Pipelined Back-Propagation for Context-Dependent Deep Neural Networks

Xie Chen[1,3], *Adam Eversole*[2], *Gang Li*[1], *Dong Yu*[2], *and Frank Seide*[1]

[1]Microsoft Research Asia, Beijing, P.R.C.
[2]Microsoft Research, Redmond, USA
[3]Department of Electronic Engineering, Tsinghua University, 10084 Beijing, P.R.C.
{adame,ganl,dongyu,fseide}@microsoft.com

Abstract

The Context-Dependent Deep-Neural-Network HMM, or CD-DNN-HMM, is a recently proposed acoustic-modeling technique for HMM-based speech recognition that can greatly outperform conventional Gaussian-mixture based HMMs. For example, a CD-DNN-HMM trained on the 2000h Fisher corpus achieves 14.4% word error rate on the Hub5'00-FSH speaker-independent phone-call transcription task, compared to 19.6% obtained by a state-of-the-art, conventional discriminatively trained GMM-based HMM.

That CD-DNN-HMM, however, took 59 days to train on a modern GPGPU—the immense computational cost of the minibatch based back-propagation (BP) training is a major roadblock. Unlike the familiar Baum-Welch training for conventional HMMs, BP cannot be efficiently parallelized across data.

In this paper we show that the *pipelined approximation to BP*, which parallelizes computation with respect to *layers*, is an efficient way of utilizing multiple GPGPU cards in a single server. Using 2 and 4 GPGPUs, we achieve a 1.9 and 3.3 times end-to-end speed-up, at parallelization efficiency of 0.95 and 0.82, respectively, at no loss of recognition accuracy.

Index Terms: speech recognition, deep neural networks, parallelization, GPGPU

1. Introduction

In this paper, we investigate *pipelined back-propagation* [1] for multi-GPGPU parallelization of training of deep neural networks, specifically *Context-Dependent Deep Neural Network Hidden Markov Models*, or CD-DNN-HMMs.

The CD-DNN-HMM [2, 3] is a recent acoustic-modeling technique for HMM-based speech recognition that can greatly outperform conventional Gaussian-mixture based HMMs. Like the ANN-HMMs of the 90's [4], CD-DNN-HMMs replace GMMs by an artificial neural network, but they differ in significantly increased network depth (7 or more hidden layers) and in that they *directly* model tied *context-dependent* states (senones) [5, 2] instead of factorizing the networks [6, 7]. [2] and [3] achieved relative error reductions of up to 33%.

However, a blocking factor for the wide-spread use of CD-DNN-HMMs are the large training times.

Conventional GMM-based HMM training is easily parallelizable. Its familiar (Extended) Baum-Welch training consists of statistics collection that can be parallelized easily over hundreds or even thousands of servers because all speech utterances are processed independently. At the end of a batch of typically hundreds of millions of frames, partial statistics from all servers are merged, and an updated model is distributed to the servers.

CD-DNN-HMMs, on the other hand, are trained with back-propagation, which involves a full model update after each minibatch of only a few hundred frames.[1] The required bandwidth makes parallelization across hundreds of servers prohibitive. However, a smaller-scale, cost-effective solution exists in the form of GPGPUs (general-purpose graphics processing units), which are used by most literature on DNNs.

In this paper, we aim to capitalize on servers with *multiple* GPU cards—two or four[2][3]—through the use of *pipelined back-propagation* [1], where different layers are computed on different GPUs in parallel, approximating model updates with delayed data. This raises two issues with CD-DNN-HMMs.

First, the error from the delay grows with network depths—will it still work for a large deep network of 7 hidden layers? Under which conditions? Secondly, a CD-DNN-HMM's output layer can be significantly larger than the other layers, making naïve pipelined BP inefficient. We address this by combining it with *model striping*, and show that in this combination, it is indeed efficient and effective for learning CD-DNN-HMMs.

2. The Context-Dependent Deep Neural Network HMM

A deep neural network (DNN) is a conventional multi-layer perceptron (MLP, [10]) with many hidden layers (such as 7 or 9). This section recaps the CD-DNN-HMM and its training by back-propagation, and discusses the question of minibatches.

