Making Deep Belief Networks Effective for Large Vocabulary Continuous Speech Recognition

Tara N. Sainath[1], Brian Kingsbury[1], Bhuvana Ramabhadran[1], Petr Fousek[2], Petr Novak[2], Abdel-rahman Mohamed[3]

[1]*IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, USA.*
[2]*IBM Research, Prague, Czech Republic.*
[3]*Department of Computer Science, University of Toronto, Canada.*
[1]{tsainath, bedk, bhuvana}@us.ibm.com,
{petr_fousek, petr.novak3}@cz.ibm.com, [3]asamir@cs.toronto.edu

***Abstract*—To date, there has been limited work in applying Deep Belief Networks (DBNs) for acoustic modeling in LVCSR tasks, with past work using standard speech features. However, a typical LVCSR system makes use of both feature and model-space speaker adaptation and discriminative training. This paper explores the performance of DBNs in a state-of-the-art LVCSR system, showing improvements over Multi-Layer Perceptrons (MLPs) and GMM/HMMs across a variety of features on an English Broadcast News task. In addition, we provide a recipe for data parallelization of DBN training, showing that data parallelization can provide linear speed-up in the number of machines, without impacting WER.**

I. Introduction

Hidden Markov Models (HMMs) continue to be widely employed for automatic speech recognition (ASR) tasks. Typically each HMM state models a speech frame using a Gaussian Mixture Model (GMM). While the HMM/GMM system continues to be a popular approach for ASR, this technique also has limitations. First, GMMs assume that the data obeys a specific distribution (i.e., Gaussian). Secondly, the GMM for each HMM state is trained using only the subset of total training data that aligned to that state, and thus data across states is not shared [1]. Therefore, training GMM parameters requires using techniques such as feature dimensionality reduction, which may throw away potentially valuable information.

Artificial neural networks (ANNs) have been explored as an alternative to GMMs [2], addressing the GMM problems discussed above. Perhaps the most popular ANN to date in speech recognition is the multi-layer perceptron (MLP), which organizes non-linear hidden units into layers and has full weight connectivity between adjacent layers. Significantly, in training, these weights are initialized with small random values. MLPs are used to estimate a set of state-based posterior probabilities, which are used in a variety of ways. In a hybrid system [2], these probabilities are used the output probabilities of an HMM. Alternatively, approaches such as TANDEM [3] and bottleneck features [4] derive a set of features from the MLP, which are then used as input features into a traditional GMM/HMM system. Hybrid systems typically perform slightly worse compared to a GMM/HMM system. While TANDEM and bottleneck methods have shown to offer improvements over GMM/HMM systems with standard features, they still use GMMs.

One problem with MLPs is that training is initialized from random weights and the objective function is non-convex, so training can get stuck in poor local optimum. While this has not been a serious hindrance for MLPs having one or two hidden layers that are trained using stochastic gradient descent [1], it poses a more serious challenge for deeper MLPs.

Recently, Restricted Boltzmann Machines (RBMs) [5] have been explored to pre-train weights of ANNs in an unsupervised fashion. This pre-training allows for a much better initial weight estimate, addressing the problems with MLP training. An ANN which is pre-trained with RBMs is generally referred to as a deep belief network (DBN). When DBNs are used in a hybrid architecture, they have shown improvements over a GMM/HMM system for various ASR tasks [1], [6].

Most DBN work in speech recognition has focused on small-vocabulary tasks (i.e., [1], [7]), with results on LVCSR tasks only having recently been explored [6], [8]. To date, DBN LVCSR research has used standard speech features, such as PLPs and MFCCs. However, a typical state-of-the-art LVCSR system [9], utilizes a specific recipe during acoustic model training which makes use of speaker adaptation (SA) and discriminative training (DT). The first goal of this paper is to explore the performance of DBNs against a state-of-the art LVCSR recognizer. Specifically, we compare the performance of DBNs to MLPs and GMM/HMMs with various features.

