Odyssey 2020 The Speaker and Language Recognition Workshop
1-5 November 2020, Tokyo, Japan

Improving Embedding-based Neural-Network Speaker Recognition

Po-Chin Wang[1], Chia-Ping Chen[1], Chung-li Lu[2], Bo-Cheng Chan[2], Shan-Wen Hsiao[2]

[1]National Sun Yat-Sen University, Taiwan
[2]Chunghwa Telecom Laboratories, Taiwan
m073040069@student.nsysu.edu.tw, cpchen@mail.cse.nsysu.edu.tw
{chungli,cbc,swhsiao}@cht.com.tw

Abstract

In this paper, we integrate multiple ideas and techniques into an embedding-based neural-network speaker recognition (NSR) system. Such an NSR system essentially consists of a front-end speaker-embedding extractor and a back-end speaker-matching component. The frontend is a neural network trained with millions of utterances from thousands of speakers. Currently, the backend is based on simple similarity measures such as angle, Euclidean distance, or probabilistic score. We begin with the well-known x-vector baseline, and then incrementally modify the system modules. Regarding front-end extractor, we investigate modification on network architecture, network function, training criteria, and hyper-parameter setting. Regarding back-end matcher, we evaluate PLDA training/adaptation data and system fusion. On the public SRE 2018 Evaluation Dataset, the performance of system as measured by equal-error rate (EER) is improved from 7.01% to 5.16%, which marks a significant relative improvement of 26.5%.

1. Introduction

Many speaker recognition systems are equipped with trainable neural-network speaker-embedding extractor. We refer this methodology as neural speaker recognition (NSR). In recent years, NSR-based systems have been more intensively researched than the once-popular i-vector methodology [1]. For example, a deep residual neural network (ResNet) [2, 3] has been used for embedding extractor, as well as a time-delay neural network (TDNN). In addition to a front-end speaker-embedding extractor, an NSR-based system is also equipped with a back-end speaker-matching module. Currently, the combination of TDNN-based x-vector [4, 5] front-end embedding extractor and PLDA-based back-end matcher [6], along with the related variants [7, 8], are the mainstream methods, achieving sound performance in speaker recognition evaluations.

The design of front-end embedding extractor of an NSR system requires consideration on network architecture, network function, loss function, and hyper-parameter setting. In this paper, we apply loss functions proposed for face recognition tasks to speaker recognition tasks [9, 10, 11, 12]. In addition, we incorporate learnable dictionary encoder (LDE) [3, 13], self-attentive pooling [14, 15] and layer normalization [16] into network architecture. Furthermore, we investigate the setting of hyper-parameters such as batch size, training archive, and learning rate. Specifically, we use a warm-up learning rate scheme [17]. Regularization methods such as GELU [18] and label smoothing [19] are attempted as well. Finally, we implement system fusion based on BOSARIS [20].

This paper is organized as follows. Section 2 describes the training data and its preprocessing, and the test data used for performance evaluation. Section 3 introduces the neural-network speaker-embedding extractor, including architecture, loss function, and training algorithm. Section 4 introduces the probabilistic speaker-matching back-end module and the score fusion method. Section 5 presents the experimental results of the proposed methods. Finally, Section 6 summarizes this work with concluding remarks and future works.

2. Data

2.1. Training Data

As with many deep learning tasks, the speaker recognition in recent years shows that performance is positively correlated to the amount of data. The selection of data from the training set in each step is also critical to successful training, since it dictates the learning trajectory in the parameter space. Essentially, building a robust speaker recognition system requires speech data from different conditions, such as conversational telephone speech (CTS) and audio from video (AfV). Note that the channels used to convey/record audio signals often differ from instance to instance. Furthermore, when the target language is under-resourced, a speaker recognition system must exploit speech data from extraneous languages. Such domain mismatch problems have certain impact on the performance of the system and this issue requires good remedy.

In an NSR system, the front-end neural-network speaker-embedding extractor and the back-end speaker matcher require training data respectively. In building our NSR system, we use the following data sets.

