A COMPARISON OF TWO OPTIMIZATION TECHNIQUES FOR SEQUENCE DISCRIMINATIVE TRAINING OF DEEP NEURAL NETWORKS

George Saon and Hagen Soltau

IBM T. J. Watson Research Center, Yorktown Heights, NY 10598, USA

ABSTRACT

We compare two optimization methods for lattice-based sequence discriminative training of neural network acoustic models: distributed Hessian-free (DHF) and stochastic gradient descent (SGD). Our findings on two different LVCSR tasks suggest that SGD running on a single GPU machine achieves the best accuracy 2.5 times faster than DHF running on multiple non-GPU machines; however, DHF training achieves a higher accuracy at the end of the optimization. In addition, we present an improved modified forward-backward algorithm for computing lattice-based expected loss functions and gradients that results in a 34% speedup for SGD.

Index Terms— sequence discriminative training, distributed Hessian-free optimization, stochastic gradient descent, neural network acoustic models

1. INTRODUCTION

Nowadays, it is safe to say that deep neural networks have displaced Gaussian mixture models as the prevalent tool for acoustic modeling in ASR. Part of the reason is that discriminative training is inherent to DNNs whereas GMMs are generative classifiers. Maximum likelihood training of GMMs only guarantees optimality in distribution not for classification. To alleviate this, one major advance for GMM-HMMs was the introduction of feature-space and model-space (sequence) discriminative training using criteria like maximum mutual information (MMI) [1] and minimum phone error (MPE) [2] which resulted in accuracy gains in the range of 20%-30% on several tasks [3]. This has prompted researchers to revisit MMI and MPE training (or variants thereof) in the context of neural network acoustic modeling [4]. The criterion of choice for training neural nets for classification is cross-entropy. This objective improves frame classification performance but does not always improve sequence (i.e. word) classification performance because of the sequence constraints imposed by the HMM phone topologies, context decision tree, pronunciation dictionary and language model which are largely ignored.

Instead of optimizing frame accuracy, sequence training can be used to minimize sentence error (as for MMI and boosted MMI [5]), expected phone error (as for MPE), or expected HMM state error (as for state-based minimum Bayes risk, MBR [6]). Compared to frame-based discrimination, sequence training using the aforementioned criteria presents a number of additional challenges. First, the search spaces for the reference and competitor sequences have to be compactly encoded into "numerator" and "denominator" lattices which requires decoding the entire training data. Second, since the path posteriors in the lattices depend on the output of the neural network, the lattice-based objective functions and gradients have to be recomputed after every parameter update. This means that the training targets for the network *change* after every update. Third, because of the previous argument, frame randomization which is beneficial for cross-entropy training becomes computationally very expensive in the stochastic setting.

There are two distinct avenues for discriminative sequence training. The first one proposed in [7] and applied to lattice-based MBR loss minimization in [8] is to use a powerful second-order optimization technique called Hessian-free or truncated Newton which can be run in a distributed environment. The main idea of this method is to form a quadratic approximation of the objective function at every step and to minimize this quadratic using a conjugate gradient (CG) method with an early stopping criterion (hence the name truncated). Solving the CG subproblems requires repeatedly computing curvature-vector products which can also be done in a distributed fashion. The second avenue favored by [9, 10] is to use stochastic gradient descent on a single GPU machine for the optimization. The main difference between the two approaches is in the number and type of updates. For DHF, the model is updated only once for every pass through the training data whereas for SGD this is typically done after every training utterance. The DHF updates are very accurate because they take into account the true gradient and the local curvature of the objective function and can make rapid progress along directions of low curvature which are typically missed by steepest descent methods. In contrast, the gradients used for SGD are very noisy but the noise components tend to cancel out over time allowing the optimization to make progress along useful directions.

Compared to [8], we arrive at a different conclusion regarding the speed of SGD versus DHF training mainly due to the use of a GPU architecture for SGD, and confirm that DHF achieves a higher accuracy at the end of the optimization (a fact also observed in [11] for cross entropy training). Compared to [9, 10], we discuss DHF training and compared to [12, 13], we contrast DHF with utterance-level SGD. Lastly, we present an improved version of the modified forward-backward algorithm [14] for MBR loss and gradient computation that has not been published before.

