This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

Tensor Deep Stacking Networks

Brian Hutchinson, *Student Member, IEEE,* Li Deng, *Fellow, IEEE,* and Dong Yu, *Senior Member, IEEE*

Abstract—A novel deep architecture, the Tensor Deep Stacking Network (T-DSN), is presented. The T-DSN consists of multiple, stacked blocks, where each block contains a bilinear mapping from two hidden layers to the output layer, using a weight tensor to incorporate higher-order statistics of the hidden binary ($[0,1]$) features. A learning algorithm for the T-DSN's weight matrices and tensors is developed and described, in which the main parameter estimation burden is shifted to a convex sub-problem with a closed-form solution. Using an efficient and scalable parallel implementation for CPU clusters, we train sets of T-DSNs in three popular tasks in an increasing order of the data size: handwritten digit recognition using MNIST (60k), isolated state/phone classification and continuous phone recognition using TIMIT (1.1m), and isolated phone classification using WSJ0 (5.2m). Experimental results in all three tasks demonstrate the effectiveness of the T-DSN and the associated learning methods in a consistent manner. In particular, a sufficient depth of the T-DSN, a symmetry in the two hidden layers structure in each T-DSN block, our model parameter learning algorithm, and a softmax layer on top of T-DSN are shown to have all contributed to the low error rates observed in the experiments for all three tasks.

Index Terms—Deep learning, stacking networks, tensor, bilinear models, handwriting image classification, phone classification and recognition, MNIST, TIMIT, WSJ

1 Introduction

Recently, a deep classification architecture built upon blocks of simplified neural network modules, each with a single nonlinear hidden layer and linear input and output layers was proposed, developed and evaluated [1], [2]. It was called the Deep Convex Network, since learning the upper-layer weights could be formulated as solving a convex optimization problem with a closed-form solution, after having initialized the lower-layer weights of each block with a fixed restricted Boltzmann machine. The network was later renamed the Deep Stacking Network (DSN) [3], emphasizing that the mechanism in this network for establishing the deep architecture shares the same philosophy as "stacked generalization" [4]. This name also recognizes that the lower-layer weights are in practice learned for greater effectiveness in classification tasks, so the overall weight learning problem in the DSN is no longer convex. In Section 2.1 of this paper, we provide a short review of the previous DSN as the background for the current work.

The new deep architecture presented in this paper, which we call the Tensor Deep Stacking Network (T-DSN), improves and extends the earlier DSN architecture in two significant ways. First, information about higher-order, covariance statistics in the data, which was not represented in DSN, is now embedded into the T-DSN via a bilinear mapping from two hidden representations to predictions using a third-order tensor. Second, while the T-DSN retains the same linear-nonlinear interleaving structure as DSN in building up the deep architecture, it shifts the major learning problem from the lower-layer, non-convex optimization component to the upper-layer, convex subproblem with a closed-form solution.

Modeling covariance structure directly on raw speech or image data, rather than on the more compact binary ($[0,1]$) hidden feature layers as achieved in T-DSN, was previously proposed in an architecture called mcRBM [5]–[7]. One key distinction is the different domains in which the higher-order structure is represented: one in the visible data as in the mcRBM and another in the hidden units as in our T-DSN. In addition, mcRBM can only be used in one single bottom layer in deep architectures and cannot be easily extended to deeper layers. This is due to the model and learning complexity that are incurred by the the factorization, required to reduce the cubic growth in the size of the weight parameters. Factorization incurs very high computational cost, which, together with the high cost of Hybrid Monte Carlo in learning, makes it impossible to scale up to very large data sets. These difficulties are removed in the proposed T-DSN presented in this paper. Specifically, the same interleaving nature of linear and nonlinear layers inherited from DSN makes it straightforward to stack up deeper layers, and the closed-form solution for the upper-layer weights enables efficient, parallel training. Because of the relatively small sizes in the hidden layers, no factorization is needed for the T-DSN's tensor weights. The mcRBM and T-DSN differ in other ways; in particular, the mcRBM is a generative model optimizing a maximum likelihood objective, while the T-DSN is a discriminative model optimizing a least squares objective. The preliminary work that introduced the T-DSN and its key advantages was described previously

- *Brian Hutchinson is with the Department of Electrical Engineering, University of Washington, Seattle; E-mail: brianhutchinson@ee.washington.edu*
- *Li Deng and Dong Yu are with Microsoft Research. Redmond; E-mail: [deng,dongyu]@microsoft.com*

in [8]. This paper significantly expands the work and contains comprehensive experimental results plus details of the learning algorithm and its implementation.

One major motivation for developing the recent DSN is the lack of scalability and parallelization in the learning algorithms for the Deep Neural Network (DNN) [9]–[11] which have achieved high success in large vocabulary speech recognition (e.g., [10]). In [1]–[3], it was shown that all computational steps of the learning algorithm for DSN are batch-mode based, and are thus amenable to parallel implementation on a cluster of CPU (and/or GPU) nodes. The same computational advantage is retained for the T-DSN architecture introduced in this paper: we are able to parallelize all computations necessary for training and evaluation and thus scale our experiments to larger training sets using a cluster. The ability to continue to benefit from increasingly large batch sizes leads the T-DSN training to use parallelism in a very different way from that of [12], which in contrast distributes asynchronous computation of mini-batch gradients for training a deep sparse autoencoder. Unlike the DNN and other deep architectures which demand GPUs in learning, all results presented in this paper are obtained using exclusively CPU-based clusters.

The organization of this paper is as follows. In Section 2 we provide a brief review of the DSN and discuss how the T-DSN generalizes the DSN. Section 2 also presents an overview of the T-DSN architecture and its bilinear structure that uses tensor weights to make predictions from two hidden representations of the input data. In Section 3, we describe a solution, including its derivation, to the T-DSN learning problem from the algorithmic perspective. A parallel implementation of the learning algorithm for practical applications is detailed in Section 4, which for comparison also includes a computational analysis of sequential learning time complexity. Section 5 presents three sets of evaluation experiments, from a smaller scale (MNIST with 60k training samples for image classification), to a larger scale (TIMIT with 1.12m training samples for both isolated phone classification and continuous phone recognition), and to a still larger scale (WSJ with 5.23m training samples for phone classification). We show the experimental results that consistently demonstrate the effectiveness of the T-DSN architecture and related learning methods.

2 Tensor Deep Stacking Network: An Overview

In this section, we first briefly review the DSN as it relates to the T-DSN, and then describe the general architecture of the T-DSN and its key properties.

2.1 Deep Stacking Networks: A Review

The Deep Stacking Network (DSN) is a scalable deep architecture amenable to parallel weight learning [1]. It is trained in a supervised, block-wise fashion, without the need for back-propagation over all blocks as is common in other popular deep architectures [13]. The DSN blocks, each consisting of a simple, easy-to-learn module, are stacked to form the overall deep network.

Each DSN block, as developed in [1], [2] and which also forms the basis of the T-DSN, is a simplified multilayer perceptron with a single hidden layer. It consists of an upper-layer weight matrix $\mathbf{U}$ that connects the logistic sigmoidal nonlinear hidden layer $\mathbf{h}$ to the linear output layer $\mathbf{y}$, and a lower-layer weight matrix $\mathbf{W}$ that links the input and hidden layers. Let the target vectors $\mathbf{t}$ be arranged to form the columns of $\mathbf{T}$, let the input data vectors $\mathbf{x}$ be arranged to form the columns of $\mathbf{X}$, let $\mathbf{H} = \sigma(\mathbf{W}^T\mathbf{X})$ denote the matrix of hidden units, and assume the lower-layer weights $\mathbf{W}$ are known. The function σ performs the element-wise logistic sigmoid operation $\sigma(x) = 1/(1 + \exp(-x))$. Then learning the upper-layer weight matrix $\mathbf{U}$ can be formulated as a convex optimization problem:

$$\min_{\mathbf{U}^T} f = \|\mathbf{U}^T\mathbf{H} - \mathbf{T}\|_F^2, \tag{1}$$

which has a closed-form solution:

(2)

At the bottom block, $\mathbf{X}$ contains only the raw input data, but for higher blocks of the DSN (as well as the T-DSN), the input data are concatenated with one or more output representations (typically y) from the previous blocks. The lower-layer weight matrix $\mathbf{W}$ in a DSN block can be optimized using an accelerated gradient descent [14] algorithm to minimize the squared error objective in Eqn. 1. Embedding the solution of Eqn. 2 into the objective and deriving the gradient, we obtain

$$\nabla_{\mathbf{W}} f = \mathbf{X}\left[\mathbf{H}^T \circ (\mathbf{1} - \mathbf{H}^T) \circ \mathbf{\Theta}\right], \tag{3}$$

where $\mathbf{1}$ is the matrix of all ones, $\circ$ denotes element-wise multiplication, and

$$\mathbf{\Theta} = 2\mathbf{H}^{\dagger}(\mathbf{H}\mathbf{T}^T)(\mathbf{T}\mathbf{H}^{\dagger}) - \mathbf{T}^T(\mathbf{T}\mathbf{H}^{\dagger}). \tag{4}$$

To train a DSN block one iteratively updates $\mathbf{W}$ using the gradient in Eqn. 3, which by design takes into consideration the optimal $\mathbf{U}$; after $\mathbf{W}$ has been estimated the closed-form $\mathbf{U}$ is computed once, explicitly.