- SRE: SRE data from 2004 to 2010 and Mixer6 speech
- SWBD: Switchboard dataset containing 6 subsets
- VoxCeleb: VoxCeleb1 and VoxCeleb2
- AISHELL: AISHELL1 and AISHELL2
- SRE18 Dev: SRE 2018 development dataset
- SRE19: SRE 2019 evaluation dataset

The usage of speech data sets in the training process is shown in Table 1. The speech data used for front-end neural network training is quite large. It mainly consists of a large amount of English data and a smaller amount of Mandarin data, including various types of channels. The data for back-end training is relatively simple. First, we use part of the English conversational telephone speech data to train an out-of-domain PLDA. Then, we use in-domain Arabic conversational telephone speech data to adapt the PLDA to the test domain.

10.21437/Odyssey.2020-8

Table 1: *Data usage in the training process.*

Datasets	neural network training	back-end training
SRE	✓	✓
SWBD	✓	
VoxCeleb	✓	
AISHELL	✓	
SRE18 Dev		✓
SRE19		✓

2.2. Data Augmentation

Data augmentation methods are often adopted to increase the amount and diversity of training data for system robustness. These methods include adding noise, reverberation, or perturbation of speed and volume, even different encoding methods to the original speech. Our system uses the augmentation methods included in Kaldi recipe to add noise and reverberation to speech. Specifically, we randomly select speech, music, noises from the MUSAN dataset to add noise to the training speech data. Furthermore, the speech data is also artificially reverberated via convolution with simulated RIRs.

2.3. Acoustic Feature

The input to the neural network is acoustic features extracted from speech data. Specifically, the system uses 23 Mel-Frequency Cepstral Coefficients (MFCC). The audio samples are coded with a 25-ms frame window and a 10-ms frameshift, and the bandwidth is limited to the range of 100 – 3,700 Hz. After feature extraction, energy-based voice activity detection (VAD) is used to estimate frame-by-frame speech activity, and the frames with silence or low signal-to-noise ratio in the audio samples are removed.

2.4. Evaluation Data

We use SRE18 Evaluation Set for system performance evaluation. SRE18 Evaluation Set is an Arabic conversational telephone speech dataset. It consists of 2306 calls from 213 speakers. The duration of the enroll segment is close to 60 seconds and the test segment is between 10 to 60 seconds. The dataset contains 2,063,007 pairs of test data. The evaluation metric used in this paper is the equal-error rate (EER), which is decided by varying decision threshold such that the false rejection (FR) rate defined by

$$P_{\text{FR}}(\theta) = P(s < \theta | y_1 = y_2) \tag{1}$$

is equal to the false acceptance (FA) rate defined by

$$P_{\text{FA}}(\theta) = P(s > \theta | y_1 \neq y_2) \tag{2}$$

In Eq. (1) and (2), θ is the acceptance/rejection decision threshold, and s is the similarity score of the first speech y_1 and second speech y_2. The equal error rate is the value of the false rejection rate or false acceptance rate by choosing θ^* such that

$$\theta^* = \arg\min_{\theta} |P_{\text{FR}}(\theta) - P_{\text{FA}}(\theta)|$$

3. Front-end Neural Network

3.1. Network Architecture

The neural-network embedding extractor is shown in Figure 1. During the training phase, the extracted pre-processed features enter the neural network. In the frame-level layers, the joint representation of increasing range of multiple frames is learned. In the pooling layer, the representation vectors obtained by the frame-level layer are integrated. In the dense layers, the representation of the entire input speech segment is tuned to specific task. We use a classification head to perform speaker classification task, calculate losses, and update the neural network parameters. Finally, the output of the dense layer is directly used as speaker embedding in an NSR system.

3.1.1. Frame Level Layers

In this work, we use two network architectures for frame-level layers. One architecture is the traditional x-vector neural network architecture, as shown in Figure 2, with five layers of TDNN connectivity. The kernel sizes of the first three layers are 5, 3, 3, and the rest is 1. The dilation rates of the second and third layers are 2 and 3. We perform a pilot experiment with TDNN as described in Section 5. The other architecture is the Extended TDNN (E-TDNN) architecture, also shown in Figure 2, with ten layers of TDNN connectivity. Each TDNN layer with a kernel of size other than 1 is followed by a kernel of size 1. Note that the receptive field of E-TDNN is 23, which is wider than TDNN's 15. We also perform a pilot experiment with E-TDNN.

