8 Future Direction

Deep learning has been extensively used in speaker recognition. It will continue to play an important role in the foreseeable future, particularly in feature learning, end-to-end systems, and domain-invariant speaker recognition.

8.1 Time-Domain Feature Learning

So far, state-of-the-art speaker recognition systems use handcrafted features inspired by the human auditory systems. MFCC, PLP, and log-mel filterbank magnitude coefficients are some of the common handcrafted features. These features can be efficiently extracted from waveforms using signal processing methods. In speech recognition, it has been shown that deep neural networks can be trained to extract filter-bank like features directly from speech waveforms using a long context window [255–258]. This type of feature learning approach has the advantage that the feature extractor and the classifier can be jointly training, resulting in end-to-end systems.

Because speech signals are nonstationary waveforms with short-time stationarity, it is natural to apply a one-D convolutional neural network to process long-term waveform segments ($\approx$ 300ms), with a hop size of 10ms [259]. The waveform of an utterance is partitioned into into long-term segments (310ms in [260]), each of which is further partitioned into a short time frame. Denote $\{x_n\}_{n=1}^{N}$ as N consecutive samples in a long-term segment of an utterance. Then, the short-term segment centered at time point t within the long-term segment is

$$\mathbf{x}_t = \left[x_{t-(K_1-1)/2}, \ldots, x_t, \ldots, x_{t+(K_1-1)/2}\right]^{\mathsf{T}},$$

where K_1 is the kernel width (50 samples in [260]). In the first convolutional layer, $\mathbf{x}_t$ is convolved with F_1 FIR filters to give a matrix of size $F_1 \times K_1$. Then, temporal max-pooling is independently applied to the output sequences of the FIR filters. Figure 8.1 shows the operation of convolution and temporal max-pooling. In the remaining convolutional layers, the kernel width is in the unit of the number of time frames (rectangular blocks in Figure 8.1). For example, in [258], $K_2 = K_3 = 7$ and the frame shift (hopped distance) is 1.

Comparing Figure 8.1 and Figure 5.5 reveals that the CNN architecture is similar to the x-vector extractor. By changing the classes in Figure 8.1 to speaker labels and adding

https://doi.org/10.1017/9781108552332.009 Published online by Cambridge University Press

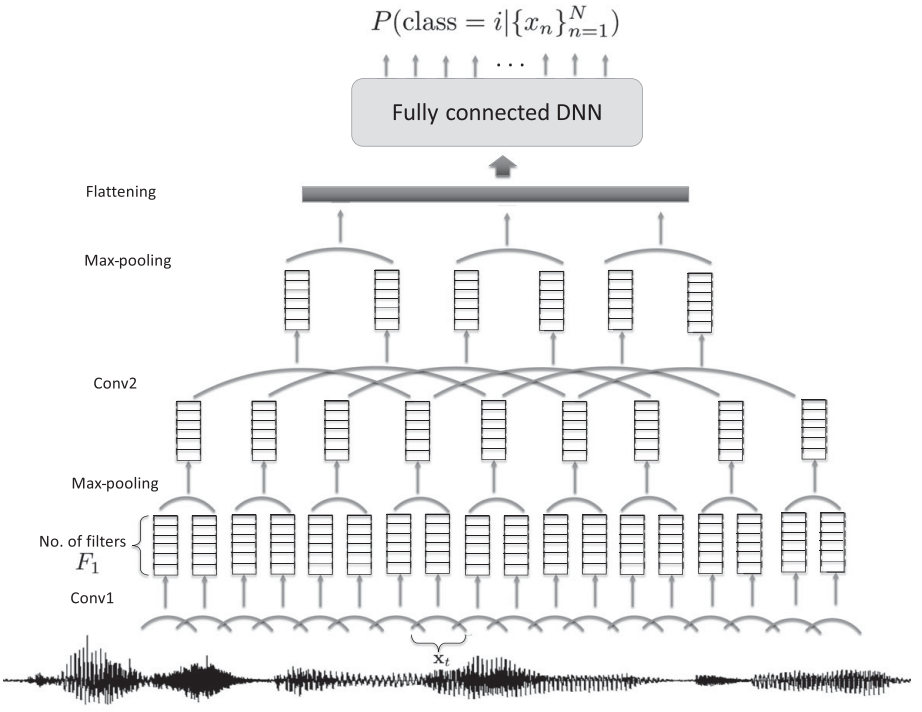

Figure 8.1 The convolution and max-pooling processes of a CNN for extracting features from speech waveforms. Nonlinearity is applied after max-pooling.

a statistical pooling layer (for pooling over time) and an embedded layer (for x-vector representation), it is possible to extract x-vectors from speech waveforms.

