FEEDBACK CONNECTION FOR DEEP NEURAL NETWORK-BASED ACOUSTIC MODELING

Dung T. Tran[1], Marc Delcroix[1], Atsunori Ogawa[1], Christian Huemmer*, Tomohiro Nakatani[1]

[1]NTT Communication Science Laboratories, NTT corporation,
2-4, Hikaridai, Seika-cho (Keihana Science City), Soraku-gun, Kyoto 619-0237 Japan
{dung.tran,marc.delcroix,ogawa.atsunori,nakatani.tomohiro}@lab.ntt.co.jp

ABSTRACT

The use of auxiliary features is an effective way to improve the performance of deep neural network (DNN)-based acoustic models. Most approaches use auxiliary features that represent the speaker or the environment. These auxiliary features are usually computed independently of the acoustic model. This paper investigates a types of auxiliary features obtained from the output of a hidden layer that feeds back to the input layer of the network. Since the auxiliary features are extracted from the hidden layer of the network no external information is required such as the speaker or the environment. Experimentally, by forcing the extraction of the auxiliary features from the same networks, we can further improve the performance of the overall network and reduce the total number of parameters used. We tested this approach with different deep neural network architectures including: deep neural networks, convolutional neural networks and unfolded recurrent convolutional networks. We confirmed the effectiveness of this approach on the CHiME3 dataset.

Index Terms— deep neural network, auxiliary features, bottle-neck features, acoustic modeling

1. INTRODUCTION

Deep neural networks (DNNs) have become an extremely powerful tool for many classification tasks and especially for automatic speech recognition (ASR) [1]. Although DNN-based approaches are clearly superior to the traditional Gaussian Mixture Model/Hidden Markov Model (GMM/HMM), the ASR performance still decreases significantly in real applications for many reasons including an environment mismatch and speaker variations. Augmenting input features with auxiliary features representing noise or a speaker is one effective way to mitigate this problem. Approaches that augment features are effective because they generally make the input features more discriminative than the original input features by concatenating them with addition information. Many approaches have been proposed such as augmenting the input features with a noise estimate [2], environment type [3], uncertainty [4], bottle-neck features [5, 6], high-level features [7, 8], I-vectors [9, 10, 11, 12, 13] or summary vectors [14]. Such approaches are effective in many tasks, including noise robustness, speaker adaptation and low resource language speech recognition.

Bottle-neck features can be classified into one type of auxiliary input feature. Using bottle-neck features turns out to be very effective but they are usually used in combination with GMM/HMM models [15, 16, 17, 18, 19], which are less powerful than hybrid DNN/HMM models. The bottle-neck features used in GMM/HMM models are extracted from one hidden layer with a small number of neurons of a neural network where the output of the neural network predicts the clean features or the HMM state posteriors. A GMM/HMM model is trained on the new features, which are obtained by concatenating bottle-neck features and MFCC features. More recently, other bottle-neck feature approaches that do not require GMM/HMM model have been investigated such as high level features extraction [7] and Prediction Adaptation Correction (PAC) [5, 6]. The PAC approach employs two separate networks. The first network is used to extract bottle-neck features and their context expansion while the second network receives the input features with their corresponding bottle-neck features and tries to correct the prediction from the first network. Two separate objective functions are minimized jointly. This method realized consistent improvement performance in a low resource language speech recognition task [6].

In this paper, we propose another approach for obtaining auxiliary input features. In a similar way to bottle-neck features, the proposed auxiliary features are obtained from a hidden layer in a neural network. However, here we propose sharing the same network parameters for bottle-neck extraction and classification. To exploit such auxiliary features in the neural network, we perform two forward-passes sequentially on the same network. At the first forward-pass, auxiliary features are initialized with 0, and then the input features are augmented with auxiliary features before propagating through the network to obtain the output of the hidden layer right before the output layer. The auxiliary features are then computed by applying a linear layer to the output of the hidden layer. Subsequently, these auxiliary features of the deep neural networks are transferred back to the input layer of the network and again augmented with the same input features to compute a new output at the second forward pass. Note that by initializing the auxiliary features with zeros, two forward passes can be performed on the same network. These types of auxiliary features and bottle-neck features are somewhat similar except that we use the same network and there is no great dimensional reduction at the hidden layer with of the proposed auxiliary feature extraction.