3.1.2. Pooling

In addition to the traditional statistics pooling layer, which simply computes mean and standard deviation without learning, we experiment with self-attentive statistics pooling layer similar to learnable dictionary encoder. The self-attention statistics pooling layer outputs a weighted mean and a weighted standard deviation of the input vectors, where the dependency between data and the frame weights is parameterized and trained. Specifically

$$\boldsymbol{\mu} = \sum_{t=1}^{T} a_t \boldsymbol{h}_t \tag{3}$$

$$\boldsymbol{\sigma} = \sqrt{\sum_{t=1}^{T} a_t \boldsymbol{h}_t \odot \boldsymbol{h}_t - \boldsymbol{\mu} \odot \boldsymbol{\mu}} \tag{4}$$

where $\boldsymbol{h}_t$ represents the frame-level output of the t-th frame, and a_t represents the attention weight of the t-th frame. The attention weight vector $\boldsymbol{a}$ composed of all a_t is calculated by

$$\boldsymbol{a} = \text{softmax}(g(\boldsymbol{H}^T \boldsymbol{W}_1 + \boldsymbol{b})\boldsymbol{W}_2) \tag{5}$$

where $\boldsymbol{H}$ is the output of the frame-level layer, and $g(\cdot)$ is the activation function. $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ represent weights that can be trained. The shapes of the two matrices $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ are determined by the number of channels output by the frame-level layer, the size of the hidden layer of the self attentive pooling layer, and the number of heads of the self attentive pooling layer.

3.1.3. Classification Head

In this work, we are inspired by the loss functions invented for the face recognition task that achieve good performance. These loss functions can be further divided into distance-based and angular-based categories. Compared with the Euclidean distance, the angular distance can be more naturally expressed in the feature space. The x-vector framework uses categorical

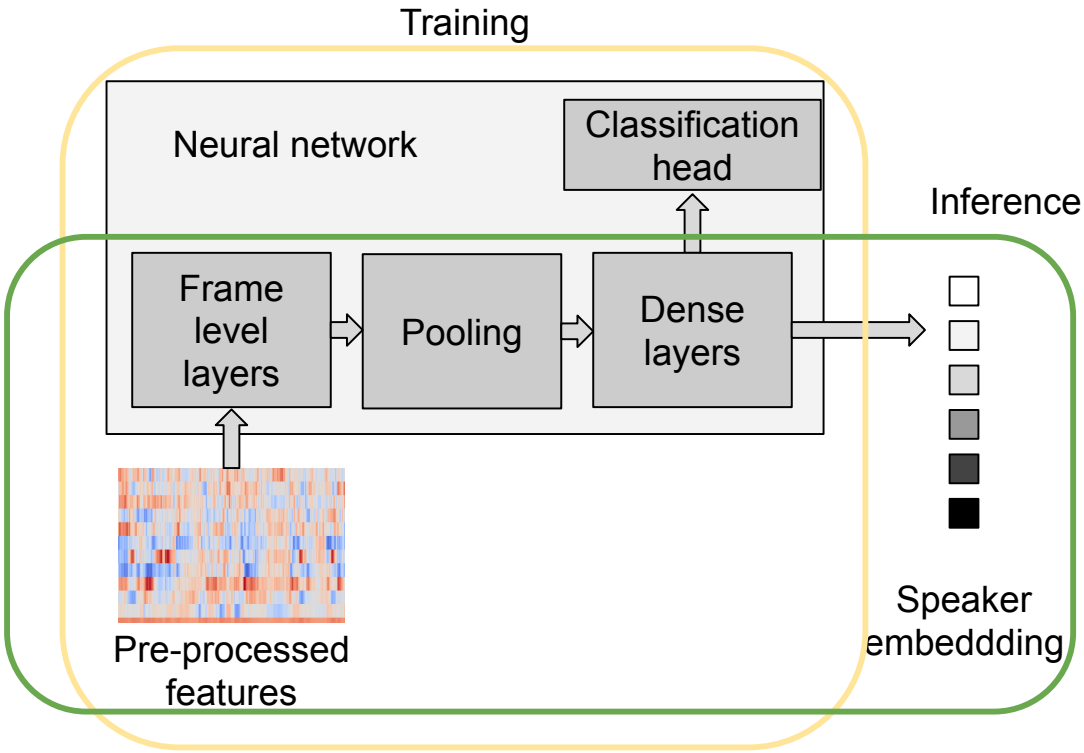


Figure 1: *The composition of the embedding extractor neural network.*

cross-entropy loss function. For data set $\mathcal{D}$

$$L(W, b; \mathcal{D}) = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{W_{y_i}^T x_i + b_{y_i}}}{\sum_{j=1}^{c} e^{W_j^T x_i + b_j}} \quad (6)$$