Another approach to extracting features from waveforms is to combine CNN, DNN, and LSTM [261]. Specifically, 25–35ms of time-domain waveform is fed to a few CNN layers that perform both time and frequency convolutions to reduce spectral variation. Then, the output of the last CNN layer is fed to a few LSTM layers. Finally, the output of the last LSTM layer is fed to a DNN. The process is repeated for every 10ms. The idea is based on the CLDNN in [262]. But unlike [262], the CLDNN in [261] receives raw waveforms as input. The advantage of using waveform as input is that the fine structure of the time-domain signals can be captured. It was shown that these networks have complementary modeling capabilities. Specifically, the CNN can reduce frequency variations, the LSTM can model long-term temporal variations, and the DNN can produce a higher-order representation that can be easily separated into distinct classes.

The first layer in the raw-waveform CLDNN [261] is a time-convolutional layer over the waveform. This layer can be considered as a finite impulse-response filterbank (such as a gammatone filterbank) followed by nonlinear operations. Therefore, the outputs of different convolutional nodes in the first layer correspond to different frequencies. By convolving M time points in a moving window with P convolutional filters with

https://doi.org/10.1017/9781108552332.009 Published online by Cambridge University Press

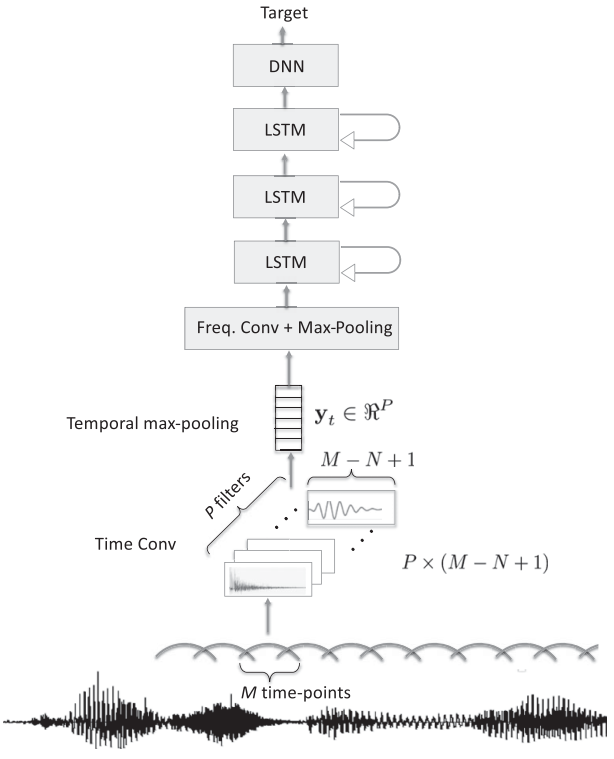

Figure 8.2 The architecture of CLDNN that accepts waveforms as input. The first time-convolutional layer has P filters (feature maps) of length N. For each frame, the first convolutional layer produces a vector $\mathbf{y}_t$, where t denotes the frame index. The elements in $\boldsymbol{y}_t$ are considered as frequency components, which are subject to frequency convolution and max-pooling. The resulting vectors are fed to the LSTM, followed by a DNN.

length N, the output of the first convolutional layer has dimension $P \times (M - N + 1)$ in time $\times$ frequency. These outputs are subjected to temporal pooling to produce $P \times 1$ outputs. Performing pooling in time has the effect of suppressing phrase information and making the feature representation in upper layers phase invariant. This convolutional and pooling process effectively convert time-domain signals into frame-level feature vectors of dimension P ($\mathbf{y}_t$ in Figure 8.2). The window is shifted by 10ms for each frame-level feature vector. Each $\mathbf{y}_t$ is subject to frequency convolution to reduce spectral variation.

In Figure 8.2, raw waveforms are processed by CNN followed by LSTM rather than directly processed by the LSTM. The advantage of the former is that each step in the LSTM corresponds to one frame (typically 10 ms), whereas in the latter, each time step corresponds to one time point (typically 0.125 ms). To model variations across 300ms, using the LSTM alone requires unrolling 2,400 time steps, which is not feasible in practice. On the other hand, using the CNN+LSTM to model the same time span, we only need to unroll 30 time steps.

