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 \( X = (x_1, ..., x_T) \), a traditional recurrent layer iterates from \( t = 1 \) to \( T \) to compute the sequence of hidden states \( H = (h_1, ..., h_T) \) via the following equations:

\[
    h_t = \sigma(W_{zh}x_t + W_{hh}h_{t-1} + b_h)
\]

where \( W_{zh} \) is the input-to-hidden weight matrix, \( W_{hh} \) is the hidden-to-hidden (recurrent) weight matrix. In addition to the inputs \( x_t \), the hidden activations \( h_{t-1} \) from the previous time step are fed to influence the hidden outputs at the current time.
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 (input, output, forget) and memory cells activations sequentially to store the temporal states of the network. Additionally, multi-

![Figure 1: A memory block of LSTM.](image)

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 $h_t$ to a lower-dimensional vector $r_t$ via a linear transform $W_{rh}$. This projection layer is recurrent in that $r_t$ will be fed as inputs for the next time step. That is, $h_{t-1}$ in Equations 2(a−d) is replaced with $r_{t-1}$. In addition to reducing model parameters, adding the projection layer is found to generate better recognition accuracy.
a HAT layer is applied to the memory cell outputs \( r_t \). The transform matrix for the speaker \( s \) is \( \mathbf{M}_s \), with the size of \( |r_t| \times |r_t| \). The transformed activations \( \tilde{r}_t = \mathbf{M}_s r_t \) 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 \( \mathbf{M}_c \) can be applied in a recurrent manner. In the forward propagation, the transformed activations \( \tilde{r}_{t-1} \) are treated as \( \mathbf{h}_{t-1} \) in Equations 2(a-d) for computation of \( f_t, f_r, c_t \) and \( o_t \). In this case, the transform has to be applied sequentially frame by frame. In the back propagation, the gradients of \( \mathbf{M}_c \) need to be propagated through the time steps. Thus, BPTT is employed to optimize \( \mathbf{M}_c \) on the adaptation data. We evaluate both the recurrent and the non-recurrent variants 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 \( e_t \) 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 transcriptions. 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 (VTLN), 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-FBanks feature give nice improvement over the FBanks features. Therefore, VTLN is effective in adapting LSTM-RNNs on the front-end side.

<table>
<thead>
<tr>
<th>Model</th>
<th>#Parameters</th>
<th>Feature</th>
<th>WER%</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNN</td>
<td>12M</td>
<td>FBanks</td>
<td>20.2 (26.8)</td>
</tr>
<tr>
<td></td>
<td>12M</td>
<td>VTLN-FBanks</td>
<td>19.3 (25.3)</td>
</tr>
<tr>
<td></td>
<td>12M</td>
<td>FMLLR</td>
<td>18.1 (24.3)</td>
</tr>
<tr>
<td>LSTM-RNN</td>
<td>8M</td>
<td>FBanks</td>
<td>19.2 (26.1)</td>
</tr>
<tr>
<td></td>
<td>8M</td>
<td>VTLN-FBanks</td>
<td>18.3 (25.1)</td>
</tr>
<tr>
<td></td>
<td>8M</td>
<td>FMLLR</td>
<td>18.0 (25.2)</td>
</tr>
</tbody>
</table>

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.
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>
<thead>
<tr>
<th>Feature</th>
<th>Model</th>
<th>WER%</th>
</tr>
</thead>
<tbody>
<tr>
<td>FBank</td>
<td>LSTM-RNN</td>
<td>19.2 (26.1)</td>
</tr>
<tr>
<td></td>
<td>+IFT (component-uniform)</td>
<td>18.8 (25.6)</td>
</tr>
<tr>
<td></td>
<td>+IFT (component-specific)</td>
<td>18.5 (25.2)</td>
</tr>
<tr>
<td></td>
<td>+HAT (non-recurrent)</td>
<td>18.6 (25.2)</td>
</tr>
<tr>
<td></td>
<td>+HAT (recurrent)</td>
<td>19.1 (25.7)</td>
</tr>
<tr>
<td>VTLN-FBank</td>
<td>LSTM-RNN</td>
<td>18.3 (25.1)</td>
</tr>
<tr>
<td></td>
<td>+IFT (component-specific)</td>
<td>18.0 (24.4)</td>
</tr>
<tr>
<td></td>
<td>+HAT (non-recurrent)</td>
<td>18.2 (24.7)</td>
</tr>
</tbody>
</table>

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 $W_{xc}$, the projection-layer matrix $W_{rm}$, 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.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Model</th>
<th>WER%</th>
</tr>
</thead>
<tbody>
<tr>
<td>FBank</td>
<td>DNN</td>
<td>16.9 (23.2)</td>
</tr>
<tr>
<td></td>
<td>LSTM-RNN</td>
<td>15.8 (21.7)</td>
</tr>
<tr>
<td></td>
<td>+IFT</td>
<td>15.5 (21.5)</td>
</tr>
<tr>
<td></td>
<td>+HAT</td>
<td>15.8 (21.8)</td>
</tr>
<tr>
<td>VTLN-FBank</td>
<td>DNN</td>
<td><strong>15.2 (21.6)</strong></td>
</tr>
<tr>
<td></td>
<td>LSTM-RNN</td>
<td>15.2 (21.6)</td>
</tr>
<tr>
<td></td>
<td>+IFT</td>
<td>15.2 (21.1)</td>
</tr>
<tr>
<td></td>
<td>+HAT</td>
<td>15.6 (21.5)</td>
</tr>
</tbody>
</table>

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