In the remainder of the paper, we introduce the auxiliary features and describe how to exploit the auxiliary features in some conventional architectures of neural network in Section 2. Related work is presented in Section 4. We analyze our experiments on the CHiME3 dataset in Section 5. Finally, Section 6 provides a conclusion and presents some potential future research directions.

*Christian Huemmer is affiliated with Multimedia Communications and Signal Processing, University of Erlangen-Nuremberg, Erlangen, Germany. This work was done while Christian Huemmer was an intern at NTT

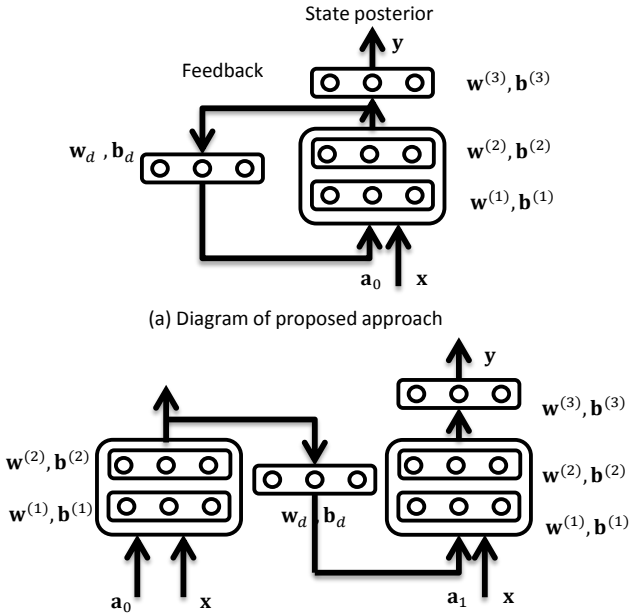

Fig. 1. *Schematic diagram of proposed method.*

2. FEEDBACK CONNECTION

2.1. Principle

In acoustic modeling tasks, a DNN-based classifier receives input features and then outputs the corresponding HMM state posterior probability vectors with the hope that these output vectors match the desired state labels as closely as possible. The network parameters are adjusted by minimizing the loss function between the HMM state posterior probability vectors and the desired state labels. The desired state labels, which encode the states, are perfectly distinguished. Ideally, we anticipate that using the estimation of the desired state labels as auxiliary input features somehow they can increase the discrimination of the input features themselves. Increasing the discrimination between input features makes the DNN classifier easy to classify. To obtain an estimation of the desired state labels, we might need to perform an independent forward pass on another DNN classifier.

There have been several studies [7, 5, 6] showing that exploiting prediction for a simple classifier as an additional feature to a second classifier can improve performance. Usually, such approaches use different classifiers for the feature extraction stage and actual classification stage. In this work we investigate whether or not we can implement a similar concept using the same classifier for the two stages. This is realized using a feedback connection.

2.2. Implementation

This part will start with the general architecture of the proposed approach and then describe in the detail the implementations of a deep fully connected network, a convolutional network and an unfolded recurrent convolutional network. The general architecture of the proposed approach can be seen in Fig. 1. The top figure shows the folded form and the bottom figure shows the unfolded form of the proposed approach. In the unfolded form which is similar to our implementation, there are two networks that share the same parameters. The first network receives the input features and the auxiliary features $\mathbf{a}_0$ initialized at 0. Once the first forward pass is performed, the new auxiliary feature $\mathbf{a}_1$ is obtained. The auxiliary features are extracted from the hidden layer right before the output layer with a non-linearity activation function. Since the output vector of the network is usually very big, we use a connection layer to reduce the dimensions of the auxiliary features. The second network receives the same input features and the auxiliary features $\mathbf{a}_1$ are obtained from the first forward pass. The connection layer is composed of a fully connected layer followed by a non-linearity activation function. It is important to note that the feedback connection is different from the recurrent connection in an RNN since the feedback comes from the same time frame. Moreover, our approach can also be applied to RNN architectures. To clarify our proposed approach we provide below a simple implementation for the 2-hidden-layers network shown in Fig. 1. A sample implementation for fully connected network can be expressed in pseudo-code:

$$\mathbf{a}_0 = [0 \dots 0]^T \tag{1}$$
$$\mathbf{z}_0 = [\mathbf{x}^T, \mathbf{a}_0^T] \tag{2}$$
$$\mathbf{h}_1^{(1)} = \sigma(\mathbf{W}^{(1)} * \mathbf{z}_0 + \mathbf{b}^{(1)}) \tag{3}$$
$$\mathbf{h}_1^{(2)} = \sigma(\mathbf{W}^{(2)} * \mathbf{h}_0^{(1)} + \mathbf{b}^{(2)}) \tag{4}$$
$$\mathbf{a}_1 = \sigma(\mathbf{W}_d * \mathbf{h}_1^{(2)} + \mathbf{b}_d) \tag{5}$$
$$\mathbf{z}_1 = [\mathbf{x}^T, \mathbf{a}_1^T] \tag{6}$$
$$\mathbf{h}_2^{(1)} = \sigma(\mathbf{W}^{(1)} * \mathbf{z}_1 + \mathbf{b}^{(1)}) \tag{7}$$
$$\mathbf{h}_2^{(2)} = \sigma(\mathbf{W}^{(2)} * \mathbf{h}_1^{(1)} + \mathbf{b}^{(2)}) \tag{8}$$
$$\mathbf{y} = \mathbf{W}^{(3)} * \mathbf{h}_2^{(2)} + \mathbf{b}^{(3)} \tag{9}$$

Here the $\mathbf{x}$ is the input feature. $\mathbf{a}_0$ and $\mathbf{a}_1$ are the initialized auxiliary features and the auxiliary features obtained from the first forward pass. σ denotes the non-linearity activation function. $\mathbf{W}^{(i)}$ and $\mathbf{b}^{(i)}$ are the weight matrices and bias vectors of the i^{th} layers. $\mathbf{h}_1^{(i)}$ and $\mathbf{h}_2^{(i)}$ are the outputs of the i^{th} layers for the 1^{st} and 2^{nd} forward passes, respectively. $\mathbf{W}_d$ and $\mathbf{b}_d$ are the weight matrix and bias vector of the connection layer respectively, $\mathbf{y}$ is the output of network right before the soft-max layer. If there are two classifiers are used, then the number of parameters will be roughly double that of a conventional DNN. By sharing parameters between two forward passes as shown in the pseudo-code, we need only one more fully connected layer to significantly reduce the total number of parameters used. In other words, this approach can potentially be applied to a more complex network. In the following, we will explaining in more detail the use of auxiliary features in different architectures.

3. INVESTIGATED NETWORK ARCHITECTURE

The feedback connection is a general one and can be used with any network architecture. We tested this approach for three different configurations which are detailed below.

3.1. Deep fully connected network

In a simple case, the deep fully connected network has 5 layers with sigmoid activation functions used at all layers. The implementation of a deep fully connected network with a feedback connection is similar to the above pseudo-code with more layers. The connection layer has only one fully connected layer to reduce the dimensions

of the auxiliary features. We reduce the dimensions to those of the input features. Note that dropout is used at every fully connected layer and right after the non-linear activation functions.

