Accelerated Parallelizable Neural Network Learning Algorithm for Speech Recognition

Dong Yu and Li Deng

Microsoft Research, Redmond, WA, 98052, USA

{dongyu, deng}@microsoft.com

Abstract

We describe a set of novel, batch-mode algorithms we developed recently as one key component in scalable, deep neural network based speech recognition. The essence of these algorithms is to structure the single-hidden-layer neural network so that the upper-layer's weights can be written as a deterministic function of the lower-layer's weights. This structure is effectively exploited during training by plugging in the deterministic function to the least square error objective function while calculating the gradients. Accelerating techniques are further exploited to make the weight updates move along the most promising directions. The experiments on TIMIT frame-level phone and phone-state classification show strong results. In particular, the error rate is strictly monotonically dropping as the mini-batch size increases. This demonstrates the potential for the proposed batch-mode algorithms in large scale speech recognition since they are easily parallelizable across computers.

Index Terms: neural network, scalability, structure, constraints, FISTA acceleration, optimization, pseudo-inverse, weighted LSE, phone state classification, speech recognition, deep learning

1. Introduction

Deep learning is a promising direction for automatic speech recognition (ASR) and other areas of information processing, as recently surveyed in [1] and demonstrated on large vocabulary and other ASR tasks [2][3][4][5][6]. For example, on the 309-hour switchboard task, we achieved greater than 30% relative word error rate (WER) reduction over the discriminatively trained Gaussian mixture model (GMM) hidden Markov models (HMMs) using our recently proposed context-dependent deep-neural-network HMM (CD-DNN-HMM) [6]. Unfortunately, scaling CD-DNN-HMMs further to thousands of hours of speech is difficult since the prevailing learning algorithm is inherently sequential and performs best and converges fastest if the mini-batch size is in the range of 100-1000. This prevents meaningful parallelization across different computers and GPGPUs are typically exploited to accelerate the learning process. The work presented in this paper offers a potential solution to the scalability problem of learning neural networks.

The core of this paper is a set of accelerated batch-mode algorithms that exploit structures of single-hidden-layer neural networks (SHLNNs). These SHLNNs can be stacked to form a deep neural network [12]. The batch-mode nature of our proposed algorithms enables easy parallelization across many machines since as demonstrated in the experiments that better accuracy is achieved consistently as the mini-batch size is increased.

The organization of this paper is as follows. In Section 2, we present the basic neural network model with a single hidden layer and a linear output layer. We develop learning algorithms for this model by taking advantage of the structural constraints among the neural network weights, as described in detail in Section 3. Enhancement of the algorithms is made by using two accelerating techniques, as presented in Section 4. In Section 5 we provide the results from a series of experiments to demonstrate the characteristics of the enhanced algorithms.

2. Basic Model

Given the set of input vectors $\mathbf{X} = [\boldsymbol{x}_1, \cdots, \boldsymbol{x}_i, \cdots, \boldsymbol{x}_N]$, in which each vector $\boldsymbol{x}_i = [x_{1i}, \cdots, x_{ji}, \cdots, x_{Di}]^T$, where D is the dimension of the input vector and N is the total number of training samples. Denote L the number of hidden units and C the dimension of the output vector, the output of the SHLNN is $\boldsymbol{y}_i = \mathbf{U}^T \boldsymbol{h}_i$, where $\boldsymbol{h}_i = \sigma(\mathbf{W}^T \boldsymbol{x}_i)$ is the hidden layer output, $\mathbf{U}$ is an $L \times C$ weight matrix at the upper layer, $\mathbf{W}$ is a $D \times L$ weight matrix at the lower layer, and $\sigma(x) = 1/(1 + e^{-x})$ is the sigmoid function. Note that the bias terms are implicitly represented in the above formulation if $\boldsymbol{x}_i$ and $\boldsymbol{h}_i$ are augmented with 1's.

