On Speaker Adaptation of Long Short-Term Memory Recurrent Neural Networks

Yajie Miao, Florian Metze

Language Technologies Institute, School of Computer Science, Carnegie Mellon University
{ymiao,gowayyed,haoz1,fmetze}@cs.cmu.edu

Abstract

Long Short-Term Memory (LSTM) is a recurrent neural network (RNN) architecture specializing in modeling long-range temporal dynamics. On acoustic modeling tasks, LSTM-RNNs have shown better performance than DNNs and conventional RNNs. In this paper, we conduct an extensive study on speaker adaptation of LSTM-RNNs. Speaker adaptation helps to reduce the mismatch between acoustic models and testing speakers. We have two main goals for this study. First, on a benchmark dataset, the existing DNN adaptation techniques are evaluated on the adaptation of LSTM-RNNs. We observe that LSTM-RNNs can be effectively adapted by using speaker-adaptive (SA) front-end, or by inserting speaker-dependent (SD) layers. Second, we propose two adaptation approaches that implement the SD-layer-insertion idea specifically for LSTM-RNNs. Using these approaches, speaker adaptation improves word error rates by 3-4% relative over a strong LSTM-RNN baseline. This improvement is enlarged to 6-7% if we exploit SA features for further adaptation.

Index Terms: Long Short-Term Memory, recurrent neural network, acoustic modeling, speaker adaptation

1. Introduction

The application of deep learning has achieved tremendous success in acoustic modeling. On a wide range of large vocabulary continuous speech recognition (LVCSR) tasks, deep neural networks (DNNs) have shown better performance than the traditional Gaussian mixture models (GMMs) [1, 2, 3]. Although making significant advances, DNNs, as well as the more advanced convolutional neural networks (CNNs) [4, 5, 6], can model only limited temporal dependency that is provided by the fixed-size window of acoustic frames. As a result, DNNs are not an optimal modeling tool for the complex speech signal with long-range dynamics. To resolve this limitation, previous work [7, 8] has studied recurrent neural networks (RNNs) as acoustic models. With self-connections on their hidden layers, RNNs allow temporal information to be propagated through many time steps. However, training of conventional RNNs can become problematic due to the gradient vanishing and exploding problem [9]. The Long Short-Term Memory (LSTM) architecture [10] provides a solution to overcome the weakness of RNNs. LSTMs exploit memory cells to store temporal information and purpose-built gates to control the information flow. The incorporation of LSTM enables RNNs to learn long-range temporal dependency. Past work [11, 12, 13, 14] has applied LSTM-RNNs to acoustic modeling and shown state-of-the-art performance.

Another issue that acoustic models, both GMMs and DNNs, encounter is the mismatch between acoustic models and testing speakers. Although displaying superior generalization ability than GMMs [15], DNN models still experience a performance degradation when ported from training speakers to unseen testing speakers. To mitigate the effects of this mismatch, past work has proposed various methods for speaker adaptation [16] of DNN and CNN models. These methods can be categorized into three classes. First, the SI model, or certain layers of the model, are re-updated on the adaptation data of each testing speaker [17, 18]. Second, SI-DNN models are augmented with additional speaker-dependent (SD) layers that are learned on the adaptation data [19, 20]. Third, the acoustic model is trained (and decoded) using speaker-adaptive (SA) features [3, 21, 22] or features enriched with SD information [23, 24, 25, 26]. Though adaptation of DNNs and CNNs is well studied, to the best of our knowledge, no previous work has dealt with speaker adaptation of LSTM-RNNs on large-scale acoustic modeling.

In this paper, we present an extensive study to investigate the unsupervised speaker adaptation of LSTM-RNN models. On the benchmark Switchboard dataset, the performance of the aforementioned three classes of adaptation techniques is evaluated for LSTM-RNNs. Moreover, we propose two approaches that implement the idea of inserting SD layers. The first approach is to insert linear *input features transforms* (IFTs) atop of single frames of network inputs. To distinguish the behaviours of different components (memory cells and gates) in LSTM, separate transforms are added for individual components. Second, instead of inserting SD LSTM layers, we propose to insert *hidden activations transforms* (HATs) between the outputs of a LSTM layer and the inputs of the next layer. We study the recurrent and non-recurrent versions of HAT. Experiments show that adaptation with the proposed methods improves a competitive SI LSTM-RNN model by 3-4% relatively. The improvement from adaptation can be further enlarged to 6-7% when we apply adaptive front-end together with IFT-based adaptation.