3.2. Convolutional neural network

The configuration of this simple CNN can be found in [20]. The CNN we used here has two convolutional layers each followed by a max-pooling layer. One MLP with three fully connected layers is used on the top of the last max-pooling layer to predict the state posterior probabilities. Dropout is used at all fully connected layers except the output layer. The auxiliary features are extracted at the second fully connected layer of the MLP. One connection layer is also used to transfer the auxiliary features to the input layer. To be consistent with CNN, the output vector of the connection layer is reformed so that it has a similar shape to the input features. The auxiliary features are then treated as additional feature maps to the input of the first CNN layer. We used two separate convolutional layers to process the auxiliary features and the input features.

3.3. Unfolded recurrent convolutional neural network

Our implementation of the unfolded recurrent convolutional neural network (uRNN-CNN) is inspired by the unfolded recurrent neural network [21]. The architecture of the network is shown in Figure 2. Similar to [21], input features with a certain number of frames are divided into $T+1$ blocks with a smaller number of frames where the next block is shifted 1 frame to the right of the current block. The frame blocks are denoted as $x_{t=0}, x_{t=1}, x_{t=2}...x_{t=T}$. Each block is input into the CNN described in Section 3.2. Note that the filter size of the first convolutional layer is smaller than the filter size in the Section 3.2 due to the input features having fewer number of frames than those in Section 3.2. The output of the CNNs is input into 3-layer MLPs to predict the state posterior probabilities. One recurrent connection is used to connect the hidden layer of the MLPs of the current block to the hidden layer of the MLPs of the next block. Dropout is used for both recurrent and non-recurrent fully connected layers at the frame level. We found that using dropout in such way provides superior performance compared to using drop-out at the non-recurrent layer only. Note that, each recurrent layer has its own dropout meaning that each recurrent connection is differently disconnected for one sample in one mini-batch. One connection layer is also used to transfer the auxiliary features to the input layer. The output vector of the connection layer is reformed so that it has a similar shape to one block of the input features. Similar to CNN, we used two separate convolutional layers to process the auxiliary features and the input features.

4. RELATION TO PRIOR WORK

The proposed method/architecture shares some similarities with other approaches [7, 5, 6] since both use information in addition to the input features with the aim of obtaining a better HMM state posterior probabilities at the network output.

However, compared with [5, 6], this approach has few differences: first, sharing parameters between two forward-pass computation steps greatly reduces the number of parameters by up to roughly 50% in the DNN configuration described in Section 3.1. Moreover, it feasible to apply it to larger, more complex networks since it needs only one more connection layer. Because two networks share the same parameters, only one objective function is used, which may arguably simplify training. Second, this approach uses only the output of the hidden layer of the network itself so no external information is required. Third, this approach is also evaluated on different general architectures of neural networks.

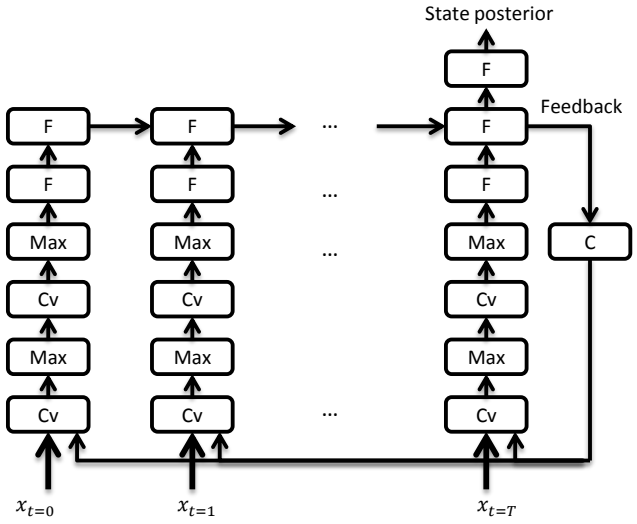

Fig. 2. *Unfolded recurrent convolutional neural network (uRNN-CNN) with auxiliary features. The Cv, Max, F, C mean a convolutional, a max-pooling, a fully connected layers and connection layer, respectively*

This approach is more similar to [7] because neither approaches uses external information. In [7], DNN-based feature extraction followed by a (shallow) sequence classifier but here we used more deeper classifier. In addition, sharing parameters between two forward-pass computation steps is a contribution of this approach compared to that described in [7]. Moreover, an effort of our approach is to show how to exploit auxiliary features for a stronger baseline such as CNN and uRNN-CNN networks in an efficient way.

