature for the adapted Gaussian classifier. Table 2 shows the performance in terms of detection cost for development and evaluation sets. It presents the cost for each data source (MLS14 and VAST) and the equalized average cost. We considered four situations depending on whether the back-end adaptation and calibration are condition dependent (CD) or independent (CI). Condition independent adaptation and calibration were trained in pooled MLS14 and VAST data. Meanwhile, condition dependent used specific models for each data source.

We compared several embedding types: i-Vectors trained on clean TRN17 only (I-Vec clean); i-Vectors trained on augmented TRN17 (I-Vec); x-vectors obtained from first layer after temporal pooling (X-Vec A); x-vectors obtained from the two layers after pooling (X-Vec A+B); q-vector where the network predicts the natural parameter of the $Q\left(\mathbf{y}|\mathbf{x}_{ij}\right)$ (Q-Vec nat); and q-vector where the network predicts the mean of $Q\left(\mathbf{y}|\mathbf{x}_{ij}\right)$ (Q-Vec mean). All x-vectors and q-vectors required augmented data to perform well.

Table 2: C_{avg} *Neural Embeddings with Gaussian classifier with condition dependent and independent (CD/CI) back-end adaptation and calibration.*

	Dev			Eval		
System	C_{Eq}	C_{MLS14}	C_{VAST}	C_{Eq}	C_{MLS14}	C_{VAST}
CI GBE + Cal.						
I-Vec clean	0.246	**0.158**	0.335	0.234	**0.165**	0.304
I-Vec	0.239	0.197	0.281	0.224	0.193	0.255
X-Vec A	0.236	0.216	0.255	0.223	0.207	0.239
X-Vec A+B	**0.207**	0.171	**0.243**	**0.204**	0.186	**0.221**
Q-Vec nat	0.253	0.200	0.307	0.237	0.189	0.286
Q-Vec mean	0.235	0.195	0.274	0.219	0.183	0.256
CI GBE + CD Cal.						
I-Vec clean	0.230	**0.154**	0.305	0.221	**0.162**	0.280
I-Vec	0.216	0.186	0.246	0.215	0.190	0.240
X-Vec A	0.220	0.203	0.238	0.214	0.204	0.224
X-Vec A+B	**0.200**	0.168	**0.233**	**0.194**	0.185	**0.203**
Q-Vec nat	0.212	0.185	0.239	0.209	0.183	0.236
Q-Vec mean	0.206	0.178	**0.233**	0.199	0.178	0.219
CD GBE + CI Cal.						
I-Vec clean	0.220	0.**156**	0.284	0.219	**0.168**	0.270
I-Vec	0.208	0.185	0.231	0.216	0.196	0.237
X-Vec A	0.222	0.201	0.243	0.216	0.212	0.219
X-Vec A+B	**0.191**	0.166	**0.215**	**0.196**	0.190	**0.202**
Q-Vec nat	0.229	0.189	0.269	0.219	0.187	0.251
Q-Vec mean	0.210	0.175	0.245	0.217	0.182	0.252
CD GBE + Cal.						
I-Vec clean	0.202	**0.150**	0.253	0.206	**0.164**	0.248
I-Vec	0.200	0.182	0.218	0.207	0.195	0.220
X-Vec A	0.202	0.195	0.209	0.197	0.206	0.187
X-Vec A+B	**0.183**	0.163	**0.203**	**0.186**	0.187	**0.185**
Q-Vec nat	0.197	0.175	0.219	0.203	0.185	0.221
Q-Vec mean	0.190	0.167	0.213	0.197	0.176	0.217

Comparing development and evaluation results, we observe that both are quite similar, so our k-fold strategy was successful avoiding over-fitting in the development set.

Overall, *X-Vec A+B* obtained the best C_{Eq} and C_{VAST}. *X-Vec A* and *Q-Vec mean* obtained the second-best C_{Eq}. However, *Q-Vec mean* was better on the clean MLS14 data and *X-Vec A* was better in the noisy VAST data. For MLS14, the clean i-vector system was still the best, followed by *Q-Vec mean*.