2. Review of LSTM-RNNs

Compared to the standard feedforward architecture, RNNs have the advantage of learning complex temporal dynamics on sequences. Given an input sequence $\mathbf{X} = (\mathbf{x}_1, ..., \mathbf{x}_T)$, a traditional recurrent layer iterates from $t = 1$ to T to compute the sequence of hidden states $\mathbf{H} = (\mathbf{h}_1, ..., \mathbf{h}_T)$ via the following equations:

$$\mathbf{h}_t = \sigma(\mathbf{W}_{xh}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h) \tag{1}$$

where $\mathbf{W}_{xh}$ is the input-to-hidden weight matrix, $\mathbf{W}_{hh}$ is the hidden-to-hidden (recurrent) weight matrix. In addition to the inputs $\mathbf{x}_t$, the hidden activations $\mathbf{h}_{t-1}$ from the previous time step are fed to influence the hidden outputs at the current time

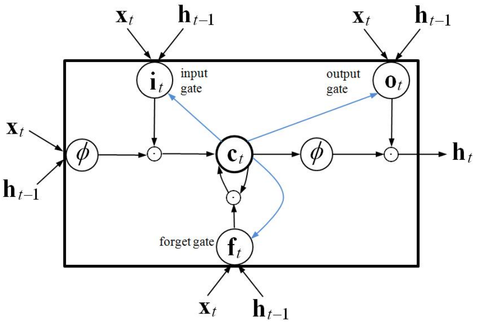

Figure 1: A memory block of LSTM.

step. Learning of RNNs can be done using back-propagation through time (BPTT). However, in practice, training RNNs to learn long-term temporal dependency can be difficult due to the well-known vanishing and exploding gradients problem [9]. Gradients propagated though the many time steps (recurrent layers) decay or blow up exponentially. The LSTM architecture [10] provides a solution that partially overcomes the weakness of RNNs. LSTM contains memory cells with self-connections to store the temporal states of the network. Additionally, multiplicative gates are added to control the flow of information: the input gate controls the flow of inputs into the memory cells; the output gate controls the outputs of memory cells activations; the forget gate regulates the memory cells so that their states can be forgotten. Furthermore, as research on LSTMs has progressed, the LSTM architecture is enriched with peephole connections [27]. These connections link the memory cells to the gates to learn precise timing of the outputs.

Given the input sequence, a LSTM layer computes the gates (input, output, forget) and memory cells activations sequentially from t = 1 to T. The computation at the time step t can be described as:

$$\mathbf{i}_t = \sigma(\mathbf{W}_{ix}\mathbf{x}_t + \mathbf{W}_{ih}\mathbf{h}_{t-1} + \mathbf{W}_{ic}\mathbf{c}_{t-1} + \mathbf{b}_i) \quad (2a)$$
$$\mathbf{f}_t = \sigma(\mathbf{W}_{fx}\mathbf{x}_t + \mathbf{W}_{fh}\mathbf{h}_{t-1} + \mathbf{W}_{fc}\mathbf{c}_{t-1} + \mathbf{b}_f) \quad (2b)$$
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \phi(\mathbf{W}_{cx}\mathbf{x}_t + \mathbf{W}_{ch}\mathbf{h}_{t-1} + \mathbf{b}_c) \quad (2c)$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_{ox}\mathbf{x}_t + \mathbf{W}_{oh}\mathbf{h}_{t-1} + \mathbf{W}_{oc}\mathbf{c}_{t-1} + \mathbf{b}_o) \quad (2d)$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \phi(\mathbf{c}_t) \quad (2e)$$

where $\mathbf{i}_t$, $\mathbf{o}_t$, $\mathbf{f}_t$, $\mathbf{c}_t$ are the activation vectors of the input gate, output gate, forget gate and memory cell respectively. The $\mathbf{W}_{.x}$ terms denote the weight matrices connecting the inputs with the units. The $\mathbf{W}_{.h}$ terms denote the weight matrices connecting the memory cell states of the previous time step $t-1$ with different units. The terms $\mathbf{W}_{ic}$, $\mathbf{W}_{oc}$, $\mathbf{W}_{fc}$ are diagonal weight matrices for peephole connections. Also, σ is the logistic sigmoid nonlinearity which squashes its inputs to the [0,1] range, whereas ϕ is the hyperbolic tangent nonlinearity squashing its inputs to [-1, 1]. The operation $\odot$ represents element-wise multiplication of vectors.