2. MBR SEQUENCE TRAINING

In order to introduce some notations, recall that in a DNN-HMM hybrid model, the DNN computes the "pseudo log-likelihood" for HMM state s given acoustic frame $\mathbf{x}_t$ as

$$\log p(\mathbf{x}_t|s) = \log p(s|\mathbf{x}_t) - \log P(s) + \log p(\mathbf{x}_t) \qquad (1)$$

where $P(s)$ is the prior probability of s and $p(\mathbf{x}_t)$ is the data probability which does not depend on s and can be ignored. The state posterior probability $p(s|\mathbf{x}_t)$ is computed by a soft-max layer and has the expression

$$\log p(s|\mathbf{x}_t) = a_{st} - \log \left[\sum_{s'} exp(a_{s't}) \right] \qquad (2)$$

978-1-4799-2893-4/14/$31.00 ©2014 IEEE

with a_{st} denoting the input to the soft-max activation function for state s at time t. Note that, for the purpose of best path and lattice-based state posterior computations, the soft-max non-linearity can be discarded because $\log[\sum_{s'} exp(a_{s't})]$ does not depend on s, i.e. $\log p(s|\mathbf{x}_t) \approx a_{st}$.

2.1. Objective function and gradients

In this paper we limit the discussion to the state-based minimum Bayes risk (MBR) objective function given that it was shown to be superior to other discriminative criteria for DNN sequence training [8, 10]. Consider the training data formed of R observation sequences and associated reference HMM state sequences $\{(\mathbf{X}^1, S^1), \dots (\mathbf{X}^R, S^R)\}$ where $\mathbf{X}^r = \mathbf{x}_1^r \dots \mathbf{x}_{T_r}^r$ and $S^r = s_1^r \dots s_{T_r}^r$. The MBR objective function can be written as

$$\begin{aligned}\mathcal{L}(\theta) &= \sum_{r=1}^{R} \sum_{S} p_\theta(S|\mathbf{X}^r) H(S, S^r) \\ &= \sum_{r=1}^{R} \sum_{S} \frac{p_\theta^\kappa(\mathbf{X}^r|S) P(S) H(S, S^r)}{\sum_{S'} p_\theta^\kappa(\mathbf{X}^r|S') P(S')}\end{aligned} \tag{3}$$

with θ encoding the DNN parameters (weight matrices and biases) and κ denoting the acoustic scaling factor. $H(S, S^r)$ is the state frame error function and simply counts the number of incorrect states in S given the reference S^r. $p_\theta(S|\mathbf{X}^r)$ and $p_\theta^\kappa(\mathbf{X}^r|S)$ denote respectively the posterior probability of the state sequence S given the acoustics $\mathbf{X}^r$ and the scaled likelihood of the acoustics given the state sequence. $P(S)$ is the prior probability of S and encodes the HMM state transition probabilities, the pronunciation dictionary and the language model. The MBR state occupancies are given by the gradient of $\mathcal{L}(\theta)$ with respect to the outputs of the DNN for utterance r and time t (also called "error signal") [4]. According to (1)

$$\epsilon_t^r(s) \triangleq \frac{\partial \mathcal{L}(\theta)}{\partial \log p(s|\mathbf{x}_t)} = \frac{\partial \mathcal{L}(\theta)}{\partial \log p(\mathbf{x}_t|s)} \tag{4}$$

Using some notations inspired by [9], it can be shown that

$$\epsilon_t^r(s) = \kappa \gamma_t^r(s) \left[E_{S|\mathbf{X}^r, s_t=s}\{H(S, S^r)\} - E_{S|\mathbf{X}^r}\{H(S, S^r)\} \right] \tag{5}$$

$E_{S|\mathbf{X}^r, s_t=s}\{H(S, S^r)\}$ is the expected state error for utterance r computed over paths going through state s at time t, whereas $E_{S|\mathbf{X}^r}\{H(S, S^r)\}$ is the expected state error computed over all the paths given the observation sequence $\mathbf{X}^r$. $\gamma_t^r(s)$ is the state posterior probability of being in s at t conditioned on the acoustics $\mathbf{X}^r$. In the following, we will propose an efficient algorithm to compute these expectations and MBR state occupancies over a lattice.