2.1. Multi-Layer Perceptron

An MLP as used in this paper models the posterior probability $P_{\mathbf{s}|\mathbf{o}}(s|o)$ of a class s given an observation vector o, as a stack of $(L+1)$ layers of log-linear models. The first L layers, $\ell = 0...L-1$, model posterior probabilities of hidden binary vectors h^ℓ given input vectors v^ℓ, while the top layer L models the desired class posterior as

$$\begin{aligned} P^\ell_{\mathbf{h}|\mathbf{v}}(h^\ell|v^\ell) &= \prod_{j=1}^{N^\ell} \frac{e^{z_j^\ell(v^\ell)\cdot h_j^\ell}}{e^{z_j^\ell(v^\ell)\cdot 1} + e^{z_j^\ell(v^\ell)\cdot 0}} \ , \ 0 \le \ell < L \\ P^L_{\mathbf{s}|\mathbf{v}}(s|v^L) &= \frac{e^{z_s^L(v^L)}}{\sum_{s'} e^{z_{s'}^L(v^L)}} \quad = \quad \text{softmax}_s(z^L(v^L)) \\ z^\ell(v^\ell) &= (W^\ell)^T v^\ell + a^\ell \quad ; \quad v^\ell \overset{def}{=} E^{\ell-1}\{h^{\ell-1}\} \end{aligned}$$

[1]No successful attempts at full-batch back-propagation of our kind have so far been published.

[2]This is of course a far cry from parallelizing over hundreds of servers, but let us remember that only a few years back, we accepted that a typical model training would take a few weeks. So we consider cutting training time from 2 months to 2-3 weeks a very useful result.

[3]Servers with 2 and 4 GPU cards are fairly standard. 8 GPU cards is possible but as of yet uncommon.

with weight matrices W^ℓ and bias vectors a^ℓ, where h_j^ℓ and $z_j^\ell(v^\ell)$ are the j-th component of h^ℓ and $z^\ell(v^\ell)$, respectively. Full out-summation over all hidden variables, which is infeasible, is approximated by a "mean-field approximation" where the input v^ℓ to each hidden layer is taken as the expectation of the output vector h^ℓ of the previous layer.

Unlike earlier ANN-HMM systems, the classes s that are modelled are tied triphone states directly [2, 8, 5]. This is a critical factor for CD-DNN-HMMs in achieving its unusual accuracy improvements.

Lastly, for use with HMMs, state posteriors $P_{\mathbf{s}|\mathbf{o}}(s|o)$ are converted to *scaled likelihoods* by dividing by their prior [4].

2.2. Training by Back-Propagation

We train the MLPs according to the cross-entropy criterion

$$D \quad = \quad \sum_{t=1}^{T_{\text{corpus}}} \log P_{\mathbf{s}|\mathbf{o}}(s(t)|o(t)), \qquad (1)$$

by stochastic gradient descent

$$(W^\ell, a^\ell) \quad \leftarrow \quad (W^\ell, a^\ell) + \epsilon \frac{\partial D}{\partial (W^\ell, a^\ell)} \quad , \quad 0 \le \ell \le L,$$

with learning rate ϵ and gradients

$$\begin{aligned} \frac{\partial D}{\partial W^\ell} &= \sum_t v^\ell(t)\,(\omega^\ell(t)\,e^\ell(t))^T \,; \; \frac{\partial D}{\partial a^\ell} = \sum_t \omega^\ell(t)\,e^\ell(t) \quad (2) \\ e^L(t) &= (\log\text{softmax})'(z^L(v^L(t))) \\ e^{\ell-1}(t) &= W^\ell \cdot \omega^\ell(t) \cdot e^\ell(t) \qquad \text{for} \quad 0 \le \ell < L \\ \omega^\ell(t) &= \begin{cases} \text{diag}\left(\sigma'(z^\ell(v^\ell(t))\right) & \text{for} \quad 0 \le \ell < L \\ 1 & \text{else} \end{cases} \end{aligned}$$

with error signals $e^\ell(t)$, the component-wise derivatives $\sigma_j'(z) = \sigma_j(z) \cdot (1 - \sigma_j(z))$ and $(\log\text{softmax})_j'(z) = \delta_{s(t),j} - \text{softmax}_j(z)$, and Kronecker delta δ. This algorithm is well-known as *error back-propagation*[4] [12].

2.3. Minibatch Training—And why it is the Problem

Critically for this paper, due to the highly non-linear nature of the objective function, reasonable convergence can only be achieved by performing the above descent in *minibatches* of randomly sampled frames from the training corpus.[5] (This is often combined with a 1st-order "low-pass filter" to smooth the minibatch gradients, a technique called *momentum*.) Minibatch sizes of 1000 or less lead to best results in our experiments.