In addition, LVCSR research using DBNs has been limited because DBN training is performed serially, and therefore can be quite slow [6]. Graphical processing units (GPUs) are often used to accelerate training, though GPU cards are expensive relative to CPU cards. Motivated by work from the machine learning community in data parallelization of serially-trained algorithms on CPUs [10], [11], the second goal of this paper is to explore data parallelization of DBN training.

Experiments are conducted on a 50 hour English Broadcast News (BN) transcription task [12]. First, we show that the hybrid DBN system offers improvements over the MLP and GMM/HMM systems for a variety of feature spaces including SA and DT features. Second, we show that data parallelization

methods provide a speed-up linear in the number of parallel machines used, without impacting final WER.

The rest of this paper is organized as follows. Section II discusses the DBNs architectures explored in this paper. Parallelization of DBN training is presented in Section III. Section IV summarizes the experiments performed, while the results for different feature sets and using parallelization are presented in Sections V and VI respectively. Finally, Section VII concludes the paper and discusses future work.

II. Training Deep Belief Networks

In this section, we review the steps in training DBNs.

A. Restricted Boltzmann Machine

An RBM is a bipartite graph where visible units $\mathbf{v}$, representing observations, are connected via undirected weights to hidden units $\mathbf{h}$. Since speech observations are real-valued, we consider RBMs with Gaussian visible units and Bernoulli hidden units. Thus the energy of the Gaussian-Bernoulli configuration between $\mathbf{v}$ and $\mathbf{h}$ is defined by Equation 1, where $\theta = (\mathbf{w}, \mathbf{b}, \mathbf{a})$ defines the parameters of the RBM including weights $\mathbf{w}$, visible bias $\mathbf{b}$ and hidden bias $\mathbf{a}$.

$$E(\mathbf{v}, \mathbf{h}; \theta) = \sum_{i=1}^{\mathcal{V}} \frac{(v_i - b_i)^2}{2} - \sum_{i=1}^{\mathcal{V}} \sum_{j=1}^{\mathcal{H}} w_{ij} v_i h_j - \sum_{j=1}^{\mathcal{H}} a_j h_j \quad (1)$$

$w_{ij} \in \mathbf{w}$ represents the connected weight between visible unit i and hidden unit j, while $b_i \in \mathbf{b}$ and $a_j \in \mathbf{a}$ are their bias terms. $\mathcal{V}$ and $\mathcal{H}$ are the total visible and hidden units.

The probability that the model assigns to a visible vector $\mathbf{v}$ is given by Equation 2:

$$p(\mathbf{v}; \theta) = \frac{\sum_{\mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})}}{\sum_{\mathbf{u}} \sum_{\mathbf{h}} e^{-E(\mathbf{u}, \mathbf{h})}} \quad (2)$$

The RBM is trained via steepest ascent, where the gradient of the log likelihood of the training data with respect to the weights is given as follows:

$$\frac{\partial}{\partial w_{ij}} \log p(\mathbf{v}; \theta) = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model} \quad (3)$$

The first term in Equation 3 $\langle v_i h_j \rangle_{data}$ represents the conditional expectation of $\mathbf{h}$ given $\mathbf{v}$, and is relatively straightforward to compute. The second term $\langle v_i h_j \rangle_{model}$ represents the expectation under the distribution of the model over all joint configurations $(\mathbf{v}, \mathbf{h})$, and takes exponential time to compute exactly. Therefore, the Contrastive Divergence (CD) method [5] is often used to estimate this second term.

B. Deep Belief Networks

An RBM is used to learn the weights for one layer of a NN in an unsupervised fashion. Once the weights are learned, the outputs from this layer are used as input features to train another RBM that learns a higher level feature representation. This greedy, layer-wise training scheme is both fast and effective [5]. After a stack of RBMs has been trained, the layers are connected together to form what is referred to as a DBN. This process of learning DBN weights is known as pre-training. Supervised fine-tuning is then performed using the initial weights, as described next.