2.2. Improved modified forward-backward algorithm

A word-level lattice is a directed graph defined by a set of nodes $\mathcal{N} = \{0, \dots, N-1\}$ and a set of arcs $\mathcal{A} = \{(i_0, j_0), \dots, (i_{Q-1}, j_{Q-1})\} \subset \mathcal{N} \times \mathcal{N}$. Each arc $(i_q, j_q) \in \mathcal{A}$ is defined by a source node $i_q \in \mathcal{N}$ and a destination node $j_q \in \mathcal{N}$ with $i_q < j_q$ meaning that the lattice is topologically sorted. The arcs are sorted in increasing source node order ($q < q' \Rightarrow i_q < i_{q'}$) so that, when processing the arcs in sequence from $0 \dots Q-1$, the lattice is traversed in topological order. By convention, the start state has index 0 and the set of final states is $\mathcal{F} \subset \mathcal{N}$. Moreover, each arc (i_q, j_q) contains a time-aligned subsequence of HMM state labels denoted by $s^q_{\tau(i_q)}, s^q_{\tau(i_q)+1}, \dots, s^q_{\tau(j_q)-1}$ where $\tau(i_q)$ and $\tau(j_q)$ are the times corresponding to the source and destination nodes, respectively. The arc score is defined as the product of scaled HMM state observation likelihoods along the arc times the LM score, i.e. $p(q) \triangleq \left[\prod_{t=\tau(i_q)}^{\tau(j_q)-1} p^\kappa(\mathbf{x}_t|s_t^q)\right] LM(q)$. The arc state error is the number of incorrect state labels along the arc, i.e. $H(q) \triangleq \sum_{t=\tau(i_q)}^{\tau(j_q)-1} [1 - \delta(s_t^q, \hat{s}_t)]$ where $\hat{s}_t$ is the reference HMM state at time t and δ is the Kronecker symbol.

Next, let us introduce the following quantities: α_i, β_i are the forward and backward likelihoods for node i and α'_i, β'_i are the forward and backward expected state errors of all the paths ending and beginning with node i, respectively. With these notations, the proposed algorithm for MBR HMM state occupancies is given below.

Algorithm 1 Improved modified forward-backward algorithm for MBR state occupancies.

Require: arc scores $p(q)$ and state errors $H(q)$
1: **for** node $i = 0 \dots N-1$ **do**
2: $\quad \alpha_i = \alpha'_i = \beta_i = \beta'_i = 0$
3: **end for**
4: $\alpha_0 = 1$
5: **for** arc $(i_q, j_q),\ q = 0 \dots Q-1$ **do**
6: $\quad \alpha_{j_q} \mathrel{+}= \alpha_{i_q} p(q)$
7: **end for**
8: **for** arc $(i_q, j_q),\ q = 0 \dots Q-1$ **do**
9: $\quad \alpha'_{j_q} \mathrel{+}= \frac{\alpha_{i_q} p(q) \left[\alpha'_{i_q} + H(q)\right]}{\alpha_{j_q}}$
10: **end for**
11: **for** node $i \in \mathcal{F}$ **do**
12: $\quad \beta_i = 1$
13: **end for**
14: **for** arc $(i_q, j_q),\ q = Q-1 \dots 0$ **do**
15: $\quad \beta_{i_q} \mathrel{+}= \beta_{j_q} p(q)$
16: **end for**
17: **for** arc $(i_q, j_q),\ q = Q-1 \dots 0$ **do**
18: $\quad \beta'_{i_q} \mathrel{+}= \frac{\beta_{j_q} p(q) \left[\beta'_{j_q} + H(q)\right]}{\beta_{i_q}}$
19: **end for**
20: $\ell = \overline{c} = 0$
21: **for** node $i \in \mathcal{F}$ **do**
22: $\quad \ell \mathrel{+}= \alpha_i$
23: $\quad \overline{c} \mathrel{+}= \alpha_i \alpha'_i$
24: **end for**
25: **for** arc $(i_q, j_q),\ q = 0 \dots Q-1$ **do**
26: $\quad \gamma_q = \frac{\alpha_{i_q} p(q) \beta_{j_q}}{\ell}$
27: $\quad c_q = \alpha'_{i_q} + H(q) + \beta'_{j_q}$
28: $\quad \epsilon_q = \gamma_q (c_q - \overline{c})$
29: **end for**
Ensure: $\epsilon_t(s) = \sum_{q: s_t^q = s} \epsilon_q$