Given the target vectors $\mathbf{T} = [\boldsymbol{t}_1, \cdots, \boldsymbol{t}_i, \cdots, \boldsymbol{t}_N]$, where each target $\boldsymbol{t}_i = [t_{1i}, \cdots, t_{ji}, \cdots, t_{Ci}]^T$, the parameters $\mathbf{U}$ and $\mathbf{W}$ are learned to minimize the square error

$$\mathrm{E} = \|\mathbf{Y} - \mathbf{T}\|^2 = \mathrm{Tr}[(\mathbf{Y} - \mathbf{T})(\mathbf{Y} - \mathbf{T})^{\mathrm{T}}], \tag{1}$$

where $\mathbf{Y} = [\boldsymbol{y}_1, \cdots, \boldsymbol{y}_i, \cdots, \boldsymbol{y}_N]$. Note that once the lower layer weights $\boldsymbol{W}$ are fixed, the hidden layer values $\mathbf{H} = [\boldsymbol{h}_1, \cdots, \boldsymbol{h}_i, \cdots, \boldsymbol{h}_N]$ are also determined uniquely. And subsequently, the upper layer weights $\mathbf{U}$ can be determined by setting the gradient

$$\begin{aligned}\frac{\partial E}{\partial \mathbf{U}} &= \frac{\partial \mathrm{Tr}[(\mathbf{U}^{\mathrm{T}}\mathbf{H} - \mathbf{T})(\mathbf{U}^{\mathrm{T}}\mathbf{H} - \mathbf{T})^{\mathrm{T}}]}{\partial \mathbf{U}} \\ &= 2\mathbf{H}(\mathbf{U}^{\mathrm{T}}\mathbf{H} - \mathbf{T})^{\mathrm{T}}\end{aligned} \tag{2}$$

$$\begin{aligned}\frac{\partial E}{\partial \mathbf{W}} &= \frac{\partial \mathrm{Tr}[(\mathbf{U}^{\mathbf{T}}\mathbf{H}-\mathbf{T})(\mathbf{U}^{\mathbf{T}}\mathbf{H}-\mathbf{T})^{\mathrm{T}}]}{\partial \mathbf{W}} = \frac{\partial \mathrm{Tr}[([(\mathbf{H}\mathbf{H}^{\mathrm{T}})^{-1}\mathbf{H}\mathbf{T}^{\mathrm{T}}]^{\mathrm{T}}\mathbf{H}-\mathbf{T})([(\mathbf{H}\mathbf{H}^{\mathrm{T}})^{-1}\mathbf{H}\mathbf{T}^{\mathrm{T}}]^{\mathrm{T}}\mathbf{H}-\mathbf{T})^{\mathrm{T}}]}{\partial \mathbf{W}}\\ &= \frac{\partial \mathrm{Tr}[\mathbf{T}\mathbf{T}^{\mathbf{T}}-\mathbf{T}\mathbf{H}^{\mathrm{T}}(\mathbf{H}\mathbf{H}^{\mathrm{T}})^{-1}\mathbf{H}\mathbf{T}^{\mathrm{T}}]}{\partial \mathbf{W}} = \frac{-\partial \mathrm{Tr}[(\mathbf{H}\mathbf{H}^{\mathrm{T}})^{-1}\mathbf{H}\mathbf{T}^{\mathrm{T}}\mathbf{T}\mathbf{H}^{\mathrm{T}}]}{\partial \mathbf{W}}\\ &= \frac{-\partial \mathbf{Tr}[(\sigma(\mathbf{W}^{\mathbf{T}}\mathbf{X})[\sigma(\mathbf{W}^{\mathbf{T}}\mathbf{X})]^{\mathrm{T}})^{-1}\sigma(\mathbf{W}^{\mathbf{T}}\mathbf{X})\mathbf{T}^{\mathbf{T}}\mathbf{T}[\sigma(\mathbf{W}^{\mathbf{T}}\mathbf{X})]^{\mathrm{T}}]}{\partial \mathbf{W}}\\ &= \mathbf{2X}\left[\mathbf{H}^{\mathbf{T}}\circ(\mathbf{1}-\mathbf{H})^{\mathbf{T}}\circ[\mathbf{H}^{\dagger}(\mathbf{H}\mathbf{T}^{\mathbf{T}})(\mathbf{T}\mathbf{H}^{\dagger})-\mathbf{T}^{\mathbf{T}}(\mathbf{T}\mathbf{H}^{\dagger})]\right].\end{aligned} \tag{3}$$