C. Fine-Tuning

During fine-tuning, each frame is labeled with a target class label. Given a DBN and a set of learned initial weights, fine-tuning is performed via back-propagation to retrain the weights such that the loss between the target and hypothesized class probabilities is minimized. In this paper, we explore two different loss functions. First, we use the cross-entropy loss [2], commonly used in NN training. One problem with cross-entropy is that it is frame-level discriminative, meaning that the loss is computed per input frame without information from neighboring frames. Because speech recognition is a sequence-level problem, frame-discriminative methods can cause certain frames to have over-emphasized probabilities, leading to potentially poor recognition hypotheses.

Alternatively, [12] proposes the use of a sequence classification criterion for fine-tuning. The benefit of this approach is that it is sequence-level discriminative, and more closely matches the overall objective of speech recognition. Experimentally, it has also been shown to outperform the cross-entropy criterion. One drawback of the sequence criterion is that it is computationally much more expensive to compute relative to the cross-entropy criterion.

III. Training DBNs for LVCSR

Most algorithms which are trained via stochastic gradient descent (i.e., NNs, RBMs), sequentially process the data. When processing hundreds of hours of data, common in many LVCSR tasks, it is computationally infeasible to process data sequentially. We motivate this further by the following example. Using the RBM training recipe in [1], [6] and [13], roughly 50 epochs are used for training the first layer and 25 epochs for subsequent layers. While this training recipe works well for tasks where the amount of training data is small, it is very slow for larger amounts of data [6]. For example, to train an 8-layer DBN with 1,024 hidden units per layer on 50 hours of data takes roughly 267 hours (11 days).

A recent area of machine learning has focused on distributed stochastic gradient training strategies. One popular class of algorithms, which has also been explored for parallelizing NN training [14], involves a distributed computation of the gradient, though [11] argues that the communication network cost of these methods is very high, and therefore these algorithms are computationally wasteful. Alternatively, [10] and [11] explore training separate models on subsets of the data in parallel, and combining models from different machines. Motivated by these previous distributed training methodologies, in this section we explore data parallelization for training RBMs.

A. Challenges

There are two difficulties with data parallelization for RBM training. Many distributed training algorithms [11] focus on parallelization methods for convex objective functions, and are

therefore able to prove convergence. However, since the RBM objective function is non-convex, proving convergence with data parallelization is difficult. Nevertheless past work [10] has successfully explored data parallelization for the the structured perceptron, also a non-convex loss function.

Secondly, in RBM training, the hidden units are sampled during the Constrastive Divergence (CD) step. This sampling is done stochastically and thus different hidden units are activated for different RBMs. This could make combining weights from RBMs trained on different subsets of the data difficult. However, in Section VI we show empirically that data used for training RBMs on each parallel machine is somewhat homogeneous and results in the activation of the same hidden units across the different RBMs. This justifies the combination of weights estimated on different machines.

B. Training Strategies

In this section, we describe various RBM training algorithms we explore to minimize training time.

1) Parallel Training - Mixing Per Epoch : The parameter mixing per epoch method, originally proposed in [10] for the structured perceptron, is outlined by Algorithm 1. In this method, we assume that training data T is divided into c disjoint subsets $T = \{T_1, T_2, \ldots, T_c\}$. An RBM is trained on each slave computer $i \in c$ using corresponding training subset T_i. Because the RBM loss function is non-convex, if different initial weights are provided to each slave, there is a higher chance that the final weights on each slave will be very different. To minimize these differences, we assume that the same initial weight is provided to each slave when RBM training starts. After one epoch of training data is completed on each slave computer, a master computer averages all weights together and recommunicates the weights back to all slave computer. This process continues for $NumEpochs$.