5. EXPERIMENTAL

5.1. Overview of CHiME3 task

We performed experiments using the CHiME-3 corpus [22], which consists of real speech recordings collected in four different environments, i.e. cafe (CAF), street junction (STR), public transport (BUS), and pedestrian area (PED). The corpus also includes simulated training and test data sets. In this study, we used the real data for evaluation. The corpus consists of read speech, where the prompts were taken from the WSJ0 corpus. The training set comprises 1600 real and 7138 simulated utterances, which amounts to 18 hours of speech. The development and evaluation sets for the real recordings consist of 1640 and 1320 utterances, respectively, spoken by four different speakers.

5.2. Experimental settings

We used speech features consisting of 40 log mel filterbank coefficients appended with static, Δ and $\Delta\Delta$ coefficients. One speech feature is a 1320 dimensional vector. These features were extracted by using a 25-msec sliding window with a 10-msec shift. The speech features were processed with utterance level cepstral mean normalization, and further normalized using mean and variance normalization parameters calculated on the training data. In all the network configurations, a soft-max layer is used to compute the HMM state posteriors. The output consists of 5976 output units corresponding

Table 1. *WER for the CHiME3 development set on noisy speech.*

Methods	BUS	CAF	PED	STR	Ave
DNN	21.96	15.50	12.01	15.72	16.30
DNN + BN	21.63	15.53	11.08	15.31	15.88
DNN wo sharing	20.87	14.78	11.34	15.26	15.56
DNN w sharing	21.26	14.35	11.17	15.10	**15.47**

Table 2. *WER for the CHiME3 eval set on noisy speech.*

Methods	BUS	CAF	PED	STR	Ave
DNN	36.66	28.45	23.36	18.85	26.83
DNN + BN	37.32	28.89	24.23	17.94	27.09
DNN wo sharing	35.30	26.99	22.16	17.10	25.38
DNN w sharing	34.60	26.82	21.47	17.06	**24.98**

Table 3. *WER for the CHiME3 development set on noisy speech.*

Methods	BUS	CAF	PED	STR	Ave
CNN	18.51	10.97	9.25	12.59	12.83
CNN + aux	18.07	10.87	8.88	12.25	12.51
uRNN-CNN	17.60	10.40	8.19	11.46	11.91
uRNN-CNN + aux	16.83	9.53	7.77	10.77	**11.22**

Table 4. *WER for the CHiME3 eval set on noisy speech.*

Methods	BUS	CAF	PED	STR	Ave
CNN	28.62	23.07	18.01	15.46	21.29
CNN + aux	27.37	22.36	17.83	14.98	20.63
uRNN-CNN	28.92	21.63	17.19	14.29	20.50
uRNN-CNN + aux	26.81	20.64	16.59	13.63	**19.41**

to the HMM states. The acoustic model was trained using only one channel of audio data from channel 5. We trained the acoustic model using mini-batch stochastic gradient descent (SGD) to minimize the cross entropy criterion. All the network parameters were randomly initialized. We used an initial learning rate of 0.08, a momentum of 0.9 and a batch size of 128. The learning rate was gradually decreased when the frame accuracy did not improve for a cross validation set. The learning was stopped after 40 epochs. The dropout rate was set at 0.5 for all configurations. We used a trigram language model for decoding.

Our DNN baseline has 5 hidden layers. Each hidden layer has 2048 neurons. We employed 11 concatenated 120×1 speech features as the input to the DNN. The input features are formed in a 1320 dimensional vector. The connection layer has only one fully connected layer to transfer the auxiliary features to a 1320-dimension vector.