It is to be emphasized that a key element of the standard DSN is that each block outputs an estimate of the final label class (expressed as the vector $\mathbf{y}$) and this estimate is concatenated with the original input vector to form the expanded "input" vector for the next block of the DSN. Because the original input is retained for each higher block, it is guaranteed to perform better on the training set than the previous block. In contrast to other deep architectures (e.g. the deep belief network [9]), the DSN does not aim to discover transformed feature representations. Due to this restrictive nature of building hierarchical structures as well as to the simplicity of each block, the core of the DSN is considerably simplified and optimizing network weights is naturally parallelizable.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

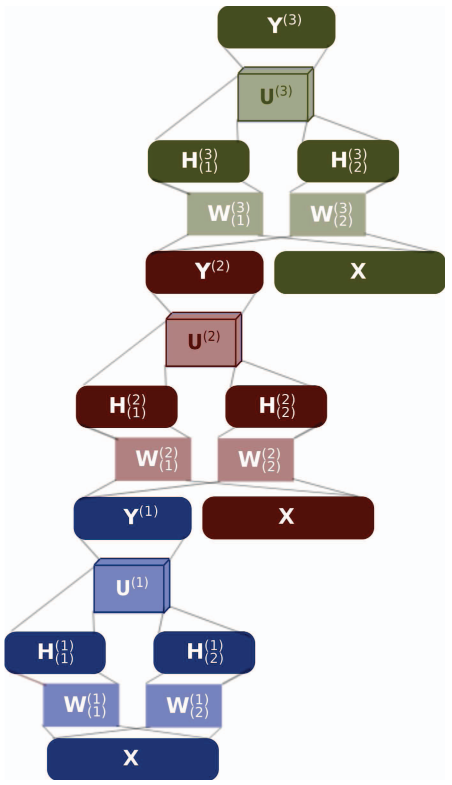

Fig. 1. An example T-DSN architecture with three stacking blocks, where each block consists of three layers, and superscript is used to indicate the block number. Inputs ($\mathbf{X}$) and outputs ($\mathbf{Y}^{(i-1)}$) are concatenated to link two adjacent blocks. The hidden layer in each block has two parallel branches ($\mathbf{H}^{(i)}_{(1)}$ and $\mathbf{H}^{(i)}_{(2)}$).

It is noteworthy that for purely discriminative tasks experiments have shown that DSN, despite its simplicity, performs better than the deep belief network [3].

It is also to be clarified that the estimate of the label class, vector $\mathbf{y}$, is continuous valued, computed as a linear combination of the hidden layer in each block. The classification decision is made at the top block where the index of the maximal value in vector $\mathbf{y}$ determines the label. At the lower blocks of the DSN, the output vector $\mathbf{y}$ is not used for decision but used to concatenate with the original input vector to feed to its immediately upper block. All these attributes have been inherited to the T-DSN to be presented next.

2.2 T-DSN: An Architectural Overview

The DSN just reviewed is a special case of the T-DSN we describe now. In Fig. 1, we illustrate the modular architecture of a T-DSN, where three complete blocks are stacked one on another. The stacking operation of the T-DSN is exactly the same as that for the DSN described in [1]. Unlike the DSN, however, each block of the T-DSN has two sets of lower-layer weight matrices $\mathbf{W}_{(1)}$ and $\mathbf{W}_{(2)}$. They connect the input layer $\mathbf{X}$ with two parallel branches of sigmoidal hidden layers $\mathbf{H}_{(1)}$ and $\mathbf{H}_{(2)}$ shown in Fig. 1. Each T-DSN block also contains a three-way, upper-layer weight tensor $\mathcal{U}$ that connects the two branches of the hidden layer with the output layer.

Note if the T-DSN is used for regression or for classification, then the basic architecture shown in Fig. 1 is sufficient. However, if the T-DSN is to be interfaced with a hidden Markov model (HMM) for structured prediction such as continuous phonetic or word recognition, it is desirable to convert the final output in Fig. 1 into posterior probabilities via an additional softmax layer (the softmax operation applied to a vector exponentiates each entry and then normalizes the vector's entries to sum to one, yielding a distribution). One set of the experiments reported in Section 5.2 are obtained with an additional softmax layer added to the top of Fig. 1.

2.3 Bilinear Prediction from Parallel Hidden Layers

Whereas each block of the DSN produces a single hidden representation of the data and linearly maps from the hidden representation to predictions, each block of the T-DSN uses two hidden representations and combines them *bilinearly* to produce the predictions. A map $\mathcal{F}(u, v)$ is bilinear if it is linear in u for every fixed v, and linear in v for every fixed u [15]. We will see that the T-DSN generalizes the single block structure, replacing a linear map from hidden representation to output with a bilinear mapping, while retaining its desirable modeling and estimation properties. That is, we have a generalization from the mapping of $\mathbb{R}^L \to \mathbb{R}^C$ in the DSN to the mapping of $\mathbb{R}^{L_1} \times \mathbb{R}^{L_2} \to \mathbb{R}^C$ in the T-DSN.

As illustrated in Fig. 1, the first step in the T-DSN operation is to map an input data vector $\mathbf{x} \in \mathbb{R}^D$ to two parallel branches of hidden representations, $\mathbf{h}_{(1)} \in \mathbb{R}^{L_1}$ and $\mathbf{h}_{(2)} \in \mathbb{R}^{L_2}$. Conceptually, these represent two different views of the data (see Appendix C for an analysis of these views). Each hidden representation is obtained non-linearly from the input data according to $\mathbf{h}_{(j)} = \sigma(\mathbf{W}_{(j)}^T \mathbf{x})$, where $\mathbf{W}_{(1)} \in \mathbb{R}^{D \times L_1}$ and $\mathbf{W}_{(2)} \in \mathbb{R}^{D \times L_2}$ are two weight matrices to be estimated. The interactions between these two hidden-layer branches and the prediction vector $\mathbf{y}$ are modeled by a third-order weight tensor, $\mathcal{U} \in \mathbb{R}^{L_1 \times L_2 \times C}$. Specifically, in the second step, the two hidden representations are bilinearly mapped to the prediction vector via $\mathcal{U}$. In tensor notation, the operation is

$$\mathcal{U}(\mathbf{h}_{(1)}, \mathbf{h}_{(2)}) \triangleq (\mathcal{U} \times_1 \mathbf{h}_{(1)}) \times_2 \mathbf{h}_{(2)} = \mathbf{y}, \tag{5}$$

where $\times_i$ denotes multiplying along the ith dimension (mode) of the tensor [16]. In more common notation,

$$y_k = \sum_{i=1}^{L_1} \sum_{j=1}^{L_2} \mathcal{U}_{ijk} h_{(1)i} h_{(2)j} = \mathbf{h}_{(1)}^T \mathbf{U}_k \mathbf{h}_{(2)}, \tag{6}$$

where $\mathbf{U}_k \in \mathbb{R}^{L_1 \times L_2}$ denotes the (matrix) slice of $\mathcal{U}$ obtained by fixing the third index to k and allowing the first two indices to vary.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