Compared to the algorithm described in [14], α, α', β, β' are defined at the node level (instead of at the arc level). This allows for an efficient calculation of these quantities in $\mathcal{O}(Q)$ through sequential processing of the arcs in topological order (direct or reversed) in steps 5, 8, 14 and 17. In contrast, the complexity in [14] is $\mathcal{O}(QR)$ where R is the average number of predecessor or successor arcs. Additional memory and execution time are saved by avoiding the computation of predecessor and successor arcs in the first place.

2.3. Hessian-free optimization

While this algorithm has been described extensively in [7], in order to compare it with SGD, we remind the reader that the underlying theory of Newton-type algorithms is to approximate the loss function $\mathcal{L}$ around θ^k by a quadratic function. Specifically, Newton-type algorithms consider the second-order Taylor expansion

$$\mathcal{L}(\theta^k + \mathbf{p}) \approx q_{\theta^k}(\mathbf{p}) = \mathcal{L}(\theta^k) + \mathbf{b}_k^T \mathbf{p} + \mathbf{p}^T \mathbf{H}_k \mathbf{p} \qquad (6)$$

where $\mathbf{b}_k = \frac{\partial \mathcal{L}}{\partial \theta}\big|_{\theta^k}$ is the gradient and $H_k = \frac{\partial^2 \mathcal{L}}{\partial \theta^2}\big|_{\theta^k}$ represents the Hessian matrix both evaluated in θ^k. The Newton algorithm finds the new estimate

$$\theta^{k+1} = \theta^k + \alpha_k \mathbf{p}_k \qquad (7)$$

where $\mathbf{p}_k$ is obtained by minimizing q_{θ^k}. In order to avoid directions of negative curvature and to make the minimization more robust, the Hessian is replaced by a damped Gauss-Newton matrix of the form $\mathbf{G}_k + \lambda \mathbf{I}$ where the damping factor λ is updated using a Levenberg-Marquardt type of heuristic [7]. The minimization of (6) is carried out using conjugate gradient (CG) which has two advantages. First, CG makes rapid progress after few iterations meaning that the minimization can be stopped early (hence the name truncated Newton). Second, CG can be formulated solely in terms of matrix-vector products which means that the Hessian matrix does not have to be computed and stored explicitly; only its effect on a vector has to be known (hence the name Hessian-free). The computation of the gradient and the curvature-vector products $(\mathbf{G}_k + \lambda \mathbf{I})\mathbf{p}$ required for the CG minimization can be parallelized and sped up as in [13]. Lastly, the stepsize α_k in (7) is chosen by performing a linesearch along the direction $\mathbf{p}_k$. Generally, α_k is less than 1 because $\mathbf{p}_k$ only minimizes $q_{\theta^k}(\mathbf{p})$ not necessarily $\mathcal{L}(\theta^k + \mathbf{p})$.

2.4. Stochastic gradient descent

In contrast to Hessian-free, the update for SGD is remarkably simple

$$\theta^{k+1} = \theta^k - \alpha_k \frac{\partial \mathcal{L}_{r_k}}{\partial \theta}\Big|_{\theta^k} \qquad (8)$$

where $\mathcal{L}_{r_k}$ is the loss function corresponding to a randomly selected utterance $r_k \in \{1, \ldots, R\}$ at iteration k. Typically, α_k is held constant over the course of one epoch. Compared to (7), here the model is updated after every single utterance as opposed to once per training data sweep.

3. EXPERIMENTS AND RESULTS

3.1. Experiments on English CTS 300 h

The first set of experiments was carried out on a 300 hour subset of the Switchboard English conversational telephone speech task. We report results on the SWB part of the Hub5 2000 testset which contains 2.1 hours of audio and 21.4K words. For comparison, results on this task and testset can also be found in [15, 8, 10, 16].