To apply the architectures in Figures 8.1 and 8.2 for speaker recognition, we may extract embedded feature vectors from the outputs of the convolutional filters.

https://doi.org/10.1017/9781108552332.009 Published online by Cambridge University Press

As demonstrated in [259, 260], the first convolutional layer acts like a filter bank. Therefore, it is reasonable to replace the MFCCs by the responses of the convolutional filter.

8.2 Speaker Embedding from End-to-End Systems

Recent advances in deep learning has led to the use of deep neural networks for speaker feature extraction. The main idea is to replace the commonly used i-vectors by creating an embedded layer in a deep neural network and train the network in an end-to-end manner. Speaker features are then extracted from the embedded layer.

In [263, 264], the authors proposed to use the Inception-Resnet-v1 architecture that is commonly used in the computer vision community for such purpose. Specifically, short segments of a spectrogram are presented to the input of an inception network, as shown in Figures 8.3 and 8.4. At the embedding layer, utterance-level features are obtained by averaging the activations across all of the short segments in the utterance. The network is optimized by minimizing the triplet loss, as shown in Figure 8.3, using a Euclidean distance metric. Because of the averaging operation, the feature dimension at the embedding layer is fixed regardless of the length of the utterance. A similar strategy has also been used in the VGGVox network that creates the Voxceleb2 dataset [265] and the use of residual blocks in [266, 267]. Another example is the small footprint speaker embedding in [268] in which knowledge distillation was used to reduce the performance gap between large and small models.

To create fix-length embedding in the inception network, [263, 264] spatial pyramid pooling (SPP) [269] was applied to a convolutional layer of the inception network. Instead of using a sliding window as in conventional CNNs, SPP has bin size propor-

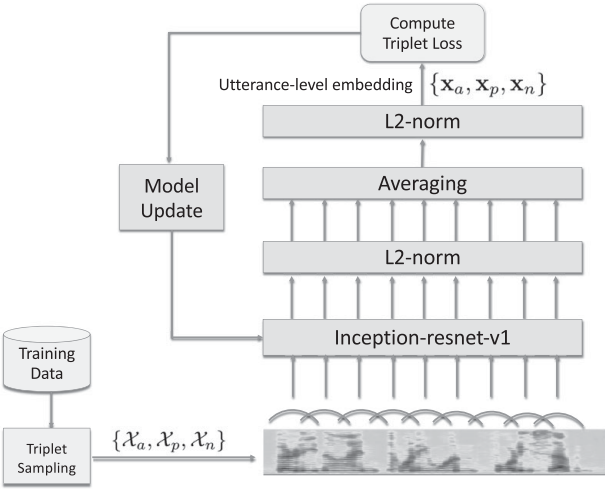

Figure 8.3 Training an inception-resnet-v1 to create an utterance-level embedded vector.

https://doi.org/10.1017/9781108552332.009 Published online by Cambridge University Press

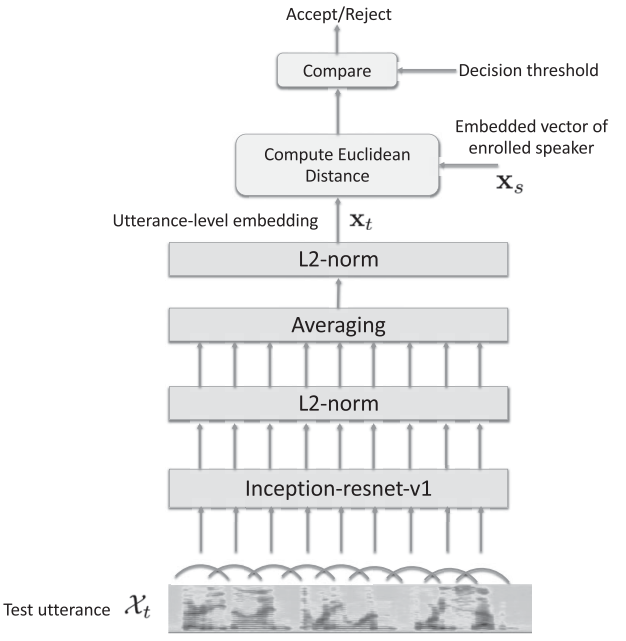

Figure 8.4 Given a test utterance, the inception network gives an utterance-level embedded vector, whose distance to the embedded vector of the enrolled speaker is compared with a decision threshold for decision making.

tional to the input feature size. As a result, the number of bins is fixed regardless of the length of the spectrogram. The embedded feature vectors can then be scored by either cosine-distance scoring or PLDA scoring.