It is instructive to link the T-DSN's behavior to that of the DSN, which we can accomplish by manipulating the bilinear notation above. First, define $\tilde{\mathbf{h}}$ to be $\tilde{\mathbf{h}} = \mathbf{h}_{(1)} \otimes \mathbf{h}_{(2)} \in \mathbb{R}^{L_1 L_2}$, where $\otimes$ denotes the Kronecker product: the $((i-1)L_2+j)$th element of $\mathbf{h}_{(1)} \otimes \mathbf{h}_{(2)}$ is equal to $h_{(1)i} h_{(2)j}$, for $i \in \{1, 2, \ldots, L_1\}, j \in \{1, 2, \ldots, L_2\}$. By definition, $\tilde{\mathbf{h}}$ contains all pairs of products between elements in $\mathbf{h}_{(1)}$ and elements in $\mathbf{h}_{(2)}$. One can then vectorize $\mathbf{U}_k$ to create $\tilde{\mathbf{u}}_k = \text{vec}(\mathbf{U}_k) \in \mathbb{R}^{L_1 L_2}$ using the ordering that matches $\tilde{\mathbf{h}}$; that is, if the ℓth element of $\tilde{\mathbf{h}}$ is $\mathbf{h}_{(1)i}\mathbf{h}_{(2)j}$ then the ℓth element of $\tilde{\mathbf{u}}_k$ is $\mathcal{U}_{ijk}$. Then

$$y_k = \sum_{i=1}^{L_1} \sum_{j=1}^{L_2} \mathcal{U}_{ijk} h_{(1)i} h_{(2)j} = \tilde{\mathbf{u}}_k^T \tilde{\mathbf{h}}. \tag{7}$$

Arranging all $\tilde{\mathbf{u}}_k, k = 1, 2, ..., C$, into a matrix $\tilde{\mathbf{U}} = [\tilde{\mathbf{u}}_1 \tilde{\mathbf{u}}_2 \cdots \tilde{\mathbf{u}}_C]$, the overall prediction then becomes

$$\mathbf{y} = \tilde{\mathbf{U}}^T \tilde{\mathbf{h}}. \tag{8}$$

Thus the bilinear mapping from the two hidden-layer branches can be viewed as a linear mapping from a single, implicit hidden representation, $\tilde{\mathbf{h}}$. The linear mapping uses matrix $\tilde{\mathbf{U}}$, which contains all of the elements of the tensor $\mathcal{U}$ *unfolded* into a matrix. Aggregating the implicit hidden representations $\tilde{\mathbf{h}}$ for each of the N training data points into the columns of an $L_1 L_2 \times N$ matrix $\tilde{\mathbf{H}}$, we obtain

$$\mathbf{Y} = \tilde{\mathbf{U}}^T \tilde{\mathbf{H}}. \tag{9}$$

This leads to the same prediction equation as in the DSN, but with an implicit hidden representation $\tilde{\mathbf{h}}$ that contains pairwise multiplicative interactions between $\mathbf{h}_{(1)}$ and $\mathbf{h}_{(2)}$, incorporating second-order statistics of the input data in a parsimonious manner. In Fig. 2 we present an equivalent architecture to the bottom block of Fig. 1, illustrating how the two hidden layers are expanded into an implicit hidden layer. The relationship between the matrices of explicit, lower-dimensional hidden units, $\mathbf{H}_{(1)} = \sigma(\mathbf{W}_{(1)}^T \mathbf{X})$ and $\mathbf{H}_{(2)} = \sigma(\mathbf{W}_{(2)}^T \mathbf{X})$, and matrix of implicit hidden units, $\tilde{\mathbf{H}}$, is

$$\tilde{\mathbf{H}} = \mathbf{H}_{(1)} \odot \mathbf{H}_{(2)}.$$

The $\odot$ operation is the Khatri-Rao product [16], which performs a column-wise Kronecker product.

3 Learning T-DSN Weights – Algorithm

Because the architectures shown in Figures 1 and 2 are equivalent, learning the second layer weights given the implicit hidden representation is the same least squares problem encountered by the DSN. Specifically, the Tikhonov regularized optimization problem (with training target matrix $\mathbf{T}$),

$$\min_{\tilde{\mathbf{U}}^{\mathbf{T}}} \|\mathbf{T} - \tilde{\mathbf{U}}^T \tilde{\mathbf{H}}\|_F^2 + \lambda \|\mathbf{U}\|_F^2, \tag{10}$$

has the closed-form solution of

$$\tilde{\mathbf{U}}^T = \mathbf{T}\tilde{\mathbf{H}}^{\ddagger} = \mathbf{T}\tilde{\mathbf{H}}^T (\tilde{\mathbf{H}}\tilde{\mathbf{H}}^T + \lambda \mathbf{I})^{-1}. \tag{11}$$

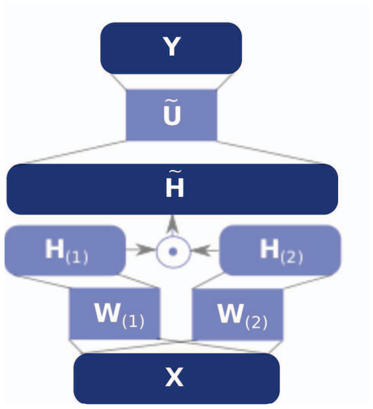

Fig. 2. Equivalent architecture to the bottom block of Fig. 1, where the tensor is unfolded into a large matrix.

Substituting the constraint of Eqn. 11 into the overall objective of Eqn. 10, we can make the full learning more effective by coupling the lower weight matrices with the upper weight matrix. In other words, as with the DSN, the upper-layer weights of a T-DSN block are deterministically computed given the lower-layer weights and thus they do not need to be learned separately and independently. This contrasts with the standard neural network learning algorithms, where both sets of weights are updated incrementally with no direct constraints on each other. Such constraints are available for T-DSN due to the use of linear output layers in each block, and not available for standard neural networks with nonlinear output layers.

We now address the more difficult learning problem for the two lower weight matrices $\mathbf{W}_{(1)}$ and $\mathbf{W}_{(2)}$, since $\tilde{\mathbf{H}}$ is a deterministic function of the lower layer weights. In this work, we adopt the strategy of optimizing $\mathbf{W}_{(1)}$ and $\mathbf{W}_{(2)}$ using methods requiring only first-order oracle information, i.e. those requiring only objective function evaluations and the gradients of the objective function with respect to $\mathbf{W}_{(1)}$ and $\mathbf{W}_{(2)}$. Note that this includes approximate second-order optimization methods such as L-BFGS. The derivation of these gradients has commonality to the DSN case, with an extra step necessitated by the Khatri-Rao product. To simplify the notation throughout the paper we assume $\lambda = 0$ in Eqn. 10 (a detailed derivation of the gradient for arbitrary λ is provided in Appendix A).

With notation analogous to Eqn. 4, let $\tilde{\mathbf{\Theta}}$ denote

$$\tilde{\mathbf{\Theta}} = \nabla_{\tilde{\mathbf{H}}^T} f = 2\tilde{\mathbf{H}}^{\dagger}(\tilde{\mathbf{H}}\mathbf{T}^T)(\mathbf{T}\tilde{\mathbf{H}}^{\dagger}) - \mathbf{T}^T(\mathbf{T}\tilde{\mathbf{H}}^{\dagger}). \tag{12}$$

By the chain rule, we obtain

$$\begin{aligned} \left[\nabla_{\mathbf{H}_{(1)}} f\right]_{in} &= \langle \tilde{\mathbf{\Theta}}^T, \mathbf{E}_{(i,n)}^{L_1 \times N} \odot \mathbf{H}_{(2)} \rangle & (13) \\ &= \sum_{k=1}^{L_2} H_{(2)kn} \tilde{\Theta}_{((i-1)L_2+k),n} & (14) \end{aligned}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

$$\begin{aligned}\left[\nabla_{\mathbf{H}_{(2)}} f\right]_{jn} &= \langle \tilde{\mathbf{\Theta}}^T, \mathbf{H}_{(1)} \odot \mathbf{E}^{L_2\times N}_{(j,n)} \rangle &(15)\\ &= \sum_{k=1}^{L_1} H_{(1)kn} \tilde{\Theta}_{((k-1)L_1+j),n}, &(16)\end{aligned}$$

where $\mathbf{E}^{m\times n}_{(i,j)}$ denotes an $m \times n$ matrix with entry (i,j) equal to one and all other entries zero. Let $\mathbf{\Psi}_{(1)} \in \mathbb{R}^{L_1\times N}$ and $\mathbf{\Psi}_{(2)} \in \mathbb{R}^{L_2\times N}$ denote the matrices $\nabla_{\mathbf{H}_{(1)}} f$ and $\nabla_{\mathbf{H}_{(2)}} f$, respectively. Then, following the derivation of the DSN, we obtain

$$\nabla_{\mathbf{W}_{(1)}} f = \mathbf{X}(\mathbf{H}^T_{(1)} \circ (\mathbf{1} - \mathbf{H^T}_{(1)}) \circ \mathbf{\Psi}_{(1)}).$$

and

$$\nabla_{\mathbf{W}_{(2)}} f = \mathbf{X}(\mathbf{H}^T_{(2)} \circ (\mathbf{1} - \mathbf{H^T}_{(2)}) \circ \mathbf{\Psi}_{(2)}). \quad (17)$$

The $\mathbf{\Psi}_{(1)}$ and $\mathbf{\Psi}_{(2)}$ matrices above have the effect of bridging the high dimensional representation used in $\tilde{\mathbf{\Theta}}$ with the low dimensional representation in $\mathbf{H}_{(1)}$ and $\mathbf{H}_{(2)}$, and are needed due to the Khatri-Rao product.