Acoustic feature processing consists in extracting 13-dimensional PLP cepstra every 10ms which are mean and variance normalized on a per speaker basis. The cepstra are warped with VTLN and every 9 consecutive cepstral frames are concatenated and projected down to 40 dimensions using LDA. The resulting features are decorrelated by means of a global semi-tied covariance transform. The LDA/STC features are further transformed with feature-space MLLR (FMLLR) per conversation side for both training and test. The FMLLR transforms are estimated using a baseline GMM-HMM system. As suggested in [15], the input to the DNNs is formed by concatenating 11 consecutive FMLLR frames.

All nets have 6 hidden layers with sigmoid activation functions: the first 5 with 2048 units and the last one with 256 units for parameter reduction and faster training time [17]. The output layer has 9300 softmax units that correspond to the context-dependent HMM states obtained by growing a phonetic decision tree with pentaphone crossword context. Following the recipe outlined in [15], the training data is fully randomized at the frame level within a window of 25 hours and we trained the nets with stochastic gradient descent on minibatches of 250 frames and a cross-entropy criterion. Prior to the cross-entropy training of the full network, we used layerwise discriminative pretraining [15] by running one cross-entropy sweep over the training data for the intermediate networks obtained by adding one hidden layer at a time. Cross-entropy training converged after 14 iterations and the final networks are used to decode the training data with a unigram LM and dump numerator and denominator lattices. The resulting denominator lattices have a link density of 8690 arcs per second and were generated according to [18].

For SGD training, we used utterance randomization and an initial step size of 1e-5 without acoustic weight scaling (same as [10]). After the first iteration, the step size was reduced to 1e-6 and kept constant for the subsequent iterations. Lattice-based processing was done on the CPU and the backpropagation for the neural net was done on a Tesla K10 GPU card. In order to minimize the effect of I/O, we found it beneficial to group the features and lattices for the randomized utterances into 100 batches of approximately 1900 utterances each and to read and process one random batch at a time.

	CPU Time (secs)	GPU Time (secs)
Algorithm [14]	1235	659
Algorithm 1	917	434
Speedup	25%	34%

Table 1. SGD processing times (excluding I/O) and speedups for 3625 utterances with two algorithms for MBR state occupancy calculation when the backpropagation is run on the CPU versus GPU.

In Table 1 we show the SGD processing times (excluding I/O) for 3625 utterances (5 hours of audio) when the neural net backpropagation is done on the CPU versus GPU and when the MBR occupancies are computed with Algorithm 1 versus [14]. We observe that the use of Algorithm 1 for the modifed forward-backward passes over the lattice results in a 34% speedup when the backpropagation is done on the GPU. A 25% speedup is observed when the entire computation is done on the CPU (which is the case for DHF). Additional speedups for SGD could be obtained by moving the MBR computation to the GPU card as done in [9] although this has not been investigated. One sequence SGD pass over the entire training data takes 11.9 hours (versus 6 hours for a cross-entropy pass). In Figure 1, we compare distributed Hessian-free training running on a cluster of 40 8-core Intel Xeon 3Ghz compute nodes with stochastic gradient descent running on a single 8-core Intel Xeon 2.7Ghz machine with a Tesla K10 GPU card. As can be seen, DHF obtains the best performance of 12.5% after approximately 6.4 days of processing time (corresponding to 24 passes through the data) whereas SGD obtains the lowest error rate of 12.7% after only 2 days of compu-

tation (corresponding to 4 passes through the data). The same error rate is achieved by DHF after approximately 5 days. Also shown is the fact that running additional DHF iterations on top of the best SGD models does not improve the performance further.

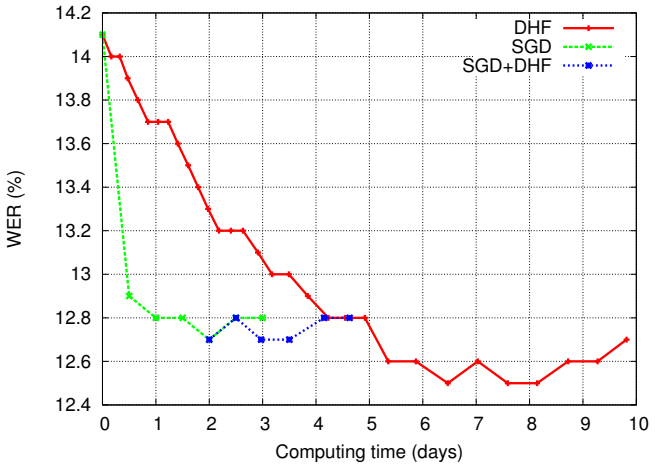

Fig. 1. Word error rates on Hub5'00 versus computation time for MBR sequence training with DHF and SGD on English CTS.