8.3 VAE–GAN for Domain Adaptation

Adversarial learning [270] has been applied to create a domain-invariant space [266, 271–274], to reduce the language mismatch [275], channel mismatch [276], noise-type mismatch [277], acoustic condition mismatch [278], and to map unlabeled speech from one domain to another domain [279]. For instance, in [207], an encoder and a discriminator network are adversarially trained to produce bottleneck features that are robust against noise. Wang et al. [211] applied domain adversarial training (DAT) [272] to generate speaker discriminative and domain-invariant feature representations by incorporating a speaker classifier into an adversarial autoencoder, which outperforms traditional DA approaches on the 2013 domain adaptation challenge. In [279], cycle-GANs are used to map microphone speech in the SITW corpus to telephone speech and the generated telephone speech are mapped back to microphone speech. The inputs to the cycle-GANs are 11 frames of 40-dimensional filter bank features. The microphone-to-telephone features were passed to an x-vector extractor trained by using telephone speech.

https://doi.org/10.1017/9781108552332.009 Published online by Cambridge University Press

Although adversarial learning based unsupervised domain adaptation [207, 211] has greatly boosted the performance of speaker verification systems under domain mismatch conditions, the adversarial training may lead to non-Gaussian latent vectors, which do not meet the Gaussianity requirement of the PLDA back end. One possible way to make the x-vectors more Gaussian is to apply Gaussian constraint on the embedding layer of the x-vector extractor during training [280]. The problem can also be overcome by using variational autoencoders [237]. One desirable property of a VAE is that its KL-divergence term can be considered as a regularizer that constrains the encoder to produce latent vectors that follow a desired distribution. This property can be leveraged to cause the encoder to produce Gasussian latent vectors, which will be amenable to PLDA modeling.

8.3.1 Variational Domain Adversarial Neural Network (VDANN)

By applying domain adversarial training (DAT), the domain adversarial neural networks (DANN) in [211, 272] can create a latent space from which domain-invariant speaker discriminative features can be obtained. It is possible to incorporate variational regularization into the DAT to make the embedded features follow a Gaussian distribution so that the Gaussian PLDA back end can be applied. The resulting network is referred to as variational domain adversarial neural network (VDANN) [281].

Denote $\mathcal{X} = \{\mathcal{X}^{(r)}\}_{r=1}^{R}$ as a training set containing samples from R domains, where $\mathcal{X}^{(r)} = \{\mathbf{x}_1^{(r)}, \ldots, \mathbf{x}_{N_r}^{(r)}\}$ contains N_r samples from domain r. Also, denote $\mathbf{y}$ and $\mathbf{d}$ as the speaker and domain labels in one-hot format, respectively. Variational autoencoders (VAE) were originally proposed to solve the variational approximate inference problem (see Section 4.5). A VAE is trained by maximizing the evidence lower bound (ELBO) [237]:

$$L_{\text{ELBO}}(\theta, \phi) = \sum_{r=1}^{R} \sum_{i=1}^{N_r} \Big\{ -\mathcal{D}_{\text{KL}} \left(q_\phi(\mathbf{z}|\mathbf{x}_i^{(r)}) \| \, p_\theta(\mathbf{z}) \right) + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}_i^{(r)})} \left[\log p_\theta \left(\mathbf{x}_i^{(r)}|\mathbf{z} \right) \right] \Big\}, \tag{8.1}$$

where $q_\phi(\mathbf{z}|\mathbf{x})$ is an approximate posterior to the intractable true posterior $p_\theta(\mathbf{z}|\mathbf{x})$ and ϕ and θ are parameters of the encoder and decoder, respectively. The network parameterized by ϕ represents a recognition model that encodes input samples into the latent space and the network parameterized by θ represents a generative model that transforms the latent representations back to the original data space.

A VAE has a desirable property in that the first term in Eq. 8.1 can constrain the variational posterior $q_\phi(\mathbf{z}|\mathbf{x})$ to be similar to the desired prior $p_\theta(\mathbf{z})$. As a result, if $p_\theta(\mathbf{z})$ is a multivariate Gaussian distribution, the encoder will make the latent vectors to follow a Gasussian distribution, which is amenable to PLDA modeling.