The need to use minibatches of only few hundred samples is the very root of the parallelization problem: Each minibatch requires a model update. Parallelization across data, common for conventional HMM training, would require prohibitive bandwidth: For a typical 7-hidden-layer CD-DNN-HMM in the order of 10^8 parameters, each minibatch would require the gathering/redistribution of 400 MB worth of gradients and another 400 MB of model parameters, per server. At about 500 ms/minibatch, we get close to the PCIe limit (about 6 GB/s).

[4]BP can easily get trapped in poor local optima for deep networks. This can be partially addressed by *pre-training*, either unsupervised using Deep Belief Networks [11] or as a supervised layer-building algorithm (Discriminative pre-training [9]). Pre-training is cheap compared to the main BP stage, and is thus not the subject of this paper.

[5]Several recent publications report the use of utterances as minibatches. We found, however, that that is not a good idea. Frames within an utterance are highly correlated. In our standard CD-DNN-HMM setup, utterance training led to a WER loss of 1.4 percentage points.

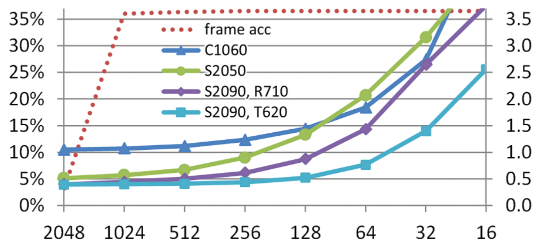

Figure 1: *Relative runtime for different minibatch sizes and GPU/server model types, and corresponding frame accuracy measured after seeing 12 hours of data.*[7]

2.4. Effect of Minibatch Size

Other than commonly found in literature, minibatch gradients in our notation are not averages over the frames of a minibatch, but the sum (i.e. we don't divide by the number of frames). This allows to experiment with different minibatch sizes without having to adjust the learning rate.

With this, we find that the minibatch size is bounded by two factors. The upper bound is set by the frequency of model update. Increasing the minibatch size means less model updates, which can be harmful especially in early iterations.

A lower bound seems not to exist w.r.t. accuracy: *Reducing* the minibatch size, even down to 1, does not hurt accuracy in our experiments. However, GPU computation becomes notably *less efficient* due to poor utilization of computation units. Thus, we consider minibatch training an optimzation of training time rather than, say, a means to obtain smoother gradients.

Figure 1 shows runtime and early frame accuracies for different minibatch sizes[6] after seeing 12 hours of data.[7] The "Goldilocks zone" is in the range 256 to 1024.

3. Parallelization Strategies

In this section, we will discuss three variants of parallelization and compare their bandwidth complexity. For that, let K denote the number of GPUs (for example, 4), T the size of a minibatch (like 1024), N the dimensions of all hidden layers[8], e.g. 2048, and J the output dimension (number of senones), e.g. 9304.

3.1. Partitioning the Data (Classic "Map-Reduce")

The classic map-reduce approach is to split the training data. While this is suitable for full-batch methods like Baum Welch, it is not for minibatches of a few hundred or a thousand frames: For each minibatch, it would require accumulation/redistribution of gradients/models of the dimension of the entire model to/from a "master" to the other $K-1$ GPUs. On the shared bus between GPGPUs, bandwidth per minibatch is of $\mathcal{O}(N \cdot (T + 2(L \cdot N + J)(K-1)))$. (A tree-structured communication architecture could reduce the $(K-1)$ to $\lceil \log_2 K \rceil$.)

3.2. Partitioning the Layer Parameters (Striping)

To avoid the prohibitive gathering and redistribution of model parameters, an alternative is to partition each layer's model parameters (W^ℓ and a^ℓ) into stripes and parallelize across these. Each GPU holds one out of K vertical stripes of each layer's parameters and gradients. In [13], this is called "node paral-

[6]In later epochs, minibatch size can be relaxed. In actual trainings, we limit it to 256 for the first 24h of data, and then relax it to, e.g., 1024.

[7]For the first 2.4 hours, minibatch size was capped at 256.

[8]For simplicity, we assume all hidden layer dimensions are the same. In our system, this is true except for the input layer, which is smaller.

Table 1: *Bandwidth order per minibatch for three parallelization strategies (T=minibatch size, N=hidden dimensions, J=output dimension, K=number of GPUs, L hidden layers) and actual values for L=7, T=1024, K=4, N=2048, J=9304.*

parallelization strategy	bandwidth $\mathcal{O}(\cdot)$	e.g. [MB]
data partioning	$N \cdot (T + 2(L \cdot N + J)(K - 1))$	1116
striped (W^ℓ, a^ℓ)	$N \cdot (K - 1) \cdot T \cdot (2L + 1)$	360
pipeline training	$N \cdot T \cdot (2K - 1)$	56

lelization," and we used this method in [3], where it achieved modest 25% speed-up on 2 GPUs.