3.2. Experiments on RATS Levantine 300 h

The second set of experiments was carried out on noisy Levantine speech for the RATS (Robust Automatic Transcription of Speech) task which is a DARPA program aimed at performing speech activity detection, language and speaker identification and keyword search in audio documents which are sent over highly degraded radio communication channels. The training data for ASR was obtained by retransmitting "clean" Callhome-type telephone conversations in Levantine Arabic using 8 different sender and receiver pairs. The resulting channels, labeled A-H, exhibit highly variable distortions with channel G being the closest to the original data. We report results on the retransmitted Levantine Dev'04 testset. Front-end processing is identical to the English CTS experiments i.e. PLP VTL-warped cepstra followed by LDA, STC and speaker-based FMLLR, the latter being applied at both training and test time. The input to the neural nets is formed by ± 5 40-dimensional FMLLR frames.

The networks have 6 hidden layers with 2048 sigmoid units each and an output layer with 7000 softmax units corresponding to the context-dependent HMM states obtained by growing a decision tree with phonetic questions in a ± 2 phones context window. The baseline DNNs are grown one layer at the time with discriminative pretraining and trained until convergence with cross-entropy on frame-randomized minibatches of size 250. The resulting networks are used to compute numerator and denominator lattices on the training data. The denominator lattices have a link density of 4732 arcs per second. In Figure 2, we compare distributed Hessian-free training running on a cluster of 48 compute nodes with stochastic gradient descent running on a single GPU machine. We observe that DHF obtains the best performance of 37.8% after approximately 17.6 days of processing time (corresponding to 27 iterations, not all of them displayed) whereas SGD obtains the lowest error rate of 38.2% after 3.5 days of computation (corresponding to 5 passes through the data). The same error rate is achieved by DHF after approximately 10 days. Despite similar amounts of audio and smaller denominator lattices compared to the Switchboard task, the computation times for the RATS task are longer because the DNN weight matrices between the last hidden layers and the output layers are not factorized which makes the forward and backward passes more costly. For example, one SGD iteration takes 17 hours as opposed to 11.9 hours for the Switchboard task.

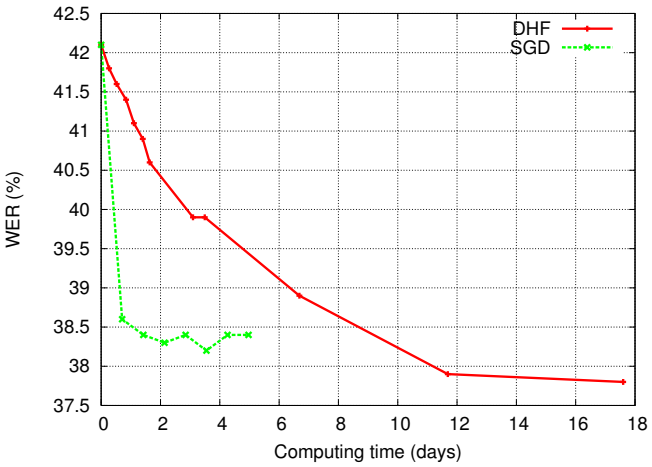

Fig. 2. Word error rates on Dev'04 channel G versus computation time for MBR sequence training with DHF and SGD on RATS.

Lastly, in Table 2 we compare the performance of DHF and SGD on all the channels and observe that the improvement obtained with DHF is 0.4% absolute higher on average than with SGD.

	A	B	C	D	E	F	G	H
CE	50.3	72.9	59.4	55.0	75.4	60.1	42.1	85.8
DHF	46.7	72.4	58.0	51.8	79.1	57.0	37.8	85.5
SGD	47.3	73.5	58.2	52.0	79.0	57.6	38.2	85.6

Table 2. Word error rates per channel on Dev'04 for CE, DHF and SGD on RATS Levantine.