In our simple CNN baseline, the first convolutional layer uses (5×11) filters, 3 input and 180 output feature maps. The second convolutional layer uses (1×5) filters, 180 input and 180 output feature maps. After each convolution layer, the resolution of the output feature map is reduced using max-pooling. Three fully connected layers with 2048 output nodes are used. These features were arranged in three (40×11) input feature maps, one for each static, Δ and $\Delta\Delta$ coefficients. The connection layer has only one fully connected layer with which to transfer the auxiliary features to a 1320-dimension vector. This auxiliary features are then reformed so that they have a similar shape to the input features.

In our uRNN-CNN, 11 frames of input features are divided into five 7-frame blocks. The connection layer has only one fully connected layer with which to transfer the auxiliary features to an 840-dimension vector. Again, this auxiliary features are then reformed so that they have a similar shape to the input features.

5.3. Results

5.3.1. Preliminary result with DNN

We performed an experiment with the DNN baseline and obtain a 26.83% WER on the evaluation set as shown the Table 2. With DNN + bottle-neck features (DNN+BN), we first trained the bottle-neck features in advance using the cross-entropy loss function. The 80-dimension bottle-neck features are then extracted from the 4^{th} layer of an DNN network that has 5 hidden layers. The input features are then concatenated with the bottle-neck features. The DNN network which has same architecture as the DNN baseline is trained on these concatenated features. As shown in Table 2, the (DNN+BN)' results are worse than the DNN baseline results on the evaluation set (although it has a better results on development set). A configuration (DNN wo sharing) is formed that has one DNN for computing auxiliary features and another DNN for classification. In this configuration, two cross-entropy objective functions are used independently for auxiliary extraction and classification. We set the weight at 0.5 for each objective function. As can be seen in Table 1 and Table 2, this configuration obtained a 25.38% WER which is significantly better than the DNN baseline. When two DNNs share the same parameters (DNN w sharing), we obtained further improvement. We achieves a 24.98% WER which is a 0.4% absolute WER reduction compared with using two separate DNN networks. It turns our that by forcing two networks to have the same parameters, we can further improve network performance and reduce the number of parameters which is greatly required for DNN applications.

5.3.2. Results with other architectures

Similar to the preliminary results, augmenting the auxiliary features also consistently provided better performance than the CNN and uRNN-CNN baselines as shown in Table 3 and Table 4. More precisely, it achieved 0.6% and 1.1% absolute WER reductions compared with the CNN and uRNN-CNN baselines, respectively. Augmenting auxiliary features to the deep fully connected network is a straightforward implementation that simply involves concatenating auxiliary features with input features. However, with CNN or uRNN-CNN, it is better to use two convolution layers separately; one for the input features and other one for the auxiliary features. Concatenating input features with auxiliary features resulted in significantly worse performance than using two separate convolutional layers. This might be because the contiguity between the input features and the auxiliary features makes networks difficult to distinguish.

6. CONCLUSION

This paper presented a way to exploit the discrimination of the output of the deeper layer to increase the performance of the network itself even without using external information. We found that augmenting the input features with the output of the last hidden layer of the same network provided significant improvements compared with a conventional network. The approach consistently outperformed to all network architecture baselines. In the future, we will investigate this approach into relation adaptation tasks and apply it to more complex and deeper networks.

7. REFERENCES

[1] G. Hinton, L. Deng, D. Yu, G. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of fours research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[2] M. L. Seltzer, D. Yu, and Y. Wang, "An investigation of deep neural networks for noise robust speech recognition," in *ICASSP*, 2013, pp. 7398–7402.

[3] S. Kim, B. Raj, and I. Lane, "Environmental noise embeddings for robust speech recognition," in *arXiv preprint arXiv:1601.02553*, 2016.

[4] Y. Tachioka and S. Watanabe, "Uncertainty training and decoding methods of deep neural networks based on stochastic representation of enhanced feature," in *INTERSPEECH16*, 2015, pp. 3541–3545.