The number of parameters in a LSTM layer is dominated by the number of memory cells. To reduce the model size, [12] proposes to add a projection layer over the memory cells. At the time step t, this layer projects the memory cells outputs $\mathbf{h}_t$ to a lower-dimensional vector $\mathbf{r}_t$ via a linear transform $\mathbf{W}_{rh}$. This projection layer is recurrent in that $\mathbf{r}_t$ will be fed as inputs for the next time step. That is, $\mathbf{h}_{t-1}$ in Equations 2(a$\sim$d) is replaced with $\mathbf{r}_{t-1}$. In addition to reducing model parameters, adding the projection layer is found to generate better recognition accuracy [12]. In this work, we use the LSTM layer with this projection layer in our LSTM-RNN architecture.

3. Speaker Adaptation of LSTM-RNNs

This work studies speaker adaptation of LSTM-RNNs. As reviewed in Section 1, there have been three classes of adaptation methods for DNNs and CNNs: re-updating the SI model, inserting (and then learning) SD layers, training and decoding with SA features. The first and third categories are straightforward to implement for LSTM-RNNs. We will show more details and their results in our experiments. In this section, we focus on the second category, that is, speaker adaptation with additional SD layers. We propose two approaches to implementing this idea.

3.1. Input Features Transform (IFT)

As with previous work on DNN adaptation, the SD layers can be inserted atop of the input features. We apply a linear transform over the single acoustic frames, without considering the context window. Formally, for each testing speaker s, we have a matrix $\mathbf{W}_s$ with the size of $d \times d$, where d is the dimension of input vector $\mathbf{x}_t$. After feature transformation, the inputs into the LSTM-RNN is $\mathbf{z}_t = \mathbf{W}_s\mathbf{x}_t$. The formulation of the LSTM-RNN on this speaker remains to be Equations 2(a$\sim$e), with the only modification that $\mathbf{x}_t$ is replaced by $\mathbf{z}_t$. Since $\mathbf{z}_t$ is not involved in the recurrence, training of the transform can be done using error back-propagation in a batch mode. From BPTT within the LSTM layer, we can get the derivative of the optimization objective (e.g., cross-entropy) $\mathbb{L}$ with respect to $\mathbf{z}_t$. The derivative vector is denoted as $\boldsymbol{\delta}_t^{(z)} = \frac{\partial \mathbb{L}}{\partial \mathbf{z}_t}$. With this, the $T \times d$ derivative matrix $\boldsymbol{\Delta}_z = (\boldsymbol{\delta}_1^{(z)}, ..., \boldsymbol{\delta}_T^{(z)})$ is constructed on the entire sequence. Then, the gradients of $\mathbf{W}_s$ accumulated on this sequence is computed as:

$$\frac{\partial \mathbb{L}}{\partial \mathbf{W}_s} = \boldsymbol{\Delta}_z^T \mathbf{X} \quad (3)$$

where $\mathbf{X} = (\mathbf{x}_1, ..., \mathbf{x}_T)$ is the input matrix with the dimension of $T \times d$.

So far, we have used a single feature transform with respect to the different LSTM components $\mathbf{i}_t$, $\mathbf{f}_t$, $\mathbf{c}_t$ and $\mathbf{o}_t$. This solution may be suboptimal because these components are responsible for different parts of the information flow within LSTM. Therefore, they may also need to operate in separate feature spaces in order for better speaker adaptation. We apply a *component-specific* version of the IFT approach, in which each component has an individual feature transform. For example, the transform $\mathbf{W}_s^{(i)}$ is specific to speaker s and the input gate. The adapted features for the input gate are $\mathbf{z}_t^{(i)} = \mathbf{W}_s^{(i)}\mathbf{x}_t$ which replaces $\mathbf{x}_t$ in Equation 2(a). In this case, learning of the feature transforms is also component specific. Gradients for each feature transform are derived from the derivatives of its corresponding transformed features.