where N is the number of training samples, W is the weight of the last layer of the model, b is the bias of the last layer, c is the number of categories to be classified and y is ground truth. In Additive Margin Softmax, we impose additional restriction on the value of ground truth, making the classification conditions more stringent. We also normalize W and x so that they have a length of 1, and remove bias. The loss function becomes

$$L = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{s(W_{y_i}^T x_i - m)}}{e^{s(W_{y_i}^T x_i - m)} + \sum_{j=1, j\neq y_i}^{c} e^{s(W_j^T x_i)}} \quad (7)$$

where m is the margin and s is the scaling factor used to scale the values. Based on past experience, the values of m and s used in this paper are 0.3 and 30 respectively.

3.2. Hyper-parameters

3.2.1. Learning Rate

Studies of warm-up learning rates in recent years have shown that it is helpful for optimizing deep learning problems. It helps us get a more generalized model, and the results are relatively unaffected by the initial values. The warm-up stage means that the optimization starts from using a very small learning rate, and then gradually increases it to a predetermined maximum with a predetermined number of iterations. After the warm-up stage, the learning rate gradually decays to a minimum value. On some neural network model architectures, it is shown that the warm-up stage is an indispensable element for successful training models. Especially when the expected gradient near the output layer is large, without a warm-up phase, directly using a large learning rate for these parameters may not lead to model improvement and may even make the optimization process unstable. Using the warm-up phase and training the model with a small learning rate can actually avoid this problem. In this work, our learning rate schedule is

$$lr = 512^{-0.8} \times min(iter^{-0.8}, iter \times w\_iters^{-1.5}) \quad (8)$$

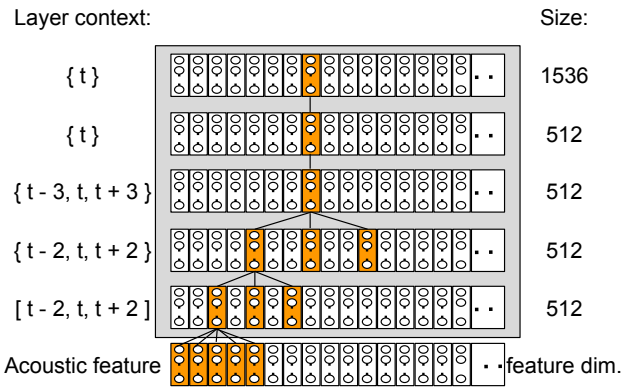
where $iter$ refers to the number of iterations of the current training, and lr is the corresponding learning rate. In addition, w_iters refers to the number of iterations that we define to gradually increase the learning rate.

3.2.2. Training Archives

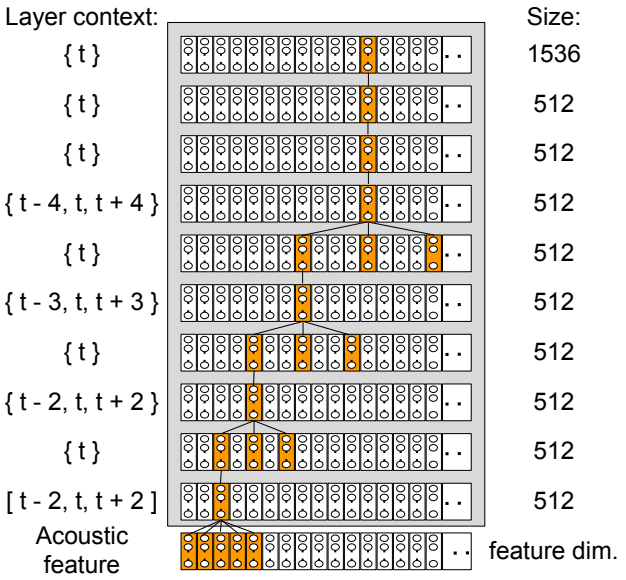
We can increase the frequency of changes in the learning rate by increasing the total number of iterations. We can achieve this effect by reducing the total number of frames per iteration. We use the Kaldi-Tensorflow toolkit [21] to develop the model. The number of frames per training example is between 200 and 400. The total number of frames for each iteration is 2 billions.