Algorithm 1 Parameter Mixing Per Epoch

Generate random initial weight w^0
for $n = 1 \rightarrow NumEpochs$ **do**
  Randomly split T into c sets $T = \{T_1, T_2, \ldots, T_c\}$
  for all $i \in \{1, \ldots, c\}$ **do**
    $w_i^n = RBM(T_i, w^{n-1})$
  end for
  Aggregate from all computers $w^n = \frac{1}{c}\sum_{i=1}^{c} w_i^n$
end for

2) Parallel Training - Mixing At End: One problem with the Mixing Per Epoch method is that each epoch of training requires that all slaves finish training before the weights can be updated. Thus, the training time at each epoch is always dictated by the slave which finishes last. An alternative method proposed in [11] looks at reducing master-slave communication time by averaging weight updates once per RBM training, though the problem is explored for convex objective functions.

This algorithm assumes that for each epoch, each computer $i \in c$ randomly samples a subset of training data T_i from all training data T, independent of the data randomly sampled on other computers. This implies that subsets belonging to two different computers T_i and T_j could contain overlapping data. [11] suggests that by randomly sampling the data per epoch, this proposed parallelization method is much faster but remains competitive in performance to the Mixing Per Epoch technique discussed in Section III-B1.

The mixing at end parallelization method is outlined in Algorithm 2. Specifically, an RBM is trained on each slave computer $i \in c$ using subset T_i. After one epoch of training is completed on each slave computer i, a subset of data $T_i \in T$ is again randomly sampled and used to estimate weights for the second epoch. After each slave runs $NumEpochs$ of training, the weights from all slaves are averaged together.

Algorithm 2 Parameter Mixing At End

Generate random initial weight w^0
for $n = 1 \rightarrow NumEpochs$ **do**
  for all $i \in \{1, \ldots, c\}$ **do**
    Randomly sample $T_i \in T$
    $w_i^n = RBM(T_i, w_i^{n-1})$
  end for
end for
Aggregate from all computers

$$w^{NumEpochs} = \frac{1}{c}\sum_{i=1}^{c} w_i^{NumEpochs}$$

3) Serial Training - Fixed Smaller Subset: To study if averaging weights from RBMs computed on different machines has any benefits, we perform a controlled experiment where we train a fixed subset of data serially on one computer, similar to [10]. At each epoch, the same fixed subset is used, and the size of the subset matches the size of the data presented to one computer during parallel training.

IV. Experiments

A. Corpora

The LVCSR experiments are conducted on an English broadcast news task [12]. The acoustic models are trained on 50 hours of data from the 1996 and 1997 English Broadcast News Speech Corpora. Results are reported on 101 speakers in the EARS Dev-04f set[1] and RT-04 test set. An LVCSR recipe described in [9] is used to create speaker-independent (SI), speaker-adapted (SAT) and discriminatively trained (DT) features. Specifically, given initial PLP features, first a set of SI features are created using Linear Discriminative Analysis (LDA). Further processing of LDA features is performed to create SAT features using vocal tract length normalization (VTLN) followed by feature space Maximum Likelihood Linear Regression (fMLLR). Finally, discriminative training is applied using the the Boosted Maximum Mutual Information (BMMI) or Minimum Phone Error (MPE) criterion.

B. DBN Training

All DBNs are pre-trained using the recipe outlined in [13]. More specifically, for the first layer, a Gaussian-Bernoulli

[1]One speaker has been removed from this Dev-04f set for faster decoding.

RBM is trained for 50 epochs. The feature input into the first layer uses a context of 9 frames, while 1,024 output features are computed. For all subsequent layers, Bernoulli-Bernoulli RBMs are trained for 25 epochs and contain 1,024 hidden units. For both RBM types, after 5 epochs of training, the momentum term is increased [13]. One key difference from [13] is that weight updates are performed per utterance, rather than a fixed mini-batch set of frames.

During fine-tuning, the final output layer is a softmax non-linearity with the number of output targets equal to context-dependent HMM states. Unless otherwise noted, we use 384 output targets in both DBN and MLP experiments, obtained by clustering the 2,220 states in the GMM/HMM system. Weight updates are again performed per utterance. After one pass through the data, loss is measured on a held-out set and the learning rate is annealed (i.e. reduced) by a factor of 2 if the held-out loss has grown from the previous iteration [2]. Unless otherwise noted, all DBN results are reported using the cross-entropy criterion, due to computational benefits. Experiments with the sequence criterion utilize the MPE objective function.