3.2. Hidden Activations Transform (HAT)

For speaker adaptation of DNNs, past work has attempted to augment the SI model with additional SD hidden layers [19, 28]. These SD layers are learned on the adaptation data. In the LSTM-RNN architecture, the recurrent LSTM layer is complex and contains many parameters. It is hard to train a SD LSTM layer reliably on the limited adaptation data. To resolve this issue, we propose to insert a SD linear-transform layer atop of the LSTM layer. When the LSTM-RNN has projection layers, such a HAT layer is applied to the memory cell outputs $\mathbf{r}_t$. The transform matrix for the speaker s is $\mathbf{M}_s$ with the size of $|\mathbf{r}_t| \times |\mathbf{r}_t|$. The transformed activations $\hat{\mathbf{r}}_t = \mathbf{M}_s \mathbf{r}_t$ are propagated as the inputs to the next layer. There is no temporal recurrence involved in the application of $\mathbf{M}_s$. Therefore, $\mathbf{M}_s$ can be applied to the entire sequence in a batch mode, with a matrix-matrix multiplication. Learning of such a matrix is also straightforward. Its gradients are derived easily from the back-propagated errors of the inputs of the atop layer.

Alternatively, the transform $\mathbf{M}_s$ can be applied in a recurrent manner. In the forward propagation, the transformed activations $\hat{\mathbf{r}}_{t-1}$ are treated as $\mathbf{h}_{t-1}$ in Equations 2(a$\sim$d) for computation of $\mathbf{i}_t$, $\mathbf{f}_t$, $\mathbf{c}_t$ and $\mathbf{o}_t$. In this case, the transform has to be applied sequentially frame by frame. In the back propagation, the gradients of $\mathbf{M}_s$ need to be propagated through the time steps. Thus, BPTT is employed to optimize $\mathbf{M}_s$ on the adaptation data. We evaluate both the recurrent and the non-recurrent HAT in our experiments.

4. GPU Implementation

We implement training of LSTM-RNNs on GPU devices. Following [12], we use the truncated version of BPTT for model training. Each utterance is partitioned into short subsequences of 20 time steps. If the last subsequence of the utterance is shorter than 20 frames, we pad this last subsequence with pseudo frames up to 20 frames. These padding frames are excluded from gradients computation. Activations in the forward pass and parameter gradients in the backward pass are derived over the subsequences rather than entire utterances. Within the same utterance, the final LSTM states ($t = 20$) from the current subsequence are used as initialization for the next subsequence. To fully exploit the power of GPUs, our implementation processes subsequences from 20 utterances in parallel. Speed up is thus achieved by replacing matrix-vector multiplication over single frames with matrix-matrix multiplication over 20 frames at a time. To ensure training stability, the activations of memory cells $\mathbf{c}_t$ are clipped to the range of [-50, 50] in the forward pass.

5. Experiments

Our experiments are conducted on the Switchboard conversational telephone transcription task. We use Switchboard-1 Release 2 (LDC97S62) as the training set which contains over 300 hours of speech. For fast turnarounds, we also select 110 hours from the training set and create a lighter setup. Our test set is the Hub5'00 (LDC2002S09) set which consists of 20 conversations from Switchboard and 20 conversations from CallHome English. We report results on the Switchboard part and also on the entire test set. For decoding, a trigram language model (LM) is trained on the training transcripts. This LM is then interpolated with another trigram LM trained on the Fisher English Part 1 transcripts (LDC2004T19).

5.1. Experiments on the 110-Hour Setup

5.1.1. Baseline GMM-HMM Systems

We first report experiments on the 110-hour setup. The GMM-HMM systems are built with the standard Kaldi Switchboard recipe [29]. We train the initial ML model based on 39-dimensional MFCC (plus deltas and double deltas) features with per-speaker mean normalization. Then 7 frames of MFCCs are spliced and projected to 40 dimensions with linear discriminant analysis (LDA). A maximum likelihood linear transform (MLLT) is estimated on the LDA features and generates the LDA+MLLT model. Over the LDA+MLLT model, speaker adaptive training (SAT) is performed with one FMLLR transform [16] per speaker.