Model updates (adding the gradient to the model parameters) happens only locally within each GPU: what gets communicated between GPUs are the frames. In forward propagation, each layer's input v^ℓ gets distributed to all GPUs, each of which computes a slice of the output vector $E^\ell\{h^\ell\}$. The slices are then distributed to all other GPUs for computing the next layer. In back-propagation, error vectors are parallelized as slices, but the resulting matrix products from each slice are partial sums that need to be further summed up. As a result, in both forward and back-propgation, each vector is transferred $K - 1$ times. Bandwidth is of $\mathcal{O}(N \cdot (K - 1) \cdot T \cdot (2L + 1))$ (with $\lceil \log_2 K \rceil$ possible with tree-structured communication).

3.3. Pipelined Parallelization Across Layers

Pipelined back-propagation [1] is an approximation that avoids the multiple copying of data vectors of the striping method, by distributing the *layers themselves* across GPUs (without striping) to form a pipeline. Data flows from GPU to GPU. All GPUs work simultaneously on the data they have. For $K < L+1$, multiple layers are grouped. Each vector travels twice per GPU, once forward and once for back-propagation. Bandwidth is $\mathcal{O}(N \cdot T \cdot (2K - 1))$. Because the minibatch data needed for a model update arrives at a delay due to the pipeline roundtrip, model updates can only use *delayed data*—and the deeper the network, the longer the delay.

Why might this work? Both "momentum" smoothing and minibatching already incur delays. Consider the last sample of a minibatch in regular minibatch training: Compared to a stochastic-gradient training where the model is updated after each sample, that last minibatch sample operates on a model that is out of date by $(T-1)$ frames. Yet, both trainings converge equally well if T is not too large (cf. section 2.4): The effect of delaying the update by $(T-1)$ frames on that last sample's gradient is small enough. If that is so, shouldn't pipelined BP also work, as long as long as its delay remains in the order of known working minibatch sizes of regular minibatch training?

That is indeed what we observe: Pipelined BP does not harm accuracy *if we also reduce the minibatch size*. The limiting factor is the GPU's inefficiency for small minibatches (Fig. 1). Experiments also indicate that the most critical delay is that of the top-most hidden layer; lower layers are more tolerant.

Lastly, naïve pipelined BP is not maximally efficient with CD-DNN-HMMs because of their significantly larger output layer. We find that this can be addressed by combining pipelined BP with striping.

4. Experimental Results

4.1. Setup and Baseline Results

We evaluate CD-DNN-HMMs on the task of speaker-independent, single-pass speech-to-text transcription using the

Table 2: *More training data and model parameters improve accuracy—but at a cost: Shown are WER and single-GPU training times (BP) across training setups (L=7 hidden layers. N=hidden dimensions, J=number of senones).*

	WER[%]		training
	Hub5'00	RT03S	time
setup	SWB	FSH	[days]
GMM BMMI, 309h SWBD-I	23.6	27.4	-
DNN N=2048, J=9304	17.1	19.8	5.4
+ J=32k (more senones)	16.4	19.5	11.5
+ DNN re-alignment	15.8	18.9	19.8
(rel. change from GMM)	(-33%)	(-31%)	
GMM BMMI, 2000h Fisher	21.7	23.0	-
+ fMPE	19.6	20.5	-
DNN N=2048, J=32k	14.9	16.0	39.1
+ N=3072 (larger hidden layers)	14.4	15.6	58.8
(rel. chg. from fMPE GMM)	(-27%)	(-24%)	

Switchboard-I and Fisher training sets [14]. The system uses 13-dimensional PLP features with mean-variance normalization and up to third-order derivatives, reduced to 39 dimensions by HLDA, and CART-tied crossword triphones.

The GMM-HMM baseline systems for 309h and 2000h use 9304 and 18004 senones with 40 and 72 Gaussian mixtures, respectively, trained discriminatively with BMMI and MMI+fMPE. The senone and mixture dimensions were optimzied for the Hub5'00 set using maximum-likelihood trained models. The CD-DNN-HMM system modifies this baseline only by replacing the GMMs with likelihoods derived from the MLP posteriors, while leaving everything else the same.