V. Results With LVCSR Features

In this section, we analyze the performance of DBNs with different types of features, as well as different numbers of layers and output targets. In addition, we compare DBN performance to both randomly initialized MLPs and GMM/HMMs.

A. DBN Performance With Different Feature Sets

Figure 1 shows the DBN performance on Dev-04f with different feature sets as the number of layers is increased. Note for all of these experiments, the number of output targets was 384. The figure shows that increasing the number of layers improves performance, though adding more than 6 layers does not allow for additional improvements, a trend also observed in other DBN tasks [1]. We hypothesize that after a certain number of layers (i.e., 6) most of the higher-order information in the data is captured by the DBN, and adding more layers does not help to capture additional information.

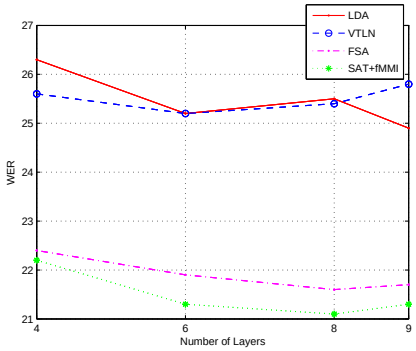

Fig. 1. WER For Different Features and Number of Layers

B. Further Analysis of SAT+fBMMI Features

Since the SAT+fBMMI features perform best, additional analysis is performed to see if WER can be further improved.

1) Increasing Output Targets: Figure 1 shows that the WER for SAT+fBMMI is within 1% absolute of the SAT features, while there is typically more than a 3% absolute difference between SAT and SAT+fBMMI for a GMM/HMM trained on 50 hours of BN [9]. An HMM captures variability in the data through thousands of context-dependent states, while a DBN typically chooses output targets equal to a much smaller number of states (i.e., 384). However, since these DBN target probabilities are in a hybrid HMM system during decoding, we explore increasing the number of output targets. Table I shows that as the number of targets increases, the WER improves up to 2,220, the number of CD states in the GMM/HMM system.

TABLE I
WER vs. Output Targets, Dev-04f

Number of Output Targets	WER
384	21.3
512	20.8
1,024	19.4
2,220	**18.5**

2) Training Criterion: Second, we compare using the cross-entropy and sequence criteria, as shown in Table II for the Dev-04f set. Even using the best performing fBMMI+SAT features, the sequence-level discrimination criterion offers improvements over the frame-discriminative cross-entropy criterion, demonstrating the importance of using an objective function which matches the sequence problem of speech recognition.

TABLE II
WER vs. Training Criterion, Dev-04f

Training Criteria	WER
Cross Entropy	18.5
Sequence	**17.0**

C. Comparison of DBNs to Other Methods

In this section, we explore the performance of DBNs, MLPs and GMM/HMMs. First, Table III compares DBNs to MLPs trained with the same architecture, namely 6 layers with 1,024 hidden units per layer. For the VTLN, LDA and SAT features, we use 384 output targets, while with the fBMMI features we explore 2,220 output targets. The table shows that pre-training improves the WER over the MLP for all feature sets.

TABLE III
Comparison of DBNs and MLPs

	Dev-04f		RT-04	
Feature	DBN	MLP	DBN	MLP
VTLN	**25.3**	26.0	**25.5**	26.5
LDA	**25.2**	26.5	**25.5**	26.7
SAT	**21.9**	22.5	**22.8**	23.7
SAT+fBMMI (2,220 targets)	**18.5**	21.9	**19.8**	23.6

Second, Table IV compares the performance of GMM/HMMs to DBNs for different features. DBN results are shown for 4 layers, where the number of DBN parameters matches the GMM/HMM system, as well as for 6 layers, which is the best performing DBN system. For both Dev-04f and RT-04, the DBN system with 4 layers offers improvements over the GMM/HMM system with the LDA and SAT features.