5.1.2. Baseline DNN and LSTM-RNN Models

Adaptation of DNNs and LSTM-RNNs can be naturally accomplished by using SA features as network inputs. We investigate three types of features: the SI filterbanks (FBanks), the SA filterbanks with vocal tract length normalization (VLTN), the SA FMLLRs. Both FBanks and VTLN-FBanks features are normalized with per-speaker mean and variance normalization. For each feature type, network inputs include 11 neighbouring frames (5 frames on each side of the center frame) which amount to 440 dimensions. The DNN has 5 hidden layers each of which contains 1200 neurons. It is initialized randomly by drawing the weights from a Gaussian distribution and biases from a uniform distribution. DNN fine-tuning optimize the cross-entropy (CE) objective using an exponentially decaying "newbob" learning rate schedule. Specifically, the learning rate starts from 0.008 and remains unchanged until the increase of the frame accuracy on a cross-validation set between two consecutive epochs falls below 0.5%. Then the learning rate is decayed by a factor of 0.5 at each of the subsequent epochs. The whole learning process terminates when the frame accuracy fails to improve by 0.2% between two successive epochs. A mini-batch size of 256 is adopted for stochastic gradient descent (SGD).

Our LSTM-RNN has 2 projected LSTM layers which are followed by the softmax layer. Each LSTM layer has 800 memory cells and 512 output units. Inputs to the architecture are single frames of FBanks or FMLLRs, without any context splicing. We also use the "newbob" learning rate schedule, with the difference of setting the initial learning rate to 0.00002. Table 1 shows the results of DNNs and LSTM-RNNs using different features. With both FBanks and VTLN-FBanks, the LSTM-RNN performs better than the DNN, demonstrating its advantage in acoustic modeling. However, the LSTM-RNN fails to outperform the DNN over the FMLLR front-end, indicating that FMLLRs are not suited for LSTM-RNN models. This is partly because FMLLRs are produced by splicing and transforming the original MFCCs. These complex transforms to some extent break the inherent temporal dependency between neighbouring frames. For LSTM-RNNs, the VTLN-FBank features give nice improvement over the FBank features. Therefore, VTLN is effective in adapting LSTM-RNNs on the front-end side.

Table 1: *Results (% WER) of the DNNs and LSTM-RNNs on the 110-hour set and using different features. The results are shown on the Hub5'00-SWB and Hub5'00 (in brackets) sets. M refers to million.*

Model	#Parameters	Feature	WER%
DNN	12M	FBank	20.2 (26.8)
	12M	VTLN-FBank	19.3 (25.6)
	12M	FMLLR	18.1 (24.3)
LSTM-RNN	8M	FBank	19.2 (26.1)
	8M	VTLN-FBank	18.3 (25.1)
	8M	FMLLR	18.0 (25.2)

5.1.3. Adaptation by Inserting SD Layers

We investigate the IFT and HAT methods presented in Section 3. The transform matrices in IFT and HAT are initialized to an identity matrix. On each testing speaker, we run 5 epochs of fine-tuning. The first epoch uses the learning rate of 0.00002 which is decayed by the factor of 0.5 in the following epochs. In Table 2, we first compare the variants of each method over the FBank features. For IFT, the component-specific version gives better results than having a component-uniform feature transform. This shows that distinguishing the LSTM components benefits model adaptation. Within the HAT method, the non-recurrent variant performs better than the more complex recurrent implementation. We think this is because under the recurrent HAT, learning of the transform requires back propagation through time steps. This may explode the parameter gradients, and thus overfit the adapted LSTM-RNN to the adaptation data quickly. At their best cases, both methods improve the LSTM-RNN baseline by 3.4% relative on the whole Hub5'00 set. When switching to the adaptive VTLN-FBank features, adaptation with IFT and HAT still generates gains. To this end, the IFT-adapted LSTM-RNN outperforms the SI LSTM-RNN by 6.5% relatively (24.4% vs 26.1%) on the entire test set.

Table 2: *Results (% WER) of the adapted LSTM-RNNs with the IFT and HAT methods described in Section 3.*

Feature	Model	WER%
FBank	LSTM-RNN	19.2 (26.1)
	+IFT (component-uniform)	18.8 (25.6)
	+IFT (component-specific)	**18.5** (**25.2**)
	+HAT (non-recurrent)	18.6 (25.2)
	+HAT (recurrent)	19.1 (25.7)
VTLN-FBank	LSTM-RNN	18.3 (25.1)
	+IFT (component-specific)	**18.0** (**24.4**)
	+HAT (non-recurrent)	18.2 (24.7)