Suppose $p_\theta(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$. Also assume that the true posterior follows a Gaussian distribution with an approximate diagonal covariance matrix. Then, the approximate posterior becomes

https://doi.org/10.1017/9781108552332.009 Published online by Cambridge University Press

$$\log q_\phi(\mathbf{z}|\mathbf{x}_i) = \log \mathcal{N}\left(\mathbf{z}; \boldsymbol{\mu}_i, \boldsymbol{\sigma}_i^2 \mathbf{I}\right), \tag{8.2}$$

where the mean $\boldsymbol{\mu}_i$ and standard derivation $\boldsymbol{\sigma}_i$ are outputs of the encoder given input $\mathbf{x}_i$, and they are parameterized by ϕ. Using the reparameterization trick (see Section 6.5.1), we obtain the lth latent sample $\mathbf{z}_{il} = \boldsymbol{\mu}_i + \boldsymbol{\sigma}_i \odot \boldsymbol{\epsilon}_l$, where $\boldsymbol{\epsilon}_l \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\odot$ is the Hadamard product. Substitute these terms into Eq. 8.1, we have the Gaussian VAE loss [237]:

$$\begin{aligned}\mathcal{L}_{\text{VAE}}(\theta, \phi) \simeq -\sum_{r=1}^{R}\sum_{i=1}^{N_r}\Bigg\{\frac{1}{2}\sum_{j=1}^{J}\Bigg[1 + \log\left(\sigma_{ij}^{(r)}\right)^2 \\ - \left(\mu_{ij}^{(r)}\right)^2 - \left(\sigma_{ij}^{(r)}\right)^2\Bigg] + \frac{1}{L}\sum_{l=1}^{L}\log p_\theta\left(\mathbf{x}_i^{(r)}|\mathbf{z}_{il}\right)\Bigg\},\end{aligned} \tag{8.3}$$

where J is the dimension of $\mathbf{z}$ and L denotes the number of latent samples. In practice, we set $L = 1$.

Figure 8.5 shows the structure of the VDANN. It comprises a speaker predictor C, a domain classifier D and a VAE, where the VAE comprises an encoder E and a decoder G. The parameters of C, D, E, and G are denoted as θ_c, θ_d, ϕ_e, and θ_g, respectively. Through adversarial training, a VDANN can form a domain-invariant space based on training data from multiple domains. More precisely, by applying adversarial training

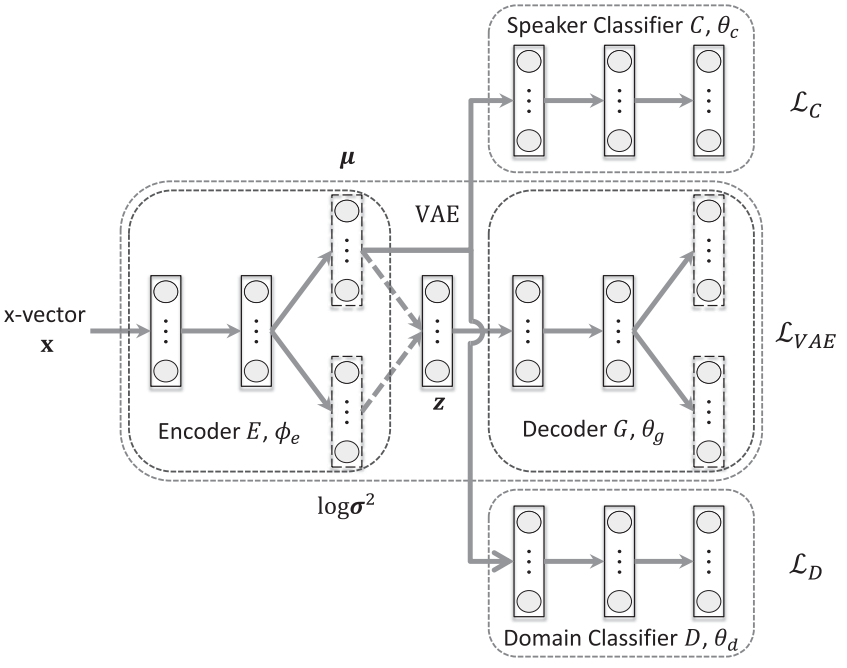

Figure 8.5 Structure of a VDANN. The dashed and solid arrows represent stochastic sampling and network connections, respectively.

https://doi.org/10.1017/9781108552332.009 Published online by Cambridge University Press

on E while fixing θ_d and minimizing the cross-entropy loss of C with respect to ϕ_e will make E to produce a domain-invariant but speaker discriminative representation.