With FSA+fBMMI features, DBN performance is only better than the GMM/HMM system for 6 layers. After applying model-space BMMI discriminative training and speaker adaptation via MLLR to the GMM/HMM system, the DBN system using sequence criteria offers better performance.

TABLE IV
COMPARISON OF DBNS AND HMMS

	Dev-04f		
Feature	GMM/HMM	DBN - 4	DBN - 6
LDA	30.7	26.3	**25.2**
SAT	23.2	22.4	**21.9**
SAT+fBMMI	19.0 +BMMI: 18.0 +MLLR: 17.2	19.9	**18.5** +Seq.: **17.0**
	RT-04		
Feature	GMM/HMM	DBN - 4	DBN - 6
LDA	31.0	27.3	**25.5**
SAT	24.0	23.8	**22.8**
SAT+fBMMI	19.8 +BMMI: 18.5 +MLLR: 18.1	20.4 -	**19.7** +Seq.: **17.7**

VI. ANALYSIS OF DBN TRAINING

In this section, we explore the behavior of different techniques discussed in Section III for improving DBN training speed. Recall that the three training methods explored are Parallel - Mixing Per Epoch, Parallel - Mixing at End and Serial - Fixed Smaller Subset. For both parallelization methods, we compare the performance when 10, 25, and 50 parallel machines are used. This means that for each epoch, 50 hours of data is split across the appropriate number of machines. For example, when using 10 machines, 5 hours of data is used to train an RBM on each machine per epoch. To compare the serial training method to the parallel methods, the same amount of data on one parallel machine is used to train one epoch for serial training. For all training methods, we follow the same strategy used for full serial training (outlined in Section IV-B), including learning rates and number of epochs.

A. Parallelization Analysis

In this section, we empirically analyze the nature of the data and activations on different parallel computers, in order to justify the proposed RBM parallelization methods.

1) Exploration of Input Data: First, we explore if the data provided to each parallel computer is somewhat homogenous. If similar data is provided to each computer, then intuitively similar weights are learned on each machine. Since all data belonging to a specific speaker is somewhat homogenous, ideally we would like to show that each computer contains utterances from the same speaker. However, in the 50 Hour BN corpus, over 50% speakers contain two or less utterances.

Alternatively, we can look at clusters of similar speakers, and analyze if the distribution of number of utterances belonging to a cluster is similar across parallel computers. During the VTLN estimation process, speakers which behave similarly are transformed by the same warp factor (WF). We can calculate the percentage of utterances belonging to each WF, given the data assigned to specific parallel computer and epoch. Homogeneity can be approximated by seeing if this WF distribution is similar across different computers. Figure 2 plots the mean and standard deviation of the WF distribution across different computers after 50 epochs, the final set of weights learned from pre-training. The figure indicates that the WF distribution across different machines is very similar, as noted by the small standard deviation error bars, suggesting that data per parallel machine is somewhat homogenous.[2]

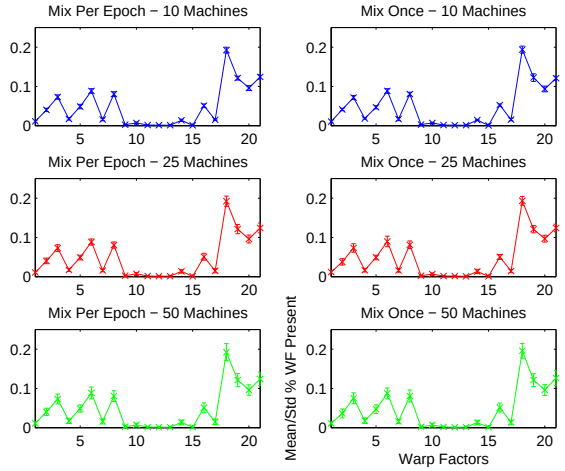

Fig. 2. Mean/Std of Warp Factor Distribution Across Machines

2) Activation Analysis: We can further justify weight averaging by analyzing if the activations calculated by RBMs trained on different machines are similar. Given the same data v_i, we calculate activations $p(h^1|v_i)$ and $p(h^2|v_i)$ using the RBM weights learned from machines 1 and 2 respectively. We define the root-mean-square error (RMSE) between activations computed from these machines by Equation 4, where h_j refers to hidden unit j. Note that since $p(h_j|v_i) \in \{0,1\}$, the maximum RMSE value between two machines is always 1.