Using the above gradients one can train a T-DSN block using a number of algorithms; in our experiments, we use the L-BFGS and gradient descent implementations in the Poblano optimization toolbox [17]. Our experience suggests that training a T-DSN block requires around 10-15 iterations of L-BFGS, with up to 7 gradient evaluations per iteration for line search. In our experiments the weight matrices $\mathbf{W}_{(1)}$ and $\mathbf{W}_{(2)}$ are initialized with random values in a range that is tuned using the validation set.

From Eqns. 12 and 17, it is clear that the bulk of the gradient computation is in matrix operations, including matrix multiplications and element-wise matrix products. To bypass memory limitations and to speed up training, we parallelize these matrix operations to run on a CPU cluster. The ability to parallelize training in this manner is key for the scalability of T-DSN training.

3.1 Connections between T-DSN and DSN

We show here that the DSN is in fact a special case of the T-DSN. To distinguish this special case from the general case, we will use $\hat{\mathbf{H}}$ and $\hat{\mathbf{U}}$ to denote the T-DSN's $\tilde{\mathbf{H}}$ and $\tilde{\mathbf{U}}$, respectively. As before, let $\mathbf{h}_{(1)}$, $\mathbf{h}_{(2)}$ denote the two hidden representations in a T-DSN block and let $\mathbf{h}$ denote the (only) hidden representation in a DSN block. Let $\mathbf{W}_{(1)}$, $\mathbf{W}_{(2)}$ and $\mathbf{W}$ denote the corresponding first-layer weight matrices. A DSN is then a special case of a T-DSN when $L_1 = 1$, $L_2 = L$, $\mathbf{W}_{(1)} = \mathbf{0}$ and $\mathbf{W}_{(2)} = \mathbf{W} \in \mathbb{R}^{L\times N}$. Then $\mathbf{h}_{(1)} = \sigma(\mathbf{0}^T\mathbf{x}) = 1/2$ and $\mathbf{h}_{(2)} = \mathbf{h}$, and it follows that

$$\hat{\mathbf{H}} = \mathbf{H}_{(1)} \odot \mathbf{H}_{(2)} = \frac{1}{2}\mathbf{H}. \quad (18)$$

In this case, the pseudo-inverse is

$$\hat{\mathbf{H}}^\dagger = \hat{\mathbf{H}}^T(\hat{\mathbf{H}}\hat{\mathbf{H}}^T)^{-1} = \frac{1}{2}\mathbf{H}^T(\frac{1}{4}\mathbf{H}\mathbf{H}^T)^{-1} = 2\mathbf{H}^\dagger. \quad (19)$$

giving the least squares solution $\hat{\mathbf{U}}^T = \mathbf{T}\hat{\mathbf{H}}^\dagger = 2\mathbf{T}\mathbf{H}^\dagger$. Hence the T-DSN and DSN predictions are identical:

$$\hat{\mathbf{U}}^T\hat{\mathbf{H}} = \left(2\mathbf{U}^T\right)\left(\frac{1}{2}\mathbf{H}\right) = \mathbf{U}^T\mathbf{H} = \mathbf{Y}. \quad (20)$$

In Appendix B, we show that under these same conditions, gradients $\nabla_{\mathbf{W}_{(2)}} f$ and $\nabla_{\mathbf{W}}$ are identical as well.

While the DSN is a special, extremely asymmetric, case of the T-DSN, we have found that the closer the two hidden-layer branches' dimensions are, the better the classification performance (see Section 5.2 for empirical evidence). In the non-degenerate cases ($L_1 \approx L_2 \gg 1$), the T-DSN dramatically shifts the balance of parameters from the lower-layer weights to the upper-layer weights. Whereas the DSN has $D \times L$ lower-layer (explicit) parameters and $L \times C$ upper-layer (implicit) parameters, the T-DSN has $D \times (L_1 + L_1)$ lower-layer parameters and $L_1 \times L_2 \times C$ upper-layer parameters. We conjecture that the observed advantages of using equal numbers of hidden units is because the symmetric case maximizes the ratio of implicit feature dimension L_1L_2 over explicit feature dimension L_1+L_2, and thus makes the best use of the closed-form upper-layer parameters. The key advantage of the non-degenerated T-DSN over the degenerated one (i.e., DSN) is the new ability to capture higher-order feature interactions via the cross product.

4 Learning T-DSN Weights – Parallel Implementation

Stochastic mini-batch training is commonly employed in deep learning training. Empirically, researchers often observe diminishing returns in classification accuracy performance for gradient methods as the mini-batch size increases. In contrast, due to the embedded least squares problem, T-DSN's accuracy in classification tasks continues to improve as the mini-batch size increases. For this reason, it is desirable to use the largest possible amount of training data at each iteration. In this section we analyze the time and space complexities of our T-DSN training algorithm, and introduce a parallel training method that allows us to scale to large training sets.

4.1 Sequential Training Computational Complexity

In order to learn the T-DSN weights, we need to evaluate both the objective function and its gradients with respect to $W_{(1)}$ and $W_{(2)}$ at each iteration. Let $L = L_1L_2$ denote the number of implicit hidden units. Then, computing the gradients defined by Eqn. 17 involves a sequence of cached intermediate steps with the time complexities listed in Table 1, and has an overall space complexity of $O((L + D + C)N)$ (due to the need to store X, $\tilde{H}$, and $\tilde{H}^\dagger$ in memory). Evaluating the objective function in Eqn. 10 has a time complexity of $O(NL(C + D))$ with the same space complexity. In practice, for large enough N this space complexity will exceed the main memory of a single machine, making the ability to parallelize over many machines crucial for handling very large data sets.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

Quantity	Complexity
$\tilde{\mathbf{H}}^{\dagger}$	$O(DN(L_1+L_2)+NL^2+L^3)$
$\tilde{\mathbf{\Theta}}$	$O(LNC)$
$\mathbf{\Psi}_{(i)}$	$O(NL)$
$\nabla\mathbf{W}_{(i)}$	$O(DNL_i)$
Total	$O(NL^2+L^3+NLC+DN(L_1+L_2))$

TABLE 1
Gradient computational complexity, assuming the earlier expressions are cached for use in the later expressions.