The VDANN is trained by minimizing the following loss function:

$$\mathcal{L}_{\text{VDANN}}(\theta_c, \theta_d, \phi_e, \theta_g) = \mathcal{L}_C(\theta_c, \phi_e) - \alpha \mathcal{L}_D(\theta_d, \phi_e) + \beta \mathcal{L}_{\text{VAE}}(\phi_e, \theta_g), \tag{8.4}$$

where

$$\mathcal{L}_C(\theta_c, \phi_e) = \sum_{r=1}^{R} \mathbb{E}_{p_{\text{data}}(\mathbf{x}^{(r)})} \left\{ -\sum_{k=1}^{K} y_k^{(r)} \log C\left(E\left(\mathbf{x}^{(r)}\right)\right)_k \right\}, \tag{8.5}$$

$$\mathcal{L}_D(\theta_d, \phi_e) = \sum_{r=1}^{R} \mathbb{E}_{p_{\text{data}}(\mathbf{x}^{(r)})} \left\{ -\log D\left(E\left(\mathbf{x}^{(r)}\right)\right)_r \right\}, \tag{8.6}$$

and $\mathcal{L}_{\text{VAE}}$ has a form similar to Eq. 8.3. The only exception is that the parameters of the encoder and decoder are changed to ϕ_e and θ_g, respectively. The subscript k in the categorical cross-entropy loss of the speaker classifier C in Eq. 8.5 represents the kth output of the classifier. α and β are hyperparameters controling the contribution of different losses that shape the features produced by E.

For each minibatch in the training process, the weights in D are optimized by minimizing the domain classification loss. Then, its weights are frozen when training the remaining part of the VDANN. To incorporate speaker information into E, the speaker cross-entropy loss as computed at the output of C is minimized. Meanwhile, the domain classification loss is maximized to ensure that E can create a domain-invariant space. The VAE loss is also minimized to make the embedded features to follow a Gaussian distribution. In summary, the training process of VDANN can be expressed by the following minimax operation:

$$\min_{\theta_c, \phi_e, \theta_g} \max_{\theta_d} \mathcal{L}_{\text{VDANN}}(\theta_c, \theta_d, \phi_e, \theta_g). \tag{8.7}$$

This minimax optimization process can be divided into two operations:

$$\hat{\theta}_d = \underset{\theta_d}{\text{argmax}}\, \mathcal{L}_{\text{VDANN}}(\hat{\theta}_c, \theta_d, \hat{\phi}_e, \hat{\theta}_g), \tag{8.8}$$

$$\left(\hat{\theta}_c, \hat{\phi}_e, \hat{\theta}_g\right) = \underset{\theta_c, \phi_e, \theta_g}{\text{argmin}}\, \mathcal{L}_{\text{VDANN}}(\theta_c, \hat{\theta}_d, \phi_e, \theta_g), \tag{8.9}$$

where symbols with a hat (e.g., $\hat{\theta}_c$) on the right-hand side of Eq. 8.8 and Eq. 8.9 mean that they are fixed during the optimization process. After training, embedded features can be extracted from the last (linear) hidden layer of the encoder E (denoted by $\boldsymbol{\mu}$ in Figure 8.5). Because the approximate variational posterior is regularized to follow a Gaussian distribution, features extracted from the encoder will tend be Gaussian.

8.3.2 Relationship with Domain Adversarial Neural Network (DANN)

DANNs have been used for reducing the mismatch between the source and target domains, where data from the source domain have speaker labels while training data

https://doi.org/10.1017/9781108552332.009 Published online by Cambridge University Press