3.2.3. Batch Size

Selecting an appropriate batch size for gradient-based optimization is also an important part. In speaker recognition task, many state-of-the-art models use large batch sizes, such as 512 or 1024. However, the decision of batch size requires consideration on the amount of neural network model parameters, the number of frames per training example, and the memory size



(a) TDNN frame-level layers



(b) E-TDNN frame-level layers

Figure 2: The architecture of the frame-level layers used in this work. (a) is a traditional TDNN architecture with 5 layers, and (b) is an extended TDNN with 10 layers. The color indicate the receptive fields between layers.

of the Graphics Processing Unit. Our batch size is 32 in the simplified (pilot) experiment and 128 in the full experiment.

3.3. Meta-algorithm and Regularization

3.3.1. Layer Normalization

Normalization methods have been widely used in neural networks. For example, batch normalization has been one of the reasons for the rapid progress of deep learning in recent years. However, batch normalization is not suitable for some network structures and does not perform well when the batch size is small. Layer normalization, unlike batch normalization which calculates statistics across the entire batch of data, calculates statistics across the feature set. Therefore, layer normalization is not affected by batch size like batch normalization, and even batch size 1 can run. It is currently widely used in recurrent neural networks and performs the same calculations during training and inference.

3.3.2. Gaussian Error Linear Unit

The Rectified Linear Unit (ReLU) is often used as the activation function in x-vector systems. Some variants of x-vector systems use Leaky ReLUs or Parametric ReLU as the activation functions. We replace the ReLU with the Gaussian Error Linear Units (GELU) for the segment-level layers. The GELUs adopt a more probabilistic view of a neuron's output. In our system, GELU function is approximated [18] by

$$G(x) = 0.5\,x\left(1+\tanh\left[\sqrt{\frac{2}{\pi}}\left(x+0.044715x^3\right)\right]\right) \quad (9)$$

3.3.3. Label Smoothing

Label smoothing is a regularization method used by many state-of-the-art neural network models. It makes the network more robust by preventing the neural network from becoming over-confident or misguided by labeling errors. Google Brain explains the reasons why label smoothing helps us by observing the changes in the features of the neural network trained with label smoothing. For a data point of class k, label smoothing adjusts the traditional one-hot label by

$$\boldsymbol{y}_k^{LS} = \boldsymbol{y}_k(1-\alpha) + \frac{\alpha}{K} \quad (10)$$

where $\boldsymbol{y}_k$ is the one-hot vector, K is the total number of classes, and α is a parameter for the degree of smoothing.

4. Back-end Scoring

4.1. Gaussian PLDA

The back-end speaker-matching module is based on Gaussian probabilistic linear discriminant analysis (G-PLDA) scoring. We perform mean normalization of the speaker embeddings extracted by the neural network, and apply mean subtraction to each dataset to reduce the difference variability between datasets. Furthermore, linear discriminant analysis (LDA) is used to project the speaker embeddings into a lower dimension space while retaining the discriminative speaker vector factor.

4.2. Score Fusion

We use ensemble method to achieve best performance. That is, we combine the outputs of multiple sub-systems. Each sub-system consists of a front-end neural-network embedding extractor and a back-end PLDA matcher. Furthermore, the calibration of fusion weights of the sub-systems are done using the BOSARIS toolkit. The labeled part of the SRE18 Dev dataset is used for fusion system calibration.

5. Experiments

5.1. Setting

Training the model with all the data requires expensive computing resources, so we have designed a pilot, i.e. a simplified set of experimental procedures, to evaluate the methods used by speaker recognition neural network models. While the methods that are effective in the simplified experimental process are not necessarily effective in the full training process, the experimental results are still worth observation and analysis.

Table 2: *Results of self-attentive pooling of the pilot experiment.*

Pooling type	Activation	Bias	EER	adapted EER
statistics	-	-	16.29	11.63
self-attentive	RELU	N	17.11	12.16
self-attentive	TANH	Y	16.23	11.49

Table 3: *Results of different number of self-attentive heads of the pilot experiment.*

num-heads	EER	adapted EER
2	16.20	11.50
4	16.23	11.49
6	15.83	11.39
1536	16.51	13.08

Table 4: *Results of classification head of the pilot experiment.*

description	EER	adapted EER
TDNN baseline	16.29	11.63
TDNN AMS	14.3	10.07
TDNN AMS+Ring	14.18	10.12
TDNN AMS+MHE	13.97	10.03

In the pilot experiment, we only use the VoxCeleb dataset without data augmentation as the training data for the neural network. Besides, the number of training iterations is also halved to 3 epochs, and the batch size setting is 32. For the part of PLDA, we used the two datasets VoxCelebCat and SRE18 development set for training and adaptation respectively. VoxCelebCat represents the concatenation of audio files from the same Youtube video in the VoxCeleb data. After that, there are about 170,000 audio data in the new dataset. Finally, the SRE18 Evaluation set is also used for evaluation. We use the standard x-vector TDNN model as the baseline.