Variable	Dimension	Operation
$\mathbf{H}_{(i)}^{\langle k\rangle}$	$L_i \times N_k$	$\sigma(\mathbf{W}_{(i)}^T\mathbf{X}^{\langle k\rangle})$
$\tilde{\mathbf{H}}_{(i)}^{\langle k\rangle}$	$L \times N_k$	$\mathbf{H}_{(1)}^{\langle k\rangle} \odot \mathbf{H}_{(2)}^{\langle k\rangle}$
$\mathbf{B}^{[k]}$	$L \times L$	$\tilde{\mathbf{H}}^{\langle k\rangle}\tilde{\mathbf{H}}^{\langle k\rangle T}$
$\mathbf{F}^{[k]}$	$L \times C$	$\tilde{\mathbf{H}}^{\langle k\rangle}\mathbf{T}^{\langle k\rangle T}$
$\mathbf{B}$	$L \times L$	$\sum_{k=1}^{P}\mathbf{B}^{[k]}$
$\mathbf{F}$	$L \times C$	$\sum_{k=1}^{P}\mathbf{F}^{[k]}$
$\tilde{\mathbf{H}}^{\dagger\langle k\rangle}$	$N_k \times L$	$\tilde{\mathbf{H}}^{\langle k\rangle T}\mathbf{B}^{-1}$
$\mathbf{U}$	$L \times C$	$\mathbf{B}^{-1}\mathbf{F}$
$\mathbf{D}^{[k]}$	$C \times L$	$\mathbf{T}^{\langle k\rangle}\tilde{\mathbf{H}}^{\dagger\langle k\rangle}$
$\mathbf{D}$	$C \times L$	$\sum_{k=1}^{P}\mathbf{D}^{[k]}$
$s^{[k]}$	1×1	$\lVert\mathbf{U}^T\mathbf{X}^{\langle k\rangle} - \mathbf{T}^{\langle k\rangle}\rVert_F^2$
s	1×1	$\sum_{k=1}^{P}s^{[k]}$
$\tilde{\mathbf{\Theta}}^{T\langle k\rangle}$	$L \times N_k$	$2\tilde{\mathbf{H}}^{\dagger\langle k\rangle}\mathbf{F}\mathbf{D} - \mathbf{T}^{\langle k\rangle T}\mathbf{D}$
$\mathbf{\Psi}_{(1)}^{\langle k\rangle}$	$L_1 \times N_k$	$\Psi_{(1)in} = \langle \mathbf{E}_{(i,n)}^{L_1\times N} \odot \mathbf{H}_{(2)}^{\langle k\rangle}, \tilde{\mathbf{\Theta}}^{T\langle k\rangle}\rangle$
$\mathbf{\Psi}_{(2)}^{\langle k\rangle}$	$L_2 \times N_k$	$\Psi_{(2)jn} = \langle \mathbf{H}_{(1)}^{\langle k\rangle} \odot \mathbf{E}_{(j,n)}^{L_2\times N}, \tilde{\mathbf{\Theta}}^{T\langle k\rangle}\rangle$
$\mathbf{G}_{(i)}^{[k]}$	$D \times L_i$	$\mathbf{X}^{\langle k\rangle}(\mathbf{H}_{(i)}^{\langle k\rangle} \circ (\mathbf{1} - \mathbf{H}_{(i)}^{\langle k\rangle}) \circ \mathbf{\Psi}_{(i)}^{\langle k\rangle})$
$\mathbf{G}^{[k]}$	$D \times (L_1+L_2)$	$[\mathbf{G}_{(1)}^{[k]}\ \mathbf{G}_{(2)}^{[k]}]$
$\mathbf{G}$	$D \times (L_1+L_2)$	$\sum_{k=1}^{P}\mathbf{G}^{[k]}$

TABLE 2
The variables computed in the parallel pipeline, their dimension and the mathematical operations required to produce them. Variables with a dimension equal to N_k are recomputed each time; others are cached to disk.

4.2 Parallelizing Matrix Operations

There are well known techniques for parallel matrix multiplication of the form $\mathbf{A} = \mathbf{B}\mathbf{C}^{\mathbf{T}} \in \mathbb{R}^{q\times r}$; in general, they break $\mathbf{B} \in \mathbb{R}^{q\times N}$ and $\mathbf{C} \in \mathbb{R}^{r\times N}$ into submatrices that can be combined to produce $\mathbf{A}$. Because our multiplies will involve instances where common dimension of $\mathbf{B}$ and $\mathbf{C}^T$ (i.e., the number of training samples) is significantly larger than both q and r, we use the following basic matrix multiplication parallelization strategy:

$$\mathbf{A} = \sum_{k=1}^{P}\mathbf{B}^{\langle k\rangle}\mathbf{C}^{\langle k\rangle T}. \tag{21}$$

Where $\mathbf{B}^{\langle k\rangle}$ denotes the kth sub-block of matrix $\mathbf{B}$ that has been divided into P sub-blocks along the second dimension:

$$\mathbf{B} = \left[\mathbf{B}^{\langle 1\rangle}\ \mathbf{B}^{\langle 2\rangle}\ \cdots\ \mathbf{B}^{\langle P\rangle}\right] \tag{22}$$

And likewise, we have for matrix $\mathbf{C}^T$:

$$\mathbf{C}^T = \begin{bmatrix}\mathbf{C}^{\langle 1\rangle T}\\ \vdots \\ \mathbf{C}^{T\langle P\rangle}\end{bmatrix} \tag{23}$$

Decomposing the large matrix operations into many small submatrix operations is the key to parallelizing the gradient and objective function computation.

4.3 Parallel Pipeline

Given current values for the T-DSN parameters $\mathbf{W}'_{(1)}$ and $\mathbf{W}'_{(2)}$, we use the above parallelization strategy to compute the objective function value and gradients. Our parallelization is over training data points: we split any matrix $\mathbf{M}$ with a second dimension N into P submatrices $\mathbf{M}^{\langle k\rangle}$, each of which has N_k columns.

The computation pipeline is broken into a large number of jobs, as illustrated by the directed acyclic graph in Fig. 3. The arrows denote dependence: a job can run once all of the jobs feeding into it have completed. The results of each job in Fig. 3 are cached and used in subsequent processing. There are three qualitatively different kinds of jobs, denoted by the different shapes in the figure. The green three-dimensional boxes each denote a set of P jobs, where the individual jobs process a fraction of the total dataset (the kth batch). The orange rectangle jobs are accumulators, and need only to sum over the P individual files on which they depend. Because they begin accumulating the sum of their ancestors as each one finishes, they introduce minimal delay into the pipeline. The jobs marked with red octagons cannot be run until all ancestor jobs have been finished, since they require synchronizing over the P batches.

Table 2 lists the full set of intermediate variables that must be computed to construct the gradient and evaluate the function; it is a superset of the variables listed in Fig. 3. Note that no variable that is dependent on the data set size (i.e. has a dimension equal to N_k) is cached. These variables are recomputed according to their definitions each time, as the cost of caching these variables to the disk was found to exceed the cost of recomputing them.

At the conclusion of the parallel pipeline, the variable s denotes the objective function f evaluated at $f(\mathbf{W}'_{(1)}, \mathbf{W}'_{(2)})$, and the variable $\mathbf{G}$ denotes the concatenation of the gradients, $\nabla_{W_{(i)}} f$.

4.3.1 *Parallel Timings*

Our original and primary motivation for the parallel implementation was to allow us to scale beyond the memory limit of a single machine. We also measure the effect of parallelization on speed. As usual, there is cost associated with parallelization; namely, the inter-process communication time. Because our implementation uses network disk to store and load cached variables, this cost is non-trivial. Fig. 4 measures empirical wall-clock run-times over repeated single instances of the parallel pipeline (i.e., each computing the gradient and evaluating the objective) on the TIMIT data (1.12m training

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

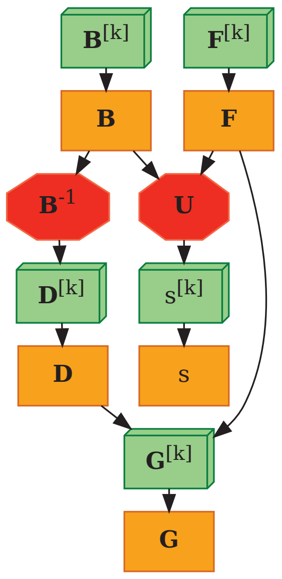

Fig. 3. T-DSN parallel pipeline. The 3D boxes denote sets of P parallel jobs, rectangles denote accumulator jobs and octagons denote sequential jobs. All other values from Table 2 are recomputed as needed.