Regarding the domain adaptation analysis, both CD back-end adaptation and CD calibration improved i-vector and x-vectors results in a similar degree. For q-vectors, CD back-end did not improve much the cost, but CD calibration had a significant impact. Overall, we obtained the best performance by

Table 3: C_{avg} *End-to-end evaluation*

System	Dev			Eval		
	C_{Eq}	C_{MLS14}	C_{VAST}	C_{Eq}	C_{MLS14}	C_{VAST}
Embed + Adapt GBE						
X-Vec A	0.202	0.195	0.209	0.197	0.206	0.187
X-Vec A+B	**0.183**	**0.163**	**0.203**	**0.186**	0.187	**0.185**
Q-Vec mean	0.190	0.167	0.213	0.197	**0.176**	0.217
E2E						
X-Vec	0.235	**0.179**	0.291	0.242	0.195	**0.290**
Q-Emb nat	0.237	0.180	0.294	0.254	0.190	0.318
Q-Emb mean	**0.231**	**0.179**	**0.284**	**0.239**	**0.180**	0.298
E2E Adapt Scoring						
X-Vec	**0.203**	**0.167**	**0.238**	**0.215**	**0.183**	**0.247**
Q-Emb nat	0.235	0.174	0.296	0.249	0.194	0.303
Q-Emb mean	0.233	0.171	0.295	0.252	0.187	0.317
E2E Adapt Full Net						
X-Vec	0.177	0.147	0.207	0.177	0.149	0.206
Q-Emb nat	**0.172**	**0.140**	**0.204**	**0.175**	0.148	**0.202**
Q-Emb mean	0.174	0.140	0.207	0.176	**0.145**	0.206

combining CD back-end and calibration.

In conclusion, x-vectors performed better in noisy data and q-vectors in the clean data. We hypothesize that this happened because q-vector over-fitted more to the training data. This over-fitting may happen because q-vectors have only one layer between the embedding and the softmax output, while x-vectors have two dense layers. The possibility of concatenating embedding from different hidden layers is another advantage of x-vectors.

6.2. End-to-end evaluation

Table 3 shows results for end-to-end evaluation of the neural network to compute language log-likelihoods. These experiments used condition dependent calibration. We compare embedding+back-end, end-to-end evaluation without network adaptation, end-to-end adapting only the layers between the embedding and the output (scoring layers), and end-to-end adapting the full network.

The end-to-end network without adaptation performed poorly w.r.t. embeddings with adapted back-end, but the result is not catastrophic. Adapting the scoring layers, we obtained some improvement for the x-vector network but not in the Q-embeddings network. We think that this is because in the former we adapted two layers while in the latter we adapted just one layer. Finally, adapting the full network attained significant improvement outperforming the embedding+back-end results. Both, x-vector and q-embeddings networks attained similar performance. The best adapted network improved by 26% relative w.r.t the non-adapted network; by 11% w.r.t. to q-vector and *X-Vec A* embeddings; and 6% w.r.t. *X-Vec A+B* embeddings.

6.3. Adapted embeddings

In the last set of experiments, we used embeddings from full adapted networks as features for the Gaussian classifier. Both, Gaussian back-end and calibration were condition dependent. Table 4 presents the results. Both, x-vectors and q-vectors improved w.r.t. to the experiment with out-of-domain embeddings. *X-Vec A* improved by 7%, *X-Vec A+B* improved by 8%, *Q-Vec* improved by 6%. Adapted *X-Vec A* attains the same performance as out-of-domain *X-Vec A+B*. Adapted *X-Vec A+B* was slightly better than the best end-to-end system but, probably, is not a significant difference.

Table 4: C_{avg} *Adapted Neural Embeddings with Gaussian Back-end classifier*

System	Dev			Eval		
	C_{Eq}	C_{MLS14}	C_{VAST}	C_{Eq}	C_{MLS14}	C_{VAST}
Embed + Adapt GBE						
X-Vec A	0.202	0.195	0.209	0.197	0.206	0.187
X-Vec A+B	**0.183**	**0.163**	**0.203**	**0.186**	0.187	**0.185**
Q-Vec mean	0.190	0.167	0.213	0.197	**0.176**	0.217
E2E Adapt Full Net						
X-Vec	0.177	0.147	0.207	0.177	0.149	0.206
Q-Emb nat	**0.172**	**0.140**	**0.204**	**0.175**	**0.148**	**0.202**
Adapt Embed + GBE						
X-Vec A	0.189	0.175	**0.202**	0.183	0.185	0.182
X-Vec A+B	**0.176**	**0.149**	0.203	**0.170**	0.168	**0.171**
Q-Vec mean	0.195	0.153	0.236	0.184	**0.157**	0.210

7. Conclusion

We experimented with neural sequence embeddings for language recognition. We compared two neural architectures. First, we evaluated state-of-the-art network that applies a temporal mean + standard deviation pooling layer to capture the long-term sequence information (a.k.a. x-vectors). Second, we presented a novel probabilistic embedding framework where the embedding is a hidden variable. The network predicts a Gaussian posterior distribution for the hidden variable given each feature frame. Following, the frame level posteriors are combined in a principled way to obtain sequence level posteriors. In this manner, we obtain an uncertainty measure about the embedding value. Finally, we evaluate the language likelihood ratios by integrating over the embedding posterior distribution.