The primary test set is the FSH half of the 6.3h Spring 2003 NIST rich transcription set (RT03S), while the 1831-segment SWB part of the NIST 2000 Hub5 eval set was used for system development. The trigram language model was trained on the 2000h Fisher-corpus transcripts and interpolated with a written-text trigram. Test-set perplexity with the 58k dictionary is 84.

Table 2 shows that our best CD-DNN-HMM trained on 309 hours reduces word errors by about one third—a substantial improvement. (This result is slightly better than what we reported in [3] due to increased model size.) Compared to [3], we now also have results for training on the 2000h Fisher corpus. Compared to a BMMI-trained baseline GMM system, error reduction is also one third. When the baseline itself is further enhanced with fMPE[9], the reduction is still in the order of one fourth (from 19.6 and 20.5 to 14.4 and 15.6%, respectively. We believe that this is one of the best published results for single-pass speaker-independent recognition on this task.

4.2. Training Cost vs. Achievable Accuracy

The good results, especially on the large training set, come at considerable cost. Table 2 shows results for various training setups, and their training times on a NVidia Tesla S2070 GPU, illustrating the need for speeding up the training. 9304 senones is the optimal number for the GMM, but we see that for the DNN, increasing them to 32k yields a healthy WER drop on Hub5'00 (17.1 to 16.4%)—at more than double the training time (11.5 days instead of 5.4). DNN realignment provides another 0.6-point WER drop, at yet another near-doubling of time.

WER is further improved—over 3 percentage points for the RT03S-FSH set—by increasing the training data to the 2000h Fisher set and raising the hidden dimension to 3k, at another

[9]fMPE arguably is structurally similar to a neural-network layer.

Table 3: *Simulation of delayed update in pipelined training, for the worst-case data delay of one GPU per layer. 309h training, 7 hidden layers, word-error rates in [%] on Hub5'00.*

	minibatch size T				
minibatch size	64	128	256	512	1024
regular update	17.1	-	-	-	17.1
delayed update	17.0	17.0	17.2	17.4	[divergent]

three-fold increase of time. Compared to our starting point, the model size increases from 45 to 156 million parameters, yielding a relative 21% error reduction on RT03S-FSH (19.8 to 15.6%) and 16% on Hub5'00. For that last and best model, we had to face a training time of 59 days.

4.3. Effect of Delayed Update

Pipelined BP approximates BP by using delayed data for model updates. In this section, we look at whether the method is amenable to our deep networks of 7 hidden layers, assuming each model layer resides on a different GPU. With respect to the delay, this is the worst case. This is a simulation to measure the accuracy impact; we do not actually have such hardware.

Table 3 shows the impact of delayed updating on accuracy for increasing minibatch sizes T. For better convergence, each training was initialized with regular (non-delayed) BP for 24 hours of data. We see that delayed update causes a 0.1-point degradation for T=256. Here, the lowest layer gets updated with a delay of 13 minibatches, that is 3328 frames. For T=512, the degradation is 0.3 points (and there is a visible loss of training-set frame accuracy, from 56.1% to 55.5%). For T=1024, training diverges.

It is apparent that any speed-up from parallelization will be a trade-off between accuracy loss (too large T) vs. GPU efficiency (too small T). Our real hardware, however, has less GPUs (max. 4), and thus less delay, reducing this problem.

4.4. Parallelized Training on Multiple GPGPUs

Table 4 shows training runtimes using up to 4 GPUs (NVidia Tesla S2090) in a single server (Dell PowerEdge T620), measured for L=7 hidden layers, N=2048 hidden dimensions, and J=9304 senones. (The baseline runtime is about 30% better than Table 2 because of newer hardware and software improvements of device synchronization and data transfers).

Going to dual GPUs (K=2), we find that for *striping*, which in our earlier work [3] yielded a 25% speed-up on C1060 GPUs, we actually observe a *slowdown* on the newer S2090s.[10]

Pipelined BP, on the other hand, yields very good speed-ups of 1.7 to 1.9 on dual GPUs (e.g. reducing runtime from 61 to 33 minutes for T=512), at no accuracy loss despite its delayed-update approximation. We found, however, that for the largest minibatch size (T=1024), convergence could only be achieved by grouping the top hidden layer together with the output layer, such that the top hidden layer has no delayed update.

Going to K=4 GPUs barely helps: The overall speed-up remains below 2.2 (e.g. 61 vs. 29 min). This is because the output layer is 4.5 times larger (9304 $\times$ 2048 parameters) than the hidden layers (2048^2), and is thus the limiting bottleneck.