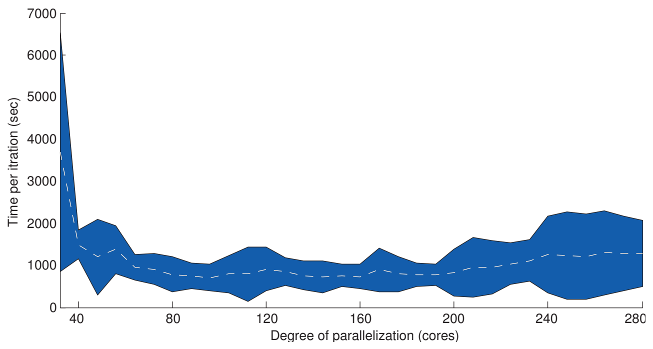

Fig. 4. Empirical wall-clock timings by number of cores for completing the parallel pipeline, repeating eight runs with $L_1 = L_2 = 40$ for each degree of parallelization measured. Upper and lower 95% confidence intervals are plotted in blue; average time is dashed white.

samples). Specifically, the mean and the upper and lower 95% confidence intervals are plotted. To produce the timing results in this figure, we use hidden representations of dimension $L_1 = L_2 = 40$, and repeat a single instance of the parallel pipeline eight times over the number of machines, P, across which the parallel training is distributed. On each machine processing is parallelized over eight cores. A fixed, additional overhead associated with initializing the data is also included in the presented times; in practice these overhead costs would become negligible over the course of training a multiple-block T-DSN. The minimum value for P is four, since lower values caused the compute nodes' memory to be exceeded. For this dataset, the lowest average times are achieved in the range between $P = 10$ and $P = 25$ (80-200 cores). After this, there is a gradual rise in the total computation times, as improvements in computation time are outpaced by the additional disk access and communication costs. In practice, because the speedup is relatively insensitive to the degree of parallelization, we simply set P to be sufficiently large that training does not exceed the compute nodes' memory limits. Note that the stacking nature of the T-DSN means that one cannot parallelize over blocks, only within blocks. Because in practice the number of blocks is limited and the bulk of the computation is spent within blocks, this does not prove to be a major obstacle to training.

5 Evaluation Experiments

In this section, we detail the experiments and present the results aimed to evaluate the effectiveness of the T-DSN architecture described in the preceding sections. Three well-known image and speech databases for benchmarking are used: MNIST, TIMIT, and WSJ.

5.1 MNIST Handwriting Image Recognition

In the first set of experiments, we evaluate the T-DSN architecture and the learning algorithms on the MNIST database of binary images of handwritten digits [18]. The digits have been size-normalized while preserving their aspect ratio. Each original image is centered by computing and translating the center of mass of the pixels, yielding a 28×28 image. The task is to classify each 28×28 image into one of the 10 digits. The MNIST training set is composed of 60,000 examples from approximately 250 writers. The test set is composed of 10,000 patterns. The sets of writers of the training set and test set are disjoint. In the experiments, a small fraction of the training data are held out as a validation set to tune hyper-parameters in the T-DSN. The properties of the validation and test sets in MNIST are found to be very similar to each other.

The architecture of the T-DSN used in the MNIST experiment was shown in Fig. 1, except with four stacking blocks instead of three. The input layer at the bottom block consists of 784 units, one for each black-white pixel in the 28×28 image. Each of the two hidden layers consists of L_1 and L_2 sigmoidal units, denoted by $L_1 \times L_2$ as its hidden-layer configuration. All blocks have the same hidden-layer configuration in all experiments reported in this section. The prediction layer of all blocks has 10 linear units, corresponding to 10 digit output classes. The input layer at the non-bottom blocks is a concatenation of the raw input data and the prediction layer's output from the previous block, thus having the dimensionality of 794.

Figure 5 shows the training objective (mean square error) between the prediction layer's output and the zero-one target averaged over the full training set, as a function of each T-DSN block and also as a function of epochs in the batch-mode gradient-decent training for

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

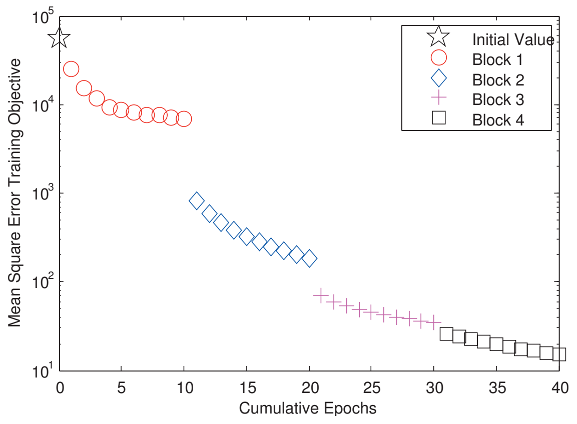

Fig. 5. MNIST: Training objective (mean square error) at each of the training epochs for each block of the T-DSN with hidden-layer configuration of 90×90.

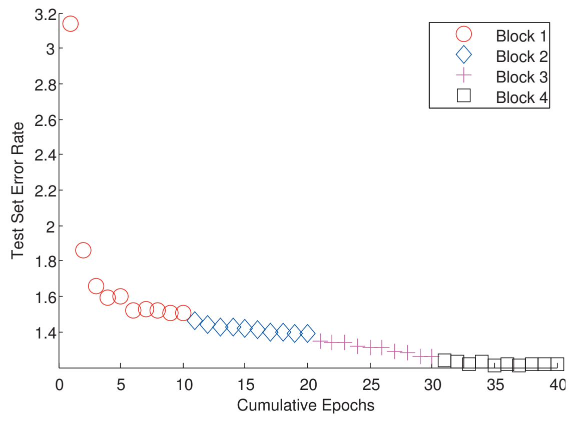

Fig. 6. MNIST: Test-set error rate as a function of training epochs at each block of the T-DSN with hidden-layer configuration of 90×90.

each T-DSN block. The corresponding test-set classification error rate is shown in Figure 6. The hidden-layer configuration of the T-DSN used here is $L_1 \times L_2 = 90 \times 90$. The two sets of input matrix weights to the two hidden layers in each T-DSN block are initialized with small uniform random numbers for parallel gradient-decent training.

The results in Figures 5 and 6 show that when a new block is added to the T-DSN, both the training objective function and the test error rate are reduced even if the training in the previous block already reached near convergence. Further, we observe in Figure 5 that as more blocks are added, the training objective continues to drop to close to zero, and, although not shown in the figure, the error rate for the training set drops to zero as well. There is no obvious rise in errors observed for the test set (Figure 6).

In Table 3, we show the relationship between the test error rate and the hidden-layer configuration. The error rate is obtained at the convergence of the training with the optimal number of blocks and several other hyper

TABLE 3
MNIST: Test-set error rate at convergence as a function of hidden-layer configuration.

Configuration	Test Error Rate
100×100	1.21%
90×90	1.22%
100×81	1.30%
180×45	1.38%
60×60	1.50%
90×40	1.65%
50×50	1.70%
40×40	1.95%
1600×1	2.70%
30×30	2.20%
900×1	3.25%

parameters (learning rates, size of the initial random weights, etc.) determined on the validation set. We observe that with the same number of implicit hidden units, the symmetric configuration is significantly better than non-symmetric configurations. Also, the hidden layers should be sufficiently large to produce low error rate, which is possibly limited by the amount of training data and can be determined on the validation set.

As an extreme case, when one of the two hidden layers in each block reduces to a single unit, the corresponding T-DSN behaves like a DSN with a significantly higher error rate — e.g., 30×30 vs. 900×1 — as shown in Table 3.

The MNIST website provides the results of 68 classifiers. A very large and deep convolutional neural network gives the state-of-the-art error rate of 0.39% [19]. The use of distortions to augment the training data is important to achieve this lowest error rate. Without the use of the distortions, which is impractical in real applications, the error rate was increased to 0.53%. Without the use of convolutional structure and distortions or any other types of special pre-processing, the lowest error rate, 0.83%, was reported in [3] by a carefully tuned and optimized DSN. The error rate of 1.21% reported in this paper is comparable to that achieved using the deep belief network as reported in [9]. It was obtained without careful tuning, without passing of the learned weights from one block to another and without pre-training of weights using restricted Boltzmann machines, which were all exploited in [3].

5.2 TIMIT Phone Classification and Recognition

In the second set of experiments, the TIMIT database is used for evaluating the T-DSN. The training set consists of 462 speakers. The total number of frames in the training data is 1,124,589. The validation set provided by the database contains 50 speakers, with a total of 122,488 frames. Results are reported using the standard 24-speaker core test set consisting of 192 sentences with 7,333 phone tokens and 57,920 frames.

The speech data is analyzed using standard Mel frequency cepstral coefficients (MFCCs). All experiments

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

used a context window of 11 frames. This gives a total of $39 \cdot 11 = 429$ elements in each feature vector as the raw input to T-DSN. This window size was shown to be optimal for the TIMIT phone recognition task in different kinds of deep networks published earlier (e.g., [20]) and has not been customized for the T-DSN in this study. For the prediction at each layer of the T-DSN, we use 183 target class labels (i.e., three states for each of the 61 phones), which we call "phone states," with a zero-one (also called one-hot) encoding scheme. Phone boundaries are labeled in the corpus; we obtain the phone state labels by a state-frame alignment using a strong GMM-HMM system, as is common for recent deep learning work for speech recognition.