We evaluated our approach in the recent NIST LRE17 dataset, which includes strong domain mismatch between the training data and the dev/eval data. To approach domain adaptation, we compared three strategies. First, we used embeddings from a network trained on out-of-domain data and used them as input to a Gaussian back-end classifier MAP adapted to the development data. Second, we adapted the out-of-domain network to the dev data by doing a few extra epochs where mini-batches contain a mixture of out and in-domain data. Then, we evaluated the network in an end-to-end fashion to compute language scores. Finally, we combined adapted embeddings with adapted back-end.

In our experiments, x-vectors outperformed probabilistic embeddings for embedding+back-end systems. However, both attained comparable results for end-to-end evaluation. The best results were for systems using neural networks fully adapted to the target domain.

8. Acknowledgements

We want to thank the organizers and participants of the Johns Hopkins University HLTCOE SCALE 2017 Workshop for fruitful discussions and sharing of ideas. We also want to thank MIT Lincoln Labs for providing the data augmentation scripts.

9. References

[1] William M. Campbell, "A covariance kernel for svm language recognition," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2008*, Las Vegas, Nevada, USA, mar 2008, pp. 4141–4144, Ieee.

[2] Najim Dehak, Pedro A. Torres-Carrasquillo, Douglas Reynolds, and Reda Dehak, "Language recognition via

Ivectors and dimensionality reduction," in *Proceedings of the 12th Annual Conference of the International Speech Communication Association, Interspeech 2011*, Florence, Italy, aug 2011, pp. 857–860, ISCA.

[3] Yun Lei, Luciana Ferrer, Aaron Lawson, Mitchell Mclaren, and Nicolas Scheffer, "Application of Convolutional Neural Networks to Language Identification in Noisy Conditions," in *Proceedings of Odyssey 2014 - The Speaker and Language Recognition Workshop*, Joensuu, Finland, jun 2014, pp. 287–292, ISCA.

[4] Fred Richardson, Douglas Reynolds, and Najim Dehak, "Deep Neural Network Approaches to Speaker and Language Recognition," *IEEE Signal Processing Letters*, vol. 22, no. 10, pp. 1671–1675, oct 2015.

[5] Sriram Ganapathy, Kyu Han, Samuel Thomas, Mohamed Omar, Maarten Van Segbroeck, and Shrikanth S Narayanan, "Robust Language Identification Using Convolutional Neural Network Features," in *Proceedings of the 15th Annual Conference of the International Speech Communication Association, INTERSPEECH 2014*, Singapore, sep 2014, pp. 1846–1850, ISCA.

[6] Pavel Matejka, Le Zhang, Tim Ng, Sri Harish Mallidi, Ondrej Glembek, Jeff Ma, and Bing Zhang, "Neural Network Bottleneck Features for Language Identification," in *Proceedings of Odyssey 2014 - The Speaker and Language Recognition Workshop*, Joensuu, Finland, jun 2014, pp. 3–8, ISCA.

[7] Ignacio Lopez-Moreno, Javier Gonzalez-Dominguez, Oldrich Plchot, David Martínez, Joaquin Gonzalez-Rodriguez, and Pedro J Moreno, "Automatic language identification using deep neural networks," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2014*, Florence, Italy, may 2014, pp. 5337–5341, IEEE.

[8] Javier Gonzalez-Dominguez, Ignacio Lopez-Moreno, Joaquin Gonzalez-Rodriguez, and Pedro J Moreno, "Automatic Language Identification using Long Short-Term Memory Recurrent Neural Networks," in *Proceedings of the 15th Annual Conference of the International Speech Communication Association, INTERSPEECH 2014*, Singapore, sep 2014, pp. 2155–2159, ISCA.