We solve this by combining pipelined BP with the striping method, which we apply only to the output layer. Two GPUs now jointly form the top stage of the pipeline, while the lower

[10]The striping runtimes in Table 4 are estimates, extrapolating from tests done with less optimized software that is about 15% slower.

Table 4: *Training runtimes in minutes per 24h of data for different parallelization configurations. [[·]] denotes divergence, and [·] denotes a WER loss $> 0.1\%$ points on the Hub5 set.*

	#GPU	minibatch size T		
parallelization method	K	256	512	1024
none (baseline)	1	68	61	59
striping[10]	2	-	67[10]	75[10]
pipeline training (0..6; 7)	2	40	34	[[33]]
vs. (0..5; 6..7)	2	36	33	31
vs. (0..2; 3..4; 5..6; 7)	4	32	29	[27]
pipeline + striped top layer	4	20	18	[[18]]

7 layers are pipelined on the other two GPUs. At no WER loss, our fastest pipelined system (T=512, 18 min) runs 3.3 times faster on 4 GPUs than our fastest single-GPU baseline (T=1024, 59 min), a parallelization efficiency of 82%.

5. Conclusion

We have taken a practical step towards parallelizing the back-propagation algorithm used for our deep-neural-network training. The focus was on making best use of multiple GPUs inside a single compute server. The familiar "map-reduce" over input utterances is not suitable for minibatch BP due to its immense bandwidth requirements.

We have shown that *pipelined back-propagation*, which updates models with *delayed data* and thereby allows to compute network layers concurrently, is effective and efficient for deep neural networks. This required to adjust the minibatch size, and, for 4 GPUs, to combine it with model striping to address the layer-size imbalance. At no accuracy loss, we achieved a 3.3 times speed-up on 4 GPUs—a 82% parallelization efficiency.

6. References

[1] Alain Pétrowski *et al.*, "Performance Analysis of a Pipelined Backpropagation Parallel Algorithm," IEEE Trans. Neural Networks, Vol. 4, No. 6, Nov. 1993.

[2] D. Yu, L. Deng, and G. Dahl, "Roles of Pretraining and Fine-Tuning in Context-Dependent DNN-HMMs for Real-World Speech Recognition," Proc. NIPS Workshop on Deep Learning and Unsupervised Feature Learning, Dec. 2010.

[3] F. Seide, G. Li, and D. Yu, "Conversational Speech Transcription Using Context-Dependent Deep Neural Networks," Proc. Interspeech, Florence, 2011.

[4] S. Renals, N. Morgan, H. Bourlard, M. Cohen, and H. Franco, "Connectionist Probability Estimators in HMM Speech Recognition," IEEE Trans. Speech and Audio Processing, January 1994.

[5] B. Kingsbury, "Lattice-based Optimization of Sequence Classification Criteria for Neural-Network Acoustic Modeling," Proc. ICASSP, Taipei, 2009.

[6] H. Franco *et al.*, "Context-Dependent Connectionist Probabilty Estimatation in a Hybrid Hidden Markov Model–Neural Net Speech Recognition System," Computer Speech and Language, Vol. 8, pp. 211–222, 1994.

[7] J. Fritsch *et al.*, "ACID/HNN: Clustering Hierarchies of Neural Networks for Context-Dependent Connectionist Acoustic Modeling," Proc. ICASSP, Seattle, 1998.

[8] G. Dahl, D. Yu, L. Deng, and A. Acero, "Context-Dependent Pre-Trained Deep Neural Networks for Large Vocabulary Speech Recognition", IEEE Trans. Speech and Audio Proc., Special Issue on Deep Learning for Speech and Lang. Processing, 2011.

[9] F. Seide, G. Li, X. Chen, and D. Yu, "Feature Engineering in Context-Dependent Deep Neural Networks for Conversational Speech Transcription," Proc. ASRU, Waikoloa Village, 2011.

[10] F. Rosenblatt, "Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms", Spartan Books, Wash. DC, 1961.

[11] G. Hinton, S. Osindero, and Y. Teh, "A Fast Learning Algorithm for Deep Belief Nets", Neural Computation, Vol. 18, 2006.

[12] D. Rumelhart, G. Hinton, and R. Williams, "Learning Representations By Back-Propagating Errors," Nature, Vol. 323, Oct. 1986.

[13] K. Vesely *et al.*, "Parallel Training of Neural Networks for Speech Recognition," Proc. Interspeech, Makuhari, 2010.

[14] J. Godfrey and E. Holliman, "Switchboard-1 Release 2," Linguistic Data Consortium, Philadelphia, 1997.