The results reported in this section are obtained using the main T-DSN architecture illustrated in Fig. 1, where the number of stacking blocks is between 8 to 13 as determined on the validation set. In some experiments, additional one or more hidden layer(s) and a softmax layer are added to the top of the T-DSN for computing frame-level state posterior probabilities. This latter step is needed for phone recognition task when a further dynamic programming step is used to reach the phone recognition decision. Only symmetric hidden-layer configurations are used, and we tune configurations between $L_1 \times L_2 = 70 \times 70$ and 100×100.

In Table 4 (a) and (b), we compare the Frame-level State classification Error Rates (F-SER) with (b) and without (a) using a trained softmax layer on top of the T-DSN. Obtaining the results of F-SER requires no additional post-processing, making the TIMIT experiment as simple as MNIST. Comparing (a) and (b), we observe noticeable error reduction after the softmax layer is added. In each case, we also compare T-DSN and its corresponding DSN. Similar to the MNIST experiments, the T-DSN gives lower errors, especially in the case of softmax output (b). With the softmax output, we can also evaluate using the cross entropy (CE) measure for the test set. Cross entropy is the average of negative log (base-e) posterior probabilities over all frames in the test set, computed from the softmax layer. The lower the cross entropy, the better the performance. In Table 4(b), we further compare T-DSN with two versions [20], [21] of deep neural nets (DNN), and show that the T-DSN and DSN are both superior to DNN in both error rate and cross entropy.

We now present a new set of TIMIT results, which are more meaningful to speech researchers, after post-processing of the T-DSN outputs in Table 5. The first measure is framewise phone error rate, computed by: 1) collapsing three sequential units (states) associated with each phone into one single phone class using majority voting over all frames within the phone boundaries in each test sentence as provided by the TIMIT database; and 2) collapsing a total of 183 output units into 39 phone-like units [22]. Using this new measure after collapsing, we observe a lower error rate using T-DSN than DSN and two versions of DNN. When the outputs of T-DSN are fed further to a 5-hidden-layer DNN, denoted as "T-DSN + DNN" in Table 5, the framewise phone error rate is dropped further.

TABLE 4
TIMIT: Comparing T-DSN (and DSN) before (a) and after (b) adding softmax layers to produce posterior probabilities, in terms of frame-level state error rate (F-SER) and cross entropy value (CE)

(a) Linear Output

Model	F-SER
TDSN	42.6%
DSN (mini-batch)	44.3%
DSN (full-batch)	42.7%

(b) Posterior Output

Model	F-SER	CE
TDSN	40.9%	2.02
DSN	41.8%	2.16
DNN	45.0%	2.28
MMI-DNN	43.0%	2.20

The second measure (shown in the last column of Table 5) is continuous phonetic recognition error rate. This is computed by: 1) collapsing a total of 183 T-DSN output units into 39 phone-like units, 2) normalizing the softmax outputs of T-DSN by state priors so that the posterior probabilities of states are converted to a quantity proportional to data likelihoods for each state, and 3) using a dynamic programming step across full sentences to determine phone recognition errors (substitution, deletion and insertion errors). This last step is also called "phonetic decoding", where a standard bigram phone-level "language" model is used with the language model weight and insertion penalty tuned using the validation data. The results in Table 5 using this measure again demonstrate superior performance of T-DSN, especially when the outputs of T-DSN are further processed by a DNN.

The original motivation of this work was to make the learning of deep networks scalable by replacing stochastic gradient descent algorithm for fine tuning with the parallelizable batch-mode learning. As both the DSN and T-DSN do "fine tuning" only within the block rather than through the entire deep network as carried out for DNN, we expected at best a matching performance to DNN. While DSN alone has not matched the low phonetic recognition error rate achieved by DNN, T-DSN produces a slightly lower error rate (22.8% vs. 22.9%). Further, for the measures of frame-level error rates and cross entropy, both DSN and T-DSN outperform DNN and even MMI-DNN [20] . The state of the art TIMIT phone recognition error rate is 20.1%, reported and analyzed very recently in [23], lower than 22.8% reported here. The differences are due to 1) the use of specially designed convolutional structure; 2) the use of filterband instead of MFCC features; and 3) very expensive optimization using backpropagation. The first two of these differences can be incorporated into the T-DSN architecture (as future work).

For pure classification problems such as MNIST and frame-level phone classification, we have found little difference between mean square error and cross entropy as the loss. However, for continuous phone recognition

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

TABLE 5
TIMIT: Comparing two versions of T-DSN, DSN, and DNN in terms of frame-level phone error rate and of continuous phone recognition error rate.

Networks	Framewise Phone Err Rate	Phone Rec Err Rate
T-DSN	21.0%	22.8%
T-DSN + DNN	20.2%	21.9%
DSN	22.9%	24.6 %
DSN + DNN	22.0%	23.5 %
DNN	23.5%	22.9 %
MMI-DNN	23.0%	22.2 %
GMM-HMM [24]	-	28.6 %

requiring the use of an HMM to interface with the frame-level classifier, the output needs to be in the form of probabilities. In our experiments reported in the right column of Table 5, we use cross entropy for learning, which is substantially more expensive in computation obtains good results.

5.3 WSJ Phone Classification

In the third set of experiments, we use another popular but larger speech database, called 5k-WSJ0, designed for speaker independent speech recognition tasks [25]. As suggested by the name, the 5k-WSJ0 database uses a 5,000 word vocabulary. The training material from the SI84 set (7077 utterances, or 15.3 hours of speech from 84 speakers) in the database is separated into a 6877-utterance training set and a 200-sentence validation set. Evaluation was carried out on the Nov92 evaluation data with 330 utterances from 8 speakers. In this work, we use the same MFCCs and their deltas as in the TIMIT experiments for the short-time spectral representation of the speech signal. With the 10 millisecond frame rate, this database gives over 5-million frames (i.e., samples) in the training data (5,232,244 to be exact), substantially larger than MNIST and TIMIT. Further, unlike the TIMIT database where the phone boundaries in training data are provided by human annotators, no phone boundaries are given in WSJ0. In this work, we generate the phone labels and their boundaries in the training data from the forced alignments using a tied-state cross-word tri-phone Gaussian-mixture-HMM speech recognizer. Test set labels are produced in the same way. These phone labels, with a total of 40 of them, together with their boundaries provide one-to-one mapping between each speech frame with its phone label as the target for training the T-DSN.

In Table 6, we show the performance of a single block of the T-DSN, measured by the frame-level phone classification error rate. In the results presented in Table 6, the two weight matrices $\mathbf{W}_{(1)}$ and $\mathbf{W}_{(2)}$ in Fig. 1 are randomized and not learned via gradient descent. Learning is applied only to tensor $\mathbf{U}$ according to Eqn. 11. Consistent with the MNIST and TIMIT results, we also observe here that larger (and symmetric) hidden layers are better than smaller ones in the T-DSN. Further, a window size of 11 gives a noticeably lower error rate than 7, which in turn gives a further lower error rate than a single frame.

TABLE 6
WSJ: Frame-level phone classification error rate achieved with only one block and with random weight matrices $\mathbf{W}_{(1)}$ and $\mathbf{W}_{(2)}$ in Fig. 1, as a function of hidden-layer configuration and of the input window size.

Configuration	WinSz1	WinSz7	WinSz11
160 × 160	31.1%	29.0%	28.6%
150 × 150	31.6%	29.4%	29.1%
140 × 140	32.8%	30.3%	29.9%
130 × 130	33.4%	30.7%	30.3%
120 × 120	34.0%	31.3%	31.0%
110 × 110	34.5%	32.2%	32.0%
100 × 100	35.3%	32.9%	32.7%
90 × 90	36.8%	34.5%	33.8%
80 × 80	37.2%	35.2%	34.7%
70 × 70	37.5%	35.9%	35.6%
60 × 60	37.8%	37.1%	36.8%
50 × 50	38.0%	37.6%	37.1%
40 × 40	38.8%	38.4%	38.0%
30 × 30	42.3%	40.4%	39.2%

TABLE 7
WSJ: Frame-level phone classification error rate, *after* stacking five blocks and training weight matrices $\mathbf{W}_{(1)}$ and $\mathbf{W}_{(2)}$ for each block, as a function of hidden-layer configuration and of the input window size.