$$RMSE^{1,2}(v_i) = \sqrt{\frac{\sum_{j=1}^{N}(p(h_j^1|v_i) - p(h_j^2|v_i))^2}{N}} \quad (4)$$

For every frame v_i, the $RMSE^{(k,l)}(v_i)$ is computed between all pairs of parallel computers k and l. This pair-wise $RMSE^{(k,l)}(v_i)$ is then computed for all frames $v_i \in V$ in a 5-hour held out set, and averaged together to produce a final $\overline{RMSE}$ per epoch, as given by Equation 5.

$$\overline{RMSE} = \text{average}\left[\sum_{i=1}^{V}\sum_{k=1}^{p}\sum_{l=k+1}^{p} RMSE^{(k,l)}(v_i))\right] \quad (5)$$

Table V shows the final $\overline{RMSE}$ value for the first layer of the DBN after 50 epochs of training. Two key trends can be observed. First, the RMSE when averaging per epoch is much lower than when averaging at the end, since averaging per epoch allows for variations in weights between different

[2]While this analysis is only done for VTLN features, other features such at SAT, SAT+fBMMI contain the VTLN stage in the feature creation.

computers to be smoothed out more frequently. Second, the RMSE for all parallelization methods is quite small compared to the maximum RMSE value of 1, justifying that weights can be averaged from different parallel machines.

TABLE V
FINAL $\overline{RMSE}$ FOR DIFFERENT PARALLELIZATION METHODS

# Parallel Machines	Mixing Per Epoch	Mixing At End
10	0.009	0.096
25	0.010	0.078
50	0.010	0.064

B. Performance

Now that we have justified the RBM parallelization methods, in this section we compare the performance of the parallel and serial methods. The experiments are performed with VTLN features. First, Figure 3 shows the total computation time of pre-training an 8 layer DBN as a function of the hours of training data used per computer and per epoch. Note that the total training time for full serial training is 267 hours for 8 layers, as stated earlier. The figure shows that training time linearly decreases as less data is used for training on each machine. The slowest method is the Parallel - Mix Per Epoch technique, where weights are averaged per epoch and the computation time is always driven by the slowest machine. Parallel - Mix At End is more than two times faster than Parallel - Mix Per Epoch since weights are averaged once. However, since averaging occurs once in Parallel - Mix at the end, it is slightly slower than the Serial - Fixed Subset method.

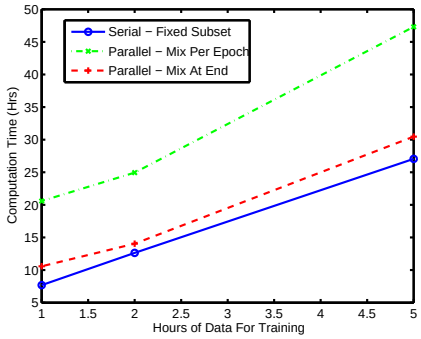

Fig. 3. Computation Time (Hours) for Different Parallelization Methods

Given that the Parallel - Mix at End and Serial - Fixed Subset techniques are much faster than Parallel - Mix Per Epoch, we focus on analyzing the WER of these faster methods. Table VI shows the WER as a function of hours of training data used per machine per epoch. The WER of 50-hour full serial training is given for reference.