$$\begin{aligned}\frac{\partial \ddot{E}}{\partial \mathbf{W}} &= \frac{\partial \mathrm{Tr}[(\mathbf{U}^{\mathbf{T}}\mathbf{H}-\mathbf{T})\boldsymbol{\Lambda}(\mathbf{U}^{\mathbf{T}}\mathbf{H}-\mathbf{T})^{\mathrm{T}}]}{\partial \mathbf{W}}\\ &= \frac{\partial \mathrm{Tr}[([(\mathbf{H}\boldsymbol{\Lambda}\mathbf{H}^{\mathrm{T}})^{-1}\mathbf{H}\boldsymbol{\Lambda}\mathbf{T}^{\mathrm{T}}]^{\mathrm{T}}\mathbf{H}-\mathbf{T})\boldsymbol{\Lambda}([(\mathbf{H}\boldsymbol{\Lambda}\mathbf{H}^{\mathrm{T}})^{-1}\mathbf{H}\boldsymbol{\Lambda}\mathbf{T}^{\mathrm{T}}]^{\mathrm{T}}\mathbf{H}-\mathbf{T})^{\mathrm{T}}]}{\partial \mathbf{W}}\\ &= \frac{\partial \mathrm{Tr}[\mathbf{T}\boldsymbol{\Lambda}\mathbf{T}^{\mathbf{T}}-\mathbf{T}\boldsymbol{\Lambda}\mathbf{H}^{\mathrm{T}}(\mathbf{H}\boldsymbol{\Lambda}\mathbf{H}^{\mathrm{T}})^{-1}\mathbf{H}\boldsymbol{\Lambda}\mathbf{T}^{\mathrm{T}}]}{\partial \mathbf{W}} = \frac{-\partial \mathrm{Tr}[(\mathbf{H}\boldsymbol{\Lambda}\mathbf{H}^{\mathrm{T}})^{-1}\mathbf{H}\boldsymbol{\Lambda}\mathbf{T}^{\mathbf{T}}\mathbf{T}\boldsymbol{\Lambda}\mathbf{H}^{\mathrm{T}}]}{\partial \mathbf{W}}\\ &= 2\mathbf{X}\left[\mathbf{H}^{\mathbf{T}}\circ(\mathbf{1}-\mathbf{H})^{\mathbf{T}}\circ[\mathbf{H}^{\ddagger}(\mathbf{H}\boldsymbol{\Lambda}\mathbf{T}^{\mathrm{T}})(\mathbf{T}\mathbf{H}^{\ddagger})-\boldsymbol{\Lambda}\mathbf{T}^{\mathrm{T}}(\mathbf{T}\mathbf{H}^{\ddagger})]\right].\end{aligned} \tag{4}$$

to zero, leading to the closed-form solution

$$\mathbf{U} = (\mathbf{H}\mathbf{H}^{\mathbf{T}})^{-1}\mathbf{H}\mathbf{T}^{\mathbf{T}}. \tag{5}$$

Note that (5) defines an implicit constraint between the two sets of weights, **U** and **W**, via the hidden layer output **H**, in the SHLNN. This gives rise to a structure that our new algorithms will exploit in optimizing the neural network.

3. Basic Learning Algorithms

In this section, we describe two learning algorithms that exploit the structural constraints indicated by (5) for the SHLNN described in Section 2.

The first algorithm makes use of the solution (5) but does not make use of the fact that **U** can be considered completely depends on **W**. Given fixed current **U** and **W**, the algorithm first computes gradient

$$\begin{aligned}\frac{\partial E}{\partial \mathbf{W}} &= \frac{\partial \mathrm{Tr}[(\mathbf{U}^{\mathbf{T}}\boldsymbol{\sigma}(\mathbf{W}^{\mathbf{T}}\mathbf{X})-\mathbf{T})(\mathbf{U}^{\mathbf{T}}\boldsymbol{\sigma}(\mathbf{W}^{\mathbf{T}}\mathbf{X})-\mathbf{T})^{\mathrm{T}}]}{\partial \mathbf{W}}\\ &= \mathbf{2X}[\mathbf{H}\circ(\mathbf{1}-\mathbf{H})\circ(\mathbf{U}\mathbf{U}^{\mathbf{T}}\mathbf{H}-\mathbf{U}\mathbf{T})^{\mathbf{T}}]\end{aligned} \tag{6}$$

where $\circ$ is element-wise product. The algorithm then updates **W** using the gradient defined in (6) as

$$\mathbf{W}_{\mathrm{k}+1} = \mathbf{W}_{\mathrm{k}} - \rho\frac{\partial \mathrm{E}}{\partial \mathbf{W}}, \tag{7}$$

where ρ is the learning rate. The algorithm subsequently calculates **U** using the closed-form solution (5).

The second algorithm makes further use of the deterministic nonlinear relationship between **U** and **W** in computing the gradient $\partial E/\partial \mathbf{W}$. By treating **U** a function of **W** and plugging (5) into criterion (1) we obtain the new gradient shown in (3), where

$$\mathbf{H}^{\dagger} = \mathbf{H}^{\mathrm{T}}(\mathbf{H}\mathbf{H}^{\mathbf{T}})^{-\mathbf{1}} \tag{8}$$

is the pseudo-inverse of **H**. In the derivation of (3) we used the fact that $\mathbf{H}\mathbf{H}^{\mathrm{T}}$ is symmetric and so is $(\mathbf{H}\mathbf{H}^{\mathbf{T}})^{-\mathbf{1}}$.