[9] David Snyder, Daniel Garcia-Romero, Daniel Povey, and Sanjeev Khudanpur, "Deep Neural Network Embeddings for Text-Independent Speaker Verification," in *Proceedings of the 18th Annual Conference of the International Speech Communication Association, INTERSPEECH 2017*, Stockholm, Sweden, aug 2017, pp. 999–1003, ISCA.

[10] Alan Mccree, David Snyder, Gregory Sell, and Daniel Garcia-Romero, "Language Recognition for Telephone and Video Speech : The JHU HLTCOE Submission for NIST LRE17," in *submitted to Odyssey 2018*, Les Sables d'Olonne, France, jun 2018.

[11] David Snyder, Daniel Garcia-Romero, Alan Mccree, Gregory Sell, Daniel Povey, and Sanjeev Khudanpur, "Spoken Language Recognition using X-vectors," in *submitted to Odyssey 2018*, Les Sables d'Olonne, France, jun 2018.

[12] "NIST 2017 Language Recognition Evaluation Plan," Tech. Rep., NIST, 2017.

[13] Wang Geng, Wenfu Wang, Yuanyuan Zhao, Xinyuan Cai, and Bo Xu, "End-to-end Language Identification using Attention-based Recurrent Neural Networks," in *Proceedings of the 17th Annual Conference of the International Speech Communication Association, INTERSPEECH 2016*, San Francisco, California, USA, sep 2016, pp. 2944–2948, ISCA.

[14] David Snyder, Pegah Ghahremani, Daniel Povey, Daniel Garcia-Romero, Yishay Carmiel, and Sanjeev Khudanpur, "Deep neural network-based speaker embeddings for end-to-end speaker verification," in *Proceedings of the 2016 IEEE Spoken Language Technology Workshop (SLT)*, San Diego, CA, USA, dec 2016, pp. 165–170, IEEE.

[15] David Snyder, Daniel Garcia-Romero, Gregory Sell, Daniel Povey, and Sanjeev Khudanpur, "X-Vectors : Robust DNN Embeddings for Speaker Recognition," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 20182*, Alberta, Canada, apr 2018, IEEE.

[16] Fred Richardson, Pedro A Torres-carrasquillo, Jonas Borgstrom, Douglas Sturim, Youngjune Gwon, Jesús Villalba, Nanxin Chen, Jan Trmal, Nanxin Chen, and Najim Dehak, "The MIT Lincoln Laboratory / JHU / EPITA-LSE LRE17 System," in *submitted to Odyssey 2018*, Les Sables d'Olonne, France, jun 2018.

[17] Jesús Villalba, Niko Brummer, and Najim Dehak, "Tied Variational Autoencoder Backends for i-Vector Speaker Recognition," in *Proceedings of the 18th Annual Conference of the International Speech Communication Association, INTERSPEECH 2017*, Stockholm, Sweden, aug 2017, ISCA.

[18] Ehsan Variani, Xin Lei, Erik McDermott, Ignacio Lopez Moreno, and Javier Gonzalez-Dominguez, "Deep neural networks for small footprint text-dependent speaker verification," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Florence, Italy, may 2014, pp. 4052–4056, IEEE.

[19] Ishaan Gulrajani, Kundan Kumar, Faruk Ahmed, Adrien Ali Taiga, Francesco Visin, David Vazquez, and Aaron Courville, "PixelVAE: A Latent Variable Model for Natural Images," in *Proceedings of the International Conference of Learning Representations, ICLR 2017*, nov 2017.

[20] Tom Ko, Vijayaditya Peddinti, Daniel Povey, Michael L Seltzer, and Sanjeev Khudanpur, "A study on data augmentation of reverberant speech for robust speech recognition," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2017*, New Orleans, LA, USA, mar 2017, pp. 5220–5224, IEEE.

[21] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter, "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)," in *Proceedings of the International Conference on Learning Representations, ICLR 2016*, San Juan, Puerto Rico, may 2016.

[22] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, J. Silovsky, G. Stemmer, and K. Vesely, "The Kaldi speech recognition toolkit," in *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding, ASRU2011*, Waikoloa, HI, USA, dec 2011, pp. 1–4, IEEE.

[23] Najim Dehak, Patrick Kenny, Reda Dehak, Pierre Dumouchel, and Pierre Ouellet, "Front-End Factor Analysis For Speaker Verification," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 19, no. 4, pp. 788 – 798, may 2011.

[24] Niko Brummer, "Focal Multi-class," .

[25] Francois Chollet, "Keras," 2015.