TABLE VI
WER FOR DIFFERENT DBN TRAINING STRATEGIES, DEV-04F

Method	5 Hours	2 Hours	1 Hour
Serial - Fixed Subset	25.2	25.2	25.5
Parallel - Mix at End	25.0	25.2	25.4
Full Serial Training	25.3	25.3	25.3

Notice that across different cluster sizes, on average the Parallel - Mix At End method performs better than Serial - Fixed subset, particularly when a small amount of data (i.e., 1 hour) is used to train an RBM on each machine. A similar trend was also observed in [10] for data parallelization of the structured perceptron. The Parallel - Mix At End sees all of the data at each epoch across the parallel machines, whereas the Serial - Fixed Subset sees only a small fraction of the data. This allows for a more reliable estimate of the weights in the parallel method, resulting in better performance. Overall, both Figure 3 and Table VI illustrate that the Parallel - Mix At End method can provide linear speed-up in the number of machines compared to full serial training, without impacting WER.

VII. CONCLUSIONS

In this paper, we explored the use of DBNs on a large vocabulary task with a state-of-the-art LVCSR system. Specifically, we demonstrated that pre-training the weights of DBNs improves performance over MLPs and GMM/HMMs for a variety of feature spaces. In addition, we provide a recipe for the parallel training of DBNs, making them a computationally feasible option for LVCSR without impacting final WER. In the future, we would like to expand this DBN work to even larger tasks with thousands of hours of data.

ACKNOWLEDGEMENTS

Thank you to Hagen Soltau, George Saon and Stanley Chen for their contributions towards the IBM toolkit and recognizer utilized in this paper. In addition, thank you to Geoff Hinton and Karel Vesely for useful discussions related to DBNs.

REFERENCES

[1] A. Mohamed, G. E. Dahl, and G. E. Hinton, "Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition," *IEEE TSAP*, 2011.
[2] H. Bourlard and N. Morgan, *Connectionist Speech Recognition: A Hybrid Approach*. H. Bourlard and N. Morgan, 1993.
[3] D. P. W. Ellis, R. Singh, and S. Sivadas, "Tandem Acoustic Modeling in Large-Vocabulary Recognition," in *Proc. ICASSP*, 2001.
[4] F. Grezl and P. Fousek, "Optimizing Bottle-neck Features for LVCSR," in *Proc. ICASSP*, 2008.
[5] G. E. Hinton, S. Osindero, and Y. Teh, "A Fast Learning Algorithm for Deep Belief Nets," *Neural Computation*, vol. 18, pp. 1527–1554, 2006.
[6] G. Dahl, D. Yu, L. Deng, and A. Acero, "Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition," *IEEE TSAP*, 2011.
[7] G. E. Dahl, M. Ranzato, A. Mohamed, and G. E. Hinton, "Phone Recognition with the Mean-Covariance Restricted Boltzmann Machine," in *Proc. NIPS*, 2010.
[8] F. Seide, G. Li, and D. Yu, "Conversational Speech Transcription Using Context-Dependent Deep Neural Networks," in *to appear in Proc. Interspeech*, 2011.
[9] G. Saon, G. Zweig, B. Kingsbury, L. Mangu, and U. Chaudhari, "An Architecture for Rapid Decoding of Large Vocabulary Conversational Speech," in *Proc. Eurospeech*, 2003.
[10] R. McDonald, K. Hall, and G. Mann, "Distributed Training Strategies for the Structured Perceptron," in *Proc. HLT-NAACL*, 2010.
[11] M. A. Zinkevich, M. Weimer, A. Smola, and L. Li, "Parallelized Stochastic Gradient Descent," in *Proc. NIPS*, 2010.
[12] B. Kingsbury, "Lattice-Based Optimization of Sequence Classification Criteria for Neural-Network Acoustic Modeling," in *Proc. ICASSP*, 2009.
[13] G. E. Hinton, "A Practical Guide to Training Restricted Boltzmann Machines," Machine Learning Group, University of Toronto, Tech. Rep. 2010-003, 2010.
[14] K. Vesely, L. Burget, and F. Grezl, "Parallel Training of Neural Networks for Speech Recognition," in *Proc. Interspeech*, 2010.