Since this second version of the algorithm takes advantage of the effect of **W** on **U,** it tends to move **W** towards a direction that finds the optimal points faster. However, due to the more complicated gradient calculation that involves a pseudo-inverse, each iteration takes longer time than the first version. Note that we grouped the products of matrices in (3). This is necessary to reduce the memory usage when the number of samples becomes unduly large.

4. Accelerated Algorithms

The algorithms described in the preceding section updates the neural network weights based on the current gradient only. However, it has been shown for the convex problems that the convergence speed can be improved if the gradient information over the history is used when updating the weights [7][8]. Although the speedup may not be guaranteed in theory for our non-convex problems, we have observed in practice that such algorithms do converge faster and to a better place. In this paper, we used the FISTA technique [8] to accelerate the learning process. More specifically, we choose $\mathbf{W}_0$ and set $\bar{\mathbf{W}}_1 = \mathbf{W}_0$ and $m_1 = 1$ during initialization. We then update **W** and $\bar{\mathbf{W}}$ according to

$$\mathbf{W}_{\mathrm{k}} = \bar{\mathbf{W}}_{\mathrm{k}} - \rho\frac{\partial E}{\partial \bar{\mathbf{W}}}, \tag{9}$$

$$m_{k+1} = \frac{1}{2}\left(1+\sqrt{1+4m_k^2}\right), \quad \text{and} \tag{10}$$

$$\bar{\mathbf{W}}_{\mathrm{k}+1} = \mathbf{W}_{\mathrm{k}} + \frac{m_{k-1}}{m_{k+1}}(\mathbf{W}_{\mathrm{k}} - \mathbf{W}_{\mathrm{k}-1}). \tag{11}$$

In (3), each sample is weighted the same. However, we can improve the convergence speed by focusing on the samples with most errors. Here we define the weight

$$\lambda_{ii} = \left(\frac{N}{E}\|\boldsymbol{y}_i - \boldsymbol{t}_i\|^2 + 1\right)/2 \tag{12}$$

for each sample i, where E is the square error over the whole training set and N is the training set size. The weights are so chosen that they are positively correlated to the errors introduced by each sample while being smoothed to make sure weights assigned to each sample is at least 0.5. At each step, instead of minimizing E directly we can minimize the weighted error

$$\ddot{E} = \mathrm{Tr}[(\mathbf{Y}-\mathbf{T})\Lambda(\mathbf{Y}-\mathbf{T})^{\mathrm{T}}], \tag{13}$$

where $\boldsymbol{\Lambda} = \mathrm{diag}[\lambda_{11},\cdots,\lambda_{ii},\cdots,\lambda_{NN}]$ is an N by N diagonal weight matrix.

To minimize $\ddot{E}$, once the lower layer weights $\mathbf{W}$ are fixed the upper layer weights $\mathbf{U}$ can be determined by setting the gradient

$$\frac{\partial \ddot{E}}{\partial \mathbf{U}} = \frac{\partial \mathrm{Tr}[(\mathbf{Y}-\mathbf{T})\boldsymbol{\Lambda}(\mathbf{Y}-\mathbf{T})^{\mathrm{T}}]}{\partial \mathbf{U}} = 2\mathbf{H}\boldsymbol{\Lambda}(\mathbf{U}^{\mathrm{T}}\mathbf{H}-\mathbf{T})^{\mathrm{T}} \tag{14}$$

to zero, which has the closed-form solution

$$\mathbf{U} = (\mathbf{H}\boldsymbol{\Lambda}\mathbf{H}^{\mathrm{T}})^{-1}\mathbf{H}\boldsymbol{\Lambda}\mathbf{T}^{\mathrm{T}}. \tag{15}$$

By plugging (15) into (13) and using similar derivation steps used to derive $\frac{\partial E}{\partial \mathbf{W}}$ in (3), we obtain the gradient shown in (4), where

$$\mathbf{H}^{\ddagger} = \boldsymbol{\Lambda}\mathbf{H}^{\mathrm{T}}(\mathbf{H}\boldsymbol{\Lambda}\mathbf{H}^{\mathrm{T}})^{-1}. \tag{16}$$

Note that since we re-estimate the weights after each iteration, the algorithm will try to move the weights with a larger step toward the direction where the error can be most effectively reduced. Once the error for a sample is reduced, the weight for that sample becomes smaller in the next iteration. This not only speeds up the convergence but also makes the training less likely to be trapped into local optima.