5.2. Self-attentive Pooling

Two variants of self-attentive statistics pooling, as given by Eq. (5), are tried. In both cases, the hidden layer of our pooling layer is fixed at 512, and it is a multi-head self-attention of 4 heads. The difference between them is that one uses the Rectified Linear Unit activation function without bias, while the other uses hyperbolic tangent activation function with bias term. The results are shown in Table 2. It can be seen that the second case achieves better performance. We also experiment with different numbers of heads. Note that the maximum number of heads in self-attentive statistics pooling is set to 1536 because this is the number of output channels of the last frame-level layer. The results are shown in Table 3.

5.3. Classification Head

The baseline system uses softmax output function with cross-entropy as the loss function. We replace the softmax function with the additive margin softmax (AM-softmax) function, as given by Eq. (7), with a margin of 0.2 and a scale of 30. As shown in Table 4, the performance of the system is significantly improved with AM-softmax. In addition, we refine the loss function with minimum hyper-spherical energy (MHE) and ring loss [12]. Note that the weight of MHE loss is 1, and the initial value and weight of the ring loss are 20 and 0.01, respectively.

Table 5: *Results of hyper-parameters of the pilot experiment.*

description	EER	adapted EER
TDNN baseline	16.29	11.63
Changed hyper-params	15.58	10.79

Table 6: *Results of meta-algorithm and regularization of the pilot experiment.*

description	EER	adapted EER
TDNN baseline	16.29	11.63
TDNN + GELU	16.66	11.48
TDNN + LN	15.89	11.3
TDNN + LS (α = 0.1)	16.29	11.63
TDNN + LS (α = 0.5)	15.91	11.45

5.4. Hyper-parameters

Regarding training archive, the number of frames per training example is random between 200 and 400, and the total number of frames per iteration is reduced from 5 billions to 2 billions. In addition, we increase the batch size from 32 to 128. The warm-up learning rate schedule, as given in Eq. (8), replaces the original exponential decay. As shown in Table 5, the adjustment of these hyper-parameters has also improved the model.

5.5. Meta-algorithm and Regularization

We change the activation function of frame-level layers from RELU to GELU. Although experimental results show that it does not significantly improve system performance, the randomness it brings can make the model more robust when more training data is used or the training time is longer. Layer normalization is commonly used in recurrent neural networks (RNN) but rarely used in convolutional neural networks (CNN). In [16], the authors pointed out that more research is needed to make layer normalization work well with convolutional neural networks. Since TDNN can be seen as a special case of CNN, it remains to be seen whether layer normalization works well with TDNN as it does with RNN. The results are summarized in Table 6. First, the results show the improvement of layer normalization for TDNN network architecture. The parameter α of label smoothing is also investigated. With 3 epochs on the same training archive, there is no effect on performance with $\alpha = 0.1$ but there is positive effect with $\alpha = 0.5$.

5.6. Single System

We train the model using the data described in Section 2. The model uses 6-head self-attentive pooling with bias and hyperbolic tangent activation function. The model's classification head is AM-Softmax with margin 0.2 and scale 30, plus MHE with unity weight. Regarding hyper-parameters, the total number of frames per iteration is 2 billions, while the batch size is 128. Warn-up learning rate and label smoothing are also adopted in our system. The frame-level layers have GELU activation function and layer normalization. We train the network model for 6 epochs. The loss and accuracy during training are shown in Figure 3. Note that loss and accuracy are calculated from a subset of the training set and the validation set. The re-