[5] Y. Zhang, D. Yu, M. L. Seltzer, and J. Droppo, "Speech recognition with prediction-adaptation-correction recurrent neural networks," in *ICASSP)*, 2015, pp. 5004–5008.

[6] Y. Zhang, E. Chuangsuwanich, J. Glass, and D. Yu, "Prediction-adaptation-correction recurrent neural networks for low-resource language speech recognition," in *ICASSP*, 2016, pp. 5415–5419.

[7] L. Deng and J. Chen, "Sequence classification using the high-level features extracted from deep neural networks," in *ICASSP*, May 2014, pp. 6844–6848.

[8] B. Li, T. N. Sainath, R. J. Weiss, K. W. Wilson, and M. Bacchiani, "Neural network adaptive beamforming for robust multichannel speech recognition," in *INTERSPEECH*, 2016.

[9] G. Saon, H. Soltau, D. Nahamoo, and M. Picheny, "Speaker adaptation of neural network acoustic models using i-vectors," in *ASRU*, Dec 2013, pp. 55–59.

[10] A. Senior and I. Lopez-Moreno, "Improving DNN speaker independence with i-vector inputs," in *ICASSP*, 2014.

[11] T. N. Sainath, B. Kingsbury, G. Saon, H. Soltau, A. Mohamed, G. Dahl, and B. Ramabhadran, "Deep convolutional neural networks for large-scale speech tasks," *Neural Networks*, vol. 64, pp. 39 – 48, 2015, Special Issue on Deep Learning of Representations.

[12] M. Delcroix, K. Kinoshita, C. Yu, A. Ogawa, T. Yoshioka, and T. Nakatani, "Context adaptive deep neural networks for fast acoustic model adaptation in noisy conditions," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, March 2016, pp. 5270–5274.

[13] M. Delcroix, K. Kinoshita, T. Hori, and T. Nakatani, "Context adaptive deep neural networks for fast acoustic model adaptation," in *ICASSP*, April 2015, pp. 4535–4539.

[14] K. Vesely, S. Watanabe, K. Zmolikova, M. Karafiat, L. Burget, and J. Cernocky, "Sequence summarizing neural network for speaker adaptation," in *ICASSP*, 2016, pp. 5315–5319.

[15] H. Hermansky, D. P. W. Ellis, and S. Sharma, "Tandem connectionist feature extraction for conventional hmm systems," in *ICASSP*, 2000, vol. 3, pp. 1635–1638.

[16] F. Grezl, M. Karafiat, S. Kontar, and J. Cernocky, "Probabilistic and bottle-neck features for lvcsr of meetings," in *ICASSP*, April 2007, vol. 4, pp. 757–760.

[17] M. Seltze D. Yu, "Improved bottleneck features using pretrained deep neural networks," in *INTERSPEECH*, 2011, pp. 27–31.

[18] T. N. Sainath, B. Kingsbury, and B. Ramabhadran, "Auto-encoder bottleneck features using deep belief networks," in *ICASSP*, 2012, pp. 4153–4156.

[19] J. Gehring, Y. Miao, F. Metze, and A. Waibel, "Extracting deep bottleneck features using stacked auto-encoders," in *ICASSP*, 2013, pp. 3377–3381.

[20] T. Yoshioka, N. Ito, M. Delcroix, A. Ogawa, K. Kinoshita, M. Fujimoto, C. Yu, W. J. Fabian, M. Espi, T. Higuchi, S. Araki, and T. Nakatani, "The ntt chime-3 system: Advances in speech enhancement and recognition for mobile multi-microphone devices," in *ASRU*, 2015, pp. 436–443.

[21] G. Saon, H. Soltau, A. Emami, and M. Picheny, "Unfolded recurrent neural networks for speech recognition," in *INTERSPEECH*, 2014, pp. 343–347.

[22] J. Barker, R. Marxer, E. Vincent, and S. Watanabe, "The third CHiME speech separation and recognition challenge: Dataset, task and baselines," in *ASRU*, Dec 2015, pp. 504–511.