5. Experimental Evaluation

Over the past year, we have conducted comprehensive experiments to evaluate the set of four learning algorithms described in this paper on the MNIST database of binary images of handwritten digits [9]. All algorithms perform significantly better than the baseline algorithm. The weighted accelerated algorithm performs the best, achieving recognition accuracy of 98.9% when the deep belief network (DBN) pretraining algorithm is used to initialize the lower-level neural network weights [13]. This is slightly better than the 98.8% accuracy of the DBN reported in [10] but with a small fraction of the training time. The DBN pretraining helped here since the objective function is non-convex w.r.t. the weights and good initialization is still important. However, with the same initialization point, our proposed weighted accelerated algorithm tends to find a better local optimum and find it faster.

In this paper, we focus on our more recent experiments in applying the weighted accelerated learning algorithm to the TIMIT database. The speech data was analyzed using a 25-ms Hamming window with a 10-ms fixed frame rate. We represented the speech using first- to 12th-order Mel frequency cepstral coefficients (MFCCs) and energy, along with their first and second temporal derivatives. The data were normalized to have zero mean and unit variance dimension-wise. All experiments used a context window of 11 frames. This gives rise to a total of 39*11=429 elements in each feature vector, or a super-frame, as the input to the single-hidden-layer neural network. For the neural network output, we used 183 target class labels (i.e., three states for each of the 61 phones), coded in binary zero or one, which we call phone states.

The standard training set of TIMIT consisting of 462 speakers was used, with all SA sentences removed, for training the SHLNN consisting of linear input and output units and sigmoid hidden units. The total number of super-frames in the training data is about 1.2 million. The standard development set of 50 speakers, with a total of 122,488 super-frames, was used for cross validation. Results are reported using the standard 24-speaker core test set consisting of 192 sentences with 7,333 phone tokens and 57,920 super-frames.

The algorithms presented in this paper all are batch-mode based, since the pseudo-inverse is carried out necessarily involving the full training set. However, in our experiments where the full training set is represented by a very large 429 by 1.2M matrix and the hidden layer outputs are represented by an $L \times 1.2M$ matrix with the number of hidden layer units L ranges from 3000 to 6000, the various batch-mode matrix multiplications required by the algorithms easily cause a single computer to be out of memory (we have not implemented our algorithms over parallel machines while carrying out the reported experiments). To avoid the memory overflow problem we organize the training data into a number of mini-batches, and use mini-batch training instead of full batch training. We re-estimate the upper-layer weights using pseudo-inverse over the whole training set after each epoch (i.e. a full sweep of full 1.2M super-frames).

Table 1. Frame-level classification error rates of phones (61 classes) and monophone states (183 classes) as a function of the training epoch; Mini-batch size is 50,000; Step size in gradient descent is 0.01; Network topology is 429-6000-183; DBN pretraining is used to initialize the low-layer network weights.

Epoch	**Train State Err %**	**Dev. State Err %**	**Test Phone Err %**	**Test State Err %**
0	34.10	54.61	45.11	55.20
1	27.19	49.50	39.18	49.83
2	24.98	48.31	38.07	48.61
3	23.43	47.50	37.53	47.86
4	22.62	47.04	36.80	47.24
5	22.09	46.50	36.53	46.87
6	21.73	46.24	36.38	46.64
7	21.45	46.20	36.26	46.48
8	21.20	46.00	36.12	46.30
9	21.10	46.03	36.10	46.33
10	20.92	46.11	36.07	46.40

In Table 1, we show the frame-level phone-state (183 in total) and phone (61 in total) classification error rates obtained by the weighted accelerated training algorithm described in Section 4, as a function of the training epoch. To get the results shown in Table 1 the mini-batch size is set to 50,000, and the step size in gradient descent in Eq. (5) is set to 0.01. The SHLNN used has 6,000 hidden units with lower-layer weights initialized using DBN pretraining procedure. We have found empirically that if weights are initialized randomly then the error rate becomes 30% higher than that presented in Table 1 in most cases due to the fact that the proposed algorithms cannot solve the local optimum problem although better local optimum may be found using them.

There has been very few published work on frame-level phone or phone-state classification. The closest work we have been able to find in [11] reported over 70% phone state error rate with an easier problem that involves only 132 phone-state classes (vs. 183 in our case) but with a more difficult speech database.