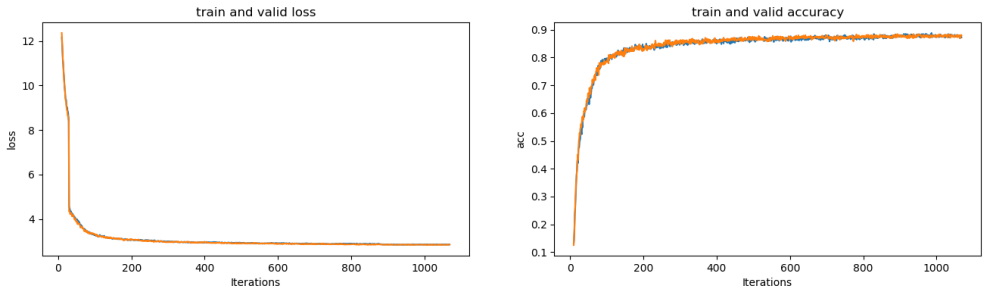


Figure 3: *The graph of the change in loss and accuracy of the model during training.*

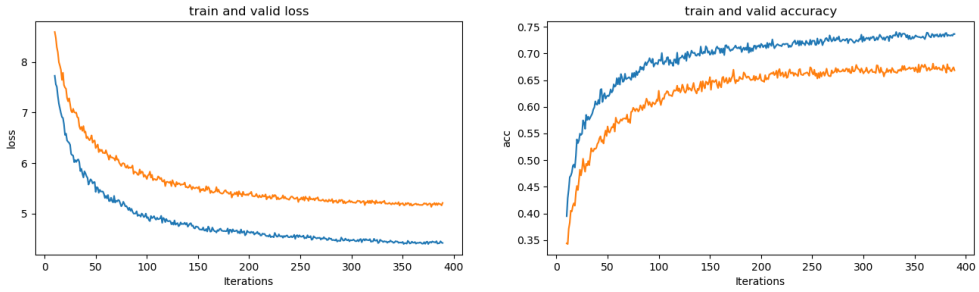


Figure 4: *The graph of the change in loss and accuracy of the previous model during training.*

Table 7: *PLDA with three different data sets and their results.*

PLDA data	adapt data	EER	adapted EER
SRE19	SRE18 Dev, SRE19	6.81	6.83
SRE, SRE19	SRE, SRE19	7.05	6.35
SRE, SRE19	SRE18 Dev, SRE19	6.42	7.41

sults in these two sets are quite similar, so it can be said that the training is not overfitting the training data.

In comparison, the training performance of baseline model is shown in Figure 4. The data used in this model does not include AISHELL data, and the VoxCeleb data is not enhanced with data augmentation. The model arcitecture is also E-TDNN but with statistics pooling, where each layer in frame-level layers uses GELU, and AM-Softmax is the loss function. We can see that the loss and accuracy of the model have obvious differences on the training subset and the validation set.

The data used to train the back-end PLDA has a great impact on the speaker recognition system. We use several data configuration with data augmentation to train and adapt the PLDA model. The results are shown in Table 7. As shown in the second setting, we can see that diversity in PLDA training and adaptation data helps the performance.

5.7. Fusion System

We also implement system fusion, where 3 frontend neural-network models and 3 backend PLDA models are combined. The neural networks include the two models mentioned in the previous section and a baseline model trained with the VoxCeleb dataset only. The PLDA models are trained with the setting as shown in Table 7. The results generated from the 9 frontend-backend combinations are fused with the BOSARIS

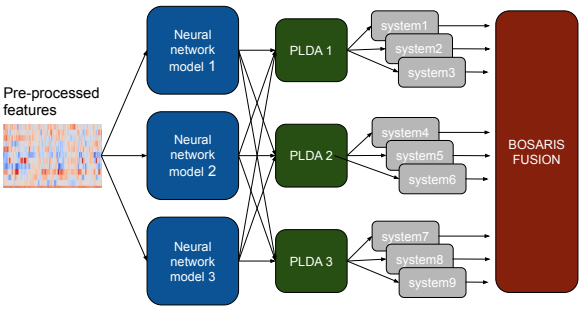


Figure 5: *Schematic illustration of fusion strategy.*

toolkit, as shown in Figure 5. The fusion system can achieve an EER of 5.16% on the SRE18 Evaluation set.