5.1.4. Adaptation by Updating SI Models

The final category of adaptation is to update the SI LSTM-RNN (or part of it) on the adaptation data. Depending on which part of the model to be updated, adaptation in this section is divided into three cases. These cases involve updating the entire SI model, the input-to-component matrices $\mathbf{W}_{\cdot x}$, the projection-layer matrix $\mathbf{W}_{rh}$, respectively on the adaptation data. Table 3 shows the results corresponding to the three cases. Without loss of generality, only the FBank features are used as network inputs. We observe that updating the whole SI model is vulnerable to overfitting. Both the frame accuracy and the WER go up quickly on the adaptation data. This is why the adapted model performs even worse than the SI model. Adaptation in the latter two indeed improves the SI model. However, the gains are not as significant as that achieved by IFT and HAT. Updating the SI model is not an effective strategy to adapt LSTM-RNNs.

5.2. Experiments on the Complete 300-Hour Setup

Speaker adaptation of LSTM-RNNs is finally evaluated on the complete 300 hours of training data. We follow the same procedures as described in Section 5.1 to build the GMM and DNN models. The number of CD states in the GMM model increases from 4287 to 8929. The DNN model has 6 hidden layers each of which contains 2048 neurons. The DNN is initialized with restricted Boltzmann machines (RBMs) that are pretrained in a greedy layerwise fashion [30]. The LSTM-RNN has 2 projected LSTM layers. Each LSTM layer has 1024 memory cells and 512 output units. Under these settings, the DNN has 40 million parameters while the LSTM-RNN has 12 million. Table 4 shows the results of the baseline DNN and LSTM-RNN on the HUB'00 test set. On both FBank and VTLN-FBank features, the LSTM-RNN model outperforms the DNN model.

Table 3: *Results (% WER) of the adapted LSTM-RNNs by updating different parts of the SI model. We only examine the FBank features as the network inputs.*

Updated Part	WER%
The entire model	19.3 (26.0)
The input-to-component matrix $\mathbf{W}_{\cdot x}$	18.9 (25.5)
The projection layer $\mathbf{W}_{rm}$	19.0 (25.5)

For adaptation for LSTM-RNNs, we adopt the best configurations discovered in Section 5.1, i.e., the component-specific IFT and non-recurrent HAT. From Table 4, we can see that on each feature type, the adapted LSTM-RNN performs consistently better than the unadapted LSTM-RNN. However, the improvement we achieve here becomes less significant than the improvement on the 110-hour set. This is because with more training data, the LSTM-RNN model encodes richer speaker availability and thus generalizes better to unseen testing speakers, which decreases the efficacy of speaker adaptation. On the whole test set, the adapted LSTM-RNN achieves the best WER of 21.1%. This translates to 2.3% relative improvement over the SA LSTM-RNN (21.6%) with VTLN-Fbanks and 4.5% over the SI LSTM-RNN (22.1%) with FBanks.

Table 4: *Results (% WER) of the baseline DNN and LSTM-RNN, and the adapted LSTM-RNN. Adaptation is performed with the (component-specific) IFT and (non-recurrent) HAT methods.*

Feature	Model	WER%
FBank	DNN	16.9 (23.2)
	LSTM-RNN	15.8 (22.1)
	+IFT	15.5 (21.5)
	+HAT	15.8 (21.8)
VTLN-FBank	DNN	*** (***)
	LSTM-RNN	15.2 (21.6)
	+IFT	15.2 (21.1)
	+HAT	15.6 (21.5)

6. Conclusions and Future Work

In this paper, we have studied the problem of speaker adaptation for LSTM-RNN models. The effectiveness of the DNN adaptation techniques is evaluated for LSTM-RNNs. We propose two approaches, IFT and HAT, to implementing the idea of inserting SD layers. Our experiments with the Switchboard dataset show that adaptation with the proposed methods improves LSTM-RNN models by 3-4% relative. Applying speaker adaptive features enlarges the improvement of adaptation further to 6-7% relative. For the future work, we will study the incorporation of speaker i-vectors [31] for adaptation of LSTM-RNNs. Also, we are interested to port the SAT-DNN idea [25, 26] to LSTM-RNNs, and achieve SAT for the LSTM-RNN architecture.

7. References

[1] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 20, no. 1, pp. 30–42, 2012.

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

[3] F. Seide, G. Li, X. Chen, and D. Yu, "Feature engineering in context-dependent deep neural networks for conversational speech transcription," in *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*. IEEE, 2011, pp. 24–29.