We next investigate the effect of the mini-batch size on the phone and state classification error rates, as a way to verify the benefit of batch-mode nature of our developed algorithms. In Table 2, we show such results, all with the same number of epochs. As observed, when the mini-batch size increases, the error rates continue to drop. The size of 100,000 shown in Table 2 is the largest that can fit into the memory of the experimental machine with the memory as large as 48G. It is expected that with more memory or with parallel implementation of the algorithm over many computers, the error rate would continue to drop until the full batch of 1.2M. This is in contrast to the standard minibatch-based stochastic descent algorithm of back-propagation for training neural networks, which does not have the above property.

Table 2. Frame-level classification error rates of phones (61 classes) and states (183 classes) as a function of the mini-batch size; 3000, 4000 or 6000 hidden units shown in parentheses.

Mini-Batch Size (# of hidden units)	Dev. State Err %	Test Phone Err %	Test State Err %
10k (3000)	49.10	39.33	49.50
20k (3000)	48.05	38.35	48.45
20k (4000)	47.65	37.76	48.06
30k (4000)	47.44	37.67	47.81
40k (4000)	47.20	37.65	47.60
50k (4000)	46.88	37.20	47.21
50k (6000)	46.24	36.38	46.64
100k (6000)	45.90	36.13	46.28

6. Conclusions and Discussions

We presented our motivation to develop efficient and parallelizable learning algorithms for neural networks. The weighted accelerated algorithm developed in this work exploits the structure of the neural networks, and moves the weights along the directions that can reduce the errors most. The proposed algorithm is verified in both MNIST handwritten digit recognition and TIMIT frame-level phone and phone-state classification tasks and is a potential solution to the scalability problem for training deep networks. One particular way of using the proposed algorithm is presented in the companion paper [12].

7. Acknowledgements

We thank Dr. G.-B. Huang for fruitful discussions on the baseline SHLNN algorithms, and Dr. A. Acero for the discussion and encouragement of this work.

8. References

[1] D. Yu and L. Deng. "Deep Learning and its Applications to Signal and Information Processing", *IEEE Signal Processing Magazine*, vol. 28, No. 1, pp. 145-154.

[2] G. Dahl, D. Yu, L. Deng, and A. Acero. "Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition", *IEEE Transactions on Audio, Speech, and Language Processing* - Special Issue on Deep Learning for Speech and Language Processing, 2011 (in press).

[3] D. Yu, L. Deng, and G. Dahl, "Roles of Pre-Training and Fine-Tuning in Context-Dependent DBN-HMMs for Real-World Speech Recognition," in *NIPS 2010 workshop on Deep Learning and Unsupervised Feature Learning*, December 2010.

[4] A. Mohamed, D. Yu, and L. Deng, "Investigation of Full-Sequence Training of Deep Belief Networks for Speech Recognition," in *Interspeech,* September 2010, pp. 2846-2849.

[5] L. Deng, M. Seltzer, D. Yu, A. Acero, A. Mohamed, and G. Hinton. "Binary Coding of Speech Spectrograms Using a Deep Auto-encoder," in *Interspeech,* September 2010, pp. 1692-1695.

[6] F. Seide, G. Li, and D. Yu, "Conversational Speech Transcription Using Context-Dependent Deep Neural Networks", *Interspeech* 2011 (to appear).

[7] Y. Nesterov, Introductory Lectures on Convex Optimization: A Basic Course, *Kluwer Academic Publishers*, 2004.

[8] A. Beck, and M. Teboulle, "Gradient-based methods with application to signal recovery problems," Convex Optimization in Signal Processing and Communications, D. Palomar and Y. Eldar (Eds.), *Cambridge University Press*, 2010.

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-Based Learning Applied to Document Recognition", *Proceedings of the IEEE*, 86(11):2278-2324, 1998.

[10] G. E. Hinton, and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks", *Science*, Vol. 313. no. 5786, pp. 504 – 507, 2006.

[11] J. Droppo, M. Seltzer, A. Acero, Y. Chiu. "Towards a non-parametric acoustic model: An acoustic decision tree for observation probability calculation," *Interspeech* 2008.

[12] L. Deng and D. Yu. "Deep Convex Network: A Scalable Architecture for Deep Learning," *Interspeech* 2011, to appear.

[13] D. Yu and L. Deng, "Efficient and Effective Algorithms for Training Single-Hidden-Layer Neural Networks", *Pattern Recognition Letters*, submitted.