6. Conclusion

In this study, we evaluate modern techniques in neural speaker recognition, including self-attentive pooling, AM-Softmax classification head, auxiliary loss functions, warm-up learning rate, batch size, generation of training archive, GELU activation function, label smoothing, and layer normalization. Furthermore, data selection for the back-end PLDA training and adaptation is experimented. In addition, fusion of single systems based on the constructed frontend/backend models is implemented. The result of the best fusion system achieves an EER of 5.16% on the SRE18 Evaluation set. In the future, a neural-network backend may be designed and integrated with NSR.

7. References

[1] Najim Dehak, Patrick J Kenny, Réda 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, 2010.

[2] Chao Li, Xiaokong Ma, Bing Jiang, Xiangang Li, Xuewei Zhang, Xiao Liu, Ying Cao, Ajay Kannan, and Zhenyao Zhu, "Deep speaker: an end-to-end neural speaker embedding system," *arXiv preprint arXiv:1705.02304*, 2017.

[3] Weicheng Cai, Jinkun Chen, and Ming Li, "Exploring the encoding layer and loss function in end-to-end speaker and language recognition system," *arXiv preprint arXiv:1804.05160*, 2018.

[4] David Snyder, Daniel Garcia-Romero, Daniel Povey, and Sanjeev Khudanpur, "Deep neural network embeddings for text-independent speaker verification.," in *Interspeech*, 2017, pp. 999–1003.

[5] David Snyder, Daniel Garcia-Romero, Gregory Sell, Daniel Povey, and Sanjeev Khudanpur, "X-vectors: Robust dnn embeddings for speaker recognition," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 5329–5333.

[6] Patrick Kenny, "Bayesian speaker verification with heavy-tailed priors.," in *Odyssey*, 2010, vol. 14.

[7] Jesús Villalba, Nanxin Chen, David Snyder, Daniel Garcia-Romero, Alan McCree, Gregory Sell, Jonas Borgstrom, Fred Richardson, Suwon Shon, François Grondin, et al., "State-of-the-art speaker recognition for telephone and video speech: the jhu-mit submission for nist sre18," *Proc. Interspeech 2019*, pp. 1488–1492, 2019.

[8] Chia-Ping Chen, Su-Yu Zhang, Chih-Ting Yeh, Jia-Ching Wang, Tenghui Wang, and Chien-Lin Huang, "Speaker characterization using tdnn-lstm based speaker embedding," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 6211–6215.

[9] Feng Wang, Jian Cheng, Weiyang Liu, and Haijun Liu, "Additive margin softmax for face verification," *IEEE Signal Processing Letters*, vol. 25, no. 7, pp. 926–930, 2018.

[10] Hao Wang, Yitong Wang, Zheng Zhou, Xing Ji, Dihong Gong, Jingchao Zhou, Zhifeng Li, and Wei Liu, "Cosface: Large margin cosine loss for deep face recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5265–5274.

[11] Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou, "Arcface: Additive angular margin loss for deep face recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4690–4699.

[12] Yi Liu, Liang He, and Jia Liu, "Large margin softmax loss for speaker verification," *arXiv preprint arXiv:1904.03479*, 2019.

[13] Weicheng Cai, Zexin Cai, Xiang Zhang, Xiaoqi Wang, and Ming Li, "A novel learnable dictionary encoding layer for end-to-end language identification," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 5189–5193.

[14] Koji Okabe, Takafumi Koshinaka, and Koichi Shinoda, "Attentive statistics pooling for deep speaker embedding," *arXiv preprint arXiv:1803.10963*, 2018.

[15] Yingke Zhu, Tom Ko, David Snyder, Brian Mak, and Daniel Povey, "Self-attentive speaker embeddings for text-independent speaker verification.," in *Interspeech*, 2018, pp. 3573–3577.

[16] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[18] Dan Hendrycks and Kevin Gimpel, "Gaussian error linear units (gelus)," *arXiv preprint arXiv:1606.08415*, 2016.

[19] Rafael Müller, Simon Kornblith, and Geoffrey E Hinton, "When does label smoothing help?," in *Advances in Neural Information Processing Systems*, 2019, pp. 4696–4705.

[20] Niko Brümmer and Edward De Villiers, "The bosaris toolkit: Theory, algorithms and code for surviving the new dcf," *arXiv preprint arXiv:1304.2865*, 2013.

[21] Hossein Zeinali, Lukas Burget, Johan Rohdin, Themos Stafylakis, and Jan Cernocky, "How to improve your speaker embeddings extractor in generic toolkits," *arXiv preprint arXiv:1811.02066*, 2018.