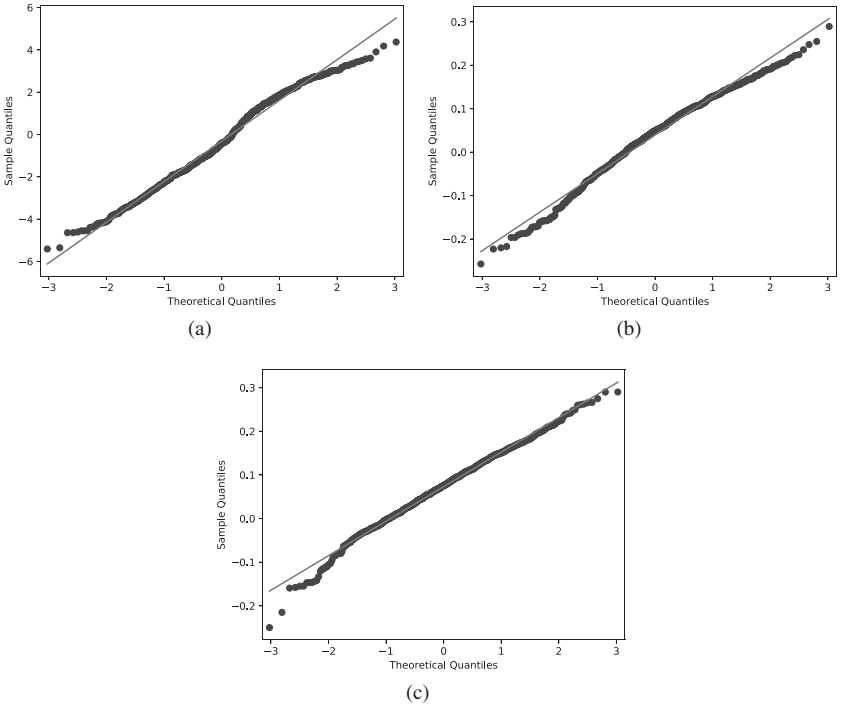

Figure 8.6 Quantile-quantile (Q–Q) plots of the eleventh component of (a) x-vectors, (b) DANN-transformed x-vectors, and (c) VDANN-transformed x-vectors. The vertical and horizontal axes correspond to the samples under test and the samples drawn from a standard normal distribution, respectively. The straight line represents the situation of perfectly Gaussian. The p-values above the graphs were obtained from Shapiro–Wilk tests in which $p > 0.05$ means failing to reject the null hypothesis that the test samples come from a Gaussian distribution.

from the target domain are unlabeled. In that case $R = 2$ in Eq. 8.5. For example, Wang et al. [211] proposed a DANN comprising a feature extractor E, a speaker predictor C and a domain classifier D. Note that the DANN in [211] is a special case of VDANN, because by setting $\beta = 0$ in Eq. 8.4, we obtain the loss function of DANNs:

$$\mathcal{L}_{\text{DANN}}(\theta_c, \theta_d, \phi_e) = \mathcal{L}_C(\theta_c, \phi_e) - \alpha \mathcal{L}_D(\theta_d, \phi_e), \tag{8.10}$$

where θ_c, θ_d, and ϕ_e are the parameters for C, D, E, respectively and α controls the trade-off between the two objectives. $\mathcal{L}_C$ and $\mathcal{L}_D$ are identical to Eq. 8.5 and Eq. 8.6, respectively. The parameters of DANN are optimized by:

$$\hat{\theta}_d = \underset{\theta_d}{\text{argmax}}\, \mathcal{L}_{\text{DANN}}\left(\hat{\theta}_c, \theta_d, \hat{\phi}_e\right), \tag{8.11}$$

$$\left(\hat{\theta}_c, \hat{\phi}_e\right) = \underset{\theta_c, \phi_e}{\text{argmin}}\, \mathcal{L}_{\text{DANN}}\left(\theta_c, \hat{\theta}_d, \phi_e\right). \tag{8.12}$$

https://doi.org/10.1017/9781108552332.009 Published online by Cambridge University Press

Because there is no extra constraint on the distribution of the embedded features, adversarial training may lead to non-Gaussian embedded vectors, which is not desirable for the PLDA back end.

8.3.3 Gaussianality Analysis

Figure 8.6 shows the normal Q–Q plots of the three dimensions of x-vectors and the x-vectors transformed by a DANN and a VDANN. Obviously, the distribution of the x-vectors transformed by the VDANN is closer to a Gaussian distribution than the others. This suggests that the VAE loss can make the embedded vectors $\mathbf{z}$'s and the hidden layer outputs $\boldsymbol{\mu}$ to follow a Gaussian distribution. The p-values obtained from Shapiro–Wilk tests [282] also suggest that the distribution of VDANN-transformed vectors is the closest to the standard Gaussian.

https://doi.org/10.1017/9781108552332.009 Published online by Cambridge University Press