[4] T. N. Sainath, A.-r. Mohamed, B. Kingsbury, and B. Ramabhadran, "Deep convolutional neural networks for lvcsr," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013, pp. 8614–8618.

[5] O. Abdel-Hamid, A.-R. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, vol. 22, no. 10, pp. 1533–1545, 2014.

[6] H. Soltau, G. Saon, and T. N. Sainath, "Joint training of convolutional and non-convolutional neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE, 2014, pp. 5572–5576.

[7] A. L. Maas, Q. V. Le, T. M. O'Neil, O. Vinyals, P. Nguyen, and A. Y. Ng, "Recurrent neural networks for noise reduction in robust asr." in *INTERSPEECH*. Citeseer, 2012.

[8] A. Graves, A.-R. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013, pp. 6645–6649.

[9] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *Neural Networks, IEEE Transactions on*, vol. 5, no. 2, pp. 157–166, 1994.

[10] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[11] A. Graves, N. Jaitly, and A.-R. Mohamed, "Hybrid speech recognition with deep bidirectional lstm," in *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*. IEEE, 2013, pp. 273–278.

[12] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," in *Proceedings of the Annual Conference of International Speech Communication Association (INTERSPEECH)*, 2014.

[13] H. Sak, O. Vinyals, G. Heigold, A. Senior, E. McDermott, R. Monga, and M. Mao, "Sequence discriminative distributed training of long short-term memory recurrent neural networks," in *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

[14] X. Li and X. Wu, "Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition," *arXiv preprint arXiv:1410.4281*, 2014.

[15] D. Yu, M. L. Seltzer, J. Li, J.-T. Huang, and F. Seide, "Feature learning in deep neural networks-studies on speech recognition tasks," *arXiv preprint arXiv:1301.3605*, 2013.

[16] M. J. Gales, "Maximum likelihood linear transformations for hmm-based speech recognition," *Computer speech & language*, vol. 12, no. 2, pp. 75–98, 1998.

[17] H. Liao, "Speaker adaptation of context dependent deep neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013, pp. 7947–7951.

[18] D. Yu, K. Yao, H. Su, G. Li, and F. Seide, "Kl-divergence regularized deep neural network adaptation for improved large vocabulary speech recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013, pp. 7893–7897.

[19] B. Li and K. C. Sim, "Comparison of discriminative input and output transformations for speaker adaptation in the hybrid nn/hmm systems," in *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[20] K. Yao, D. Yu, F. Seide, H. Su, L. Deng, and Y. Gong, "Adaptation of context-dependent deep neural networks for automatic speech recognition." in *SLT*, 2012, pp. 366–369.

[21] S. P. Rath, D. Povey, K. Veselỳ, and J. Cernockỳ, "Improved feature processing for deep neural networks." in *INTERSPEECH*, 2013, pp. 109–113.

[22] T. N. Sainath, B. Kingsbury, A.-r. Mohamed, G. E. Dahl, G. Saon, H. Soltau, T. Beran, A. Y. Aravkin, and B. Ramabhadran, "Improvements to deep convolutional neural networks for lvcsr," in *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*. IEEE, 2013, pp. 315–320.

[23] G. Saon, H. Soltau, D. Nahamoo, and M. Picheny, "Speaker adaptation of neural network acoustic models using i-vectors," in *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*. IEEE, 2013, pp. 55–59.

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

[25] Y. Miao, H. Zhang, and F. Metze, "Towards speaker adaptive training of deep neural network acoustic models," in *Proc. Interspeech*, 2014.

[26] Y. Miao, L. Jiang, H. Zhang, and F. Metze, "Improvements to speaker adaptive training of deep neural networks," 2014.

[27] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, "Learning precise timing with lstm recurrent networks," *The Journal of Machine Learning Research*, vol. 3, pp. 115–143, 2003.

[28] P. Swietojanski and S. Renals, "Learning hidden unit contributions for unsupervised speaker adaptation of neural network acoustic models."

[29] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlíček, Y. Qian, P. Schwarz *et al.*, "The kaldi speech recognition toolkit," 2011.

[30] G. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[31] N. Dehak, R. Dehak, P. Kenny, N. Brümmer, P. Ouellet, and P. Dumouchel, "Support vector machines versus fast scoring in the low-dimensional total variability space for speaker verification." in *Interspeech*, vol. 9, 2009, pp. 1559–1562.