4. DISCUSSION

The previous results suggest that a mixed strategy which combines the speed of SGD with the accuracy of DHF should be investigated. According to Figure 1, switching to DHF after SGD has converged does not seem to work because the DHF iterations become expensive at that point which nullifies the speed advantage of SGD. Perhaps a better approach would be to use curvature information for SGD as proposed in [19]. Alternatively, one can opt for a stochastic form of DHF as done recently in [12] where the training data is split into several batches and the model is updated with DHF after every batch. As GPU architectures become more widespread, we currently investigate the possibility of running DHF on a cluster of GPU machines which should make the training speed comparable to SGD.

5. ACKNOWLEDGMENT

The authors wish to thank Brian Kingsbury for useful discussions. This work was supported in part by Contract No. D11PC20192 DOI/NBC under the RATS program. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

6. REFERENCES

[1] L. R. Bahl, P. F. Brown, P. V. de Souza, and R. L. Mercer, "Maximum mutual information estimation of hidden Markov model parameters for speech recognition," in *Proc. of ICASSP*, 1986, pp. 49–52.

[2] D. Povey and P. C. Woodland, "Minimum phone error and I-smoothing for improved discriminative training," in *Proc. of ICASSP*, 2002, pp. 105–108.

[3] G. Saon and J.-T. Chien, "Large vocabulary continuous speech recognition systems: a look at some recent advances," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 18–33, 2012.

[4] B. Kingsbury, "Lattice-based optimization of sequence classification criteria for neural-network acoustic modeling," in *Proc. of ICASSP*, 2009.

[5] D. Povey, D. Kanevsky, B. Kingsbury, B. Ramabhadran, G. Saon, and K. Visweswariah, "Boosted MMI for model and feature-space discriminative training," in *Proc. of ICASSP*, 2008, pp. 4057–4060.

[6] J. Kaiser, B. Horvat, and Z. Kacic, "A novel loss function for the overall risk criterion based discriminative training of HMM models," in *Proc. ICSLP*, 2000.

[7] J. Martens, "Deep learning via Hessian-free optimization," in *Proc. of ICML*, 2010.

[8] B. Kingsbury, T. Sainath, and H. Soltau, "Scalable minimum Bayes risk training of deep neural network acoustic models using distributed Hessian-free optimization," in *Proc. Interspeech*, 2012.

[9] H. Su, G. Li, D. Yu, and F. Seide, "Error back propagation for sequence training of context-dependent deep networks for conversational speech transcription," in *Proc. of ICASSP*, 2013.

[10] K. Vesely, A. Ghoshal, L. Burget, and D. Povey, "Sequence-discriminative training of deep neural networks," in *Proc. Interspeech*, 2013.

[11] S. Wiesler, J. Li, and J. Xue, "Investigations on Hessian-free optimization for cross-entropy training of deep neural networks," in *Proc. Interspeech*, 2013.

[12] P. Dognin and V. Goel, "Combining stochastic average gradient and Hessian-free optimization for sequence training of deep neural networks," in *Proc. ASRU*, 2013.

[13] T. Sainath, L. Horesh, B. Kingsbury, A. Aravkin, and B. Ramabhadran, "Improving training time of Hessian-free optimization of deep neural networks using preconditioning and sampling," in *Proc. ASRU*, 2013.

[14] D. Povey, *Discriminative Training for Large Voculabulary Speech Recognition*, Ph.D. thesis, Cambridge University, 2004.

[15] F. Seide, G. Li, X. Chien, and D. Yu, "Feature engineering in context-dependent deep neural networks for conversational speech transcription," in *Proc. ASRU*, 2011.

[16] G. Saon, H. Soltau, D. Nahamoo, and M. Picheny, "Speaker adaptation of neural network acoustic models using i-vectors," in *Proc. ASRU*, 2013.

[17] T. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran, "Low-rank matrix factorization for deep neural network training with high-dimensional output targets," in *Proc. of ICASSP*, 2013.

[18] G. Saon, D. Povey, and G. Zweig, "Anatomy of an extremely fast LVCSR decoder," in *Proc. Interspeech*, 2005.

[19] R. Kiros, "Training neural networks with stochastic Hessian-free optimization," in *Proc. of Int. Conf. on Learning Representations*, 2013.