Configuration	WinSz1	WinSz7	WinSz11
160 × 160	23.0%	21.6%	18.9%
150 × 150	23.5%	21.9%	19.2%
140 × 140	24.9%	22.4%	20.0%
130 × 130	25.5%	23.3%	20.8%
120 × 120	26.0%	24.2%	21.5%
110 × 110	27.1%	24.9%	22.2%
100 × 100	28.4%	25.6%	22.8%
90 × 90	29.5%	26.0%	23.6%
80 × 80	30.0%	26.8%	24.9%
70 × 70	30.3%	27.4%	25.5%
60 × 60	30.6%	28.2%	26.4%
50 × 50	30.7%	28.8%	28.0%
40 × 40	31.6%	30.5%	29.4%
30 × 30	34.5%	32.5%	30.6%

In Table 7, frame-level phone classification error rates are shown after five blocks of T-DSN are built, where each block runs gradient-decent learning until convergence with details described in Sections 3 and 4. No softmax layer is added to top of the T-DSN. Again, at the learning convergence, the T-DSNs with larger hidden layer sizes and larger input window sizes are superior to those using smaller ones. Importantly, each entry in Table 7 shows a significantly lower error rate than the corresponding entry in Table 6, demonstrating the effectiveness of building deep T-DSN and of the learning algorithms with its parallel implementation described in Sections 3 and 4.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

6 Discussions and Conclusion

A new architecture for deep learning is presented, the T-DSN, generalizing the earlier DSN architecture. The principal novelty is to split the original large hidden layer (in each block) into two smaller ones, and through their multiplicative outer product and the associated tensor weights, to create a bilinear model exploiting the higher-order covariance structure in binary ($[0,1]$) hidden feature interactions.

The T-DSN retains the computational advantage of the DSN in parallelism and scalability during learning all parameters, including the second layer tensor and the first layer projection weight matrices. Note that the parallelism in learning the T-DSN can be implemented either in a CPU cluster (as carried out in the current study) or in a GPU cluster. A single GPU parallelization speed up over CPU can be between 10-100$\times$ but CPU programming is easier and CPUs are much more affordable than GPUs. All the experimental results presented in this paper have been obtained by parallel implementation of the learning algorithm described in Section 4, using a cluster of CPUs exclusively.

In addition to the above main strengths, the T-DSN has another advantage over the earlier DSN architecture (an advantage shared by the deep tensor neural network [26] over the DNN), in its potential to explicitly incorporate speaker and/or environmental factors, by training one of the hidden representations to encode speaker or environmental information, while effectively gating the other hidden-to-output mapping. Moreover, the T-DSN is equipped with the new stacking mechanism where the more compact dual hidden representations can be concatenated with the input data in stacking the T-DSN blocks. The significantly smaller hidden representation size in the T-DSN than DSN has the effect of bottle-necking the data, aiding "stackability" in the deep architecture by providing flexibility in stacking choices. One can concatenate the raw input data $\mathbf{x}$ with $\mathbf{h}_{(1)}$ and $\mathbf{h}_{(2)}$ instead of the output $\mathbf{y}$ which may potentially be very large in some applications. The bottle-necking effect would permit the T-DSN to pass more information between the blocks without dramatically increasing the input dimension in the higher-level blocks.

With the parallelized implementation of T-DSN already in place, we expect meaningful improvements in real-world speech recognition and other pattern recognition tasks. Further, encouraged by our recent results of DNN and DSN in applications of speech understanding [27] and in speech attribute detection [28], we expect greater success with the use of T-DSN in these and other applications.

Appendix A
T-DSN Gradient Derivation

In this appendix we derive the gradients used for training our lower level weight matrices, $\mathbf{W}_{(1)}$ and $\mathbf{W}_{(2)}$ under the most general conditions.

A.1 Finding the optimal $\mathbf{U}^T$

Given a fixed implicit hidden representation matrix $\tilde{\mathbf{H}}$, consider the Tikhonov regularized least squares objective:

$$f = \|\mathbf{U}^T\tilde{\mathbf{H}} - \mathbf{T}\|_F^2 + \lambda\|\mathbf{U}\|_F^2 \tag{24}$$

The well-known closed-form solution to this problem is

$$\mathbf{U}^T = \mathbf{T}\tilde{\mathbf{H}}^{\ddagger}, \quad \tilde{\mathbf{H}}^{\ddagger} = \tilde{\mathbf{H}}^T\mathbf{A}^{-1}, \quad \mathbf{A} = (\tilde{\mathbf{H}}\tilde{\mathbf{H}}^T + \lambda\mathbf{I}) \tag{25}$$

We reserve the notation $\tilde{\mathbf{H}}^{\dagger}$ for that pseudo-inverse of $\tilde{\mathbf{H}}$, which is clearly equal to $\tilde{\mathbf{H}}^{\ddagger}$ when $\lambda = 0$.

A.2 Deriving $\nabla_{\tilde{\mathbf{H}}^T} f$

We can substitute the closed form solution for $\mathbf{U}^T$ given in Eqn. 25 into the objective function. Our ultimate goal is to express this as a function of $\mathbf{W}_{(1)}$ and $\mathbf{W}_{(2)}$, and compute the gradients with respect to these lower level weight matrices. As an intermediate step, we first compute the gradient of the objective with respect to $\tilde{\mathbf{H}}^T$.

A.2.1 General gradient

$$\begin{aligned} f &= \|\mathbf{U}^T\tilde{\mathbf{H}} - \mathbf{T}\|_F^2 + \lambda\|\mathbf{U}\|_F^2 \\ &= \operatorname{Tr}\left((\mathbf{U}^T\tilde{\mathbf{H}} - \mathbf{T})(\mathbf{U}^T\tilde{\mathbf{H}} - \mathbf{T})^T\right) + \lambda \operatorname{Tr}\left(\mathbf{U}\mathbf{U}^T\right) \end{aligned} \tag{26}$$

Denote the first term of Eqn. 26 by $\alpha(\tilde{\mathbf{H}})$ and the second term by $\beta(\tilde{\mathbf{H}})$. First, we derive $\nabla_{\tilde{\mathbf{H}}^T}\alpha(\tilde{\mathbf{H}})$. By the linearity of trace, we obtain

$$\begin{aligned} \nabla_{\tilde{\mathbf{H}}^T}\alpha(\tilde{\mathbf{H}}) &= \nabla_{\tilde{\mathbf{H}}^T}\operatorname{Tr}\left(\mathbf{T}\tilde{\mathbf{H}}^{\ddagger}\tilde{\mathbf{H}}\tilde{\mathbf{H}}^T\tilde{\mathbf{H}}^{\ddagger T}\mathbf{T}^T\right) \\ &+ \quad -2\nabla_{\tilde{\mathbf{H}}^T}\operatorname{Tr}\left(\mathbf{T}\tilde{\mathbf{H}}^{\ddagger}\tilde{\mathbf{H}}\mathbf{T}^T\right) \end{aligned} \tag{27}$$

Before we can evaluate $\nabla_{\tilde{\mathbf{H}}^T} f$, let us introduce five lemmas.

Lemma A.1: Let $\mathbf{A}$ denote $(\tilde{\mathbf{H}}\tilde{\mathbf{H}}^T + \lambda\mathbf{I})^{-1}$, as before, and let $\mathbf{Z}$ denote an arbitrary real $N \times N$ matrix. Then

$$\nabla_{\tilde{\mathbf{H}}^T}\operatorname{Tr}\left(\mathbf{A}^{-1}\tilde{\mathbf{H}}\mathbf{Z}\tilde{\mathbf{H}}^T\right) = (\mathbf{I} - \tilde{\mathbf{H}}^{\ddagger}\tilde{\mathbf{H}})(\mathbf{Z} + \mathbf{Z}^T)\tilde{\mathbf{H}}^{\ddagger} \tag{28}$$

When $\mathbf{Z}$ is symmetric, this simplifies to

$$\nabla_{\tilde{\mathbf{H}}^T}\operatorname{Tr}\left(\mathbf{A}^{-1}\tilde{\mathbf{H}}\mathbf{Z}\tilde{\mathbf{H}}^T\right) = 2(\mathbf{I} - \tilde{\mathbf{H}}^{\ddagger}\tilde{\mathbf{H}})\mathbf{Z}\tilde{\mathbf{H}}^{\ddagger} \tag{29}$$

Proof: Let $\mathbf{P}$ denote the constant matrix that is the result of evaluating $\mathbf{A}^{-1}$ with a fixed $\tilde{\mathbf{H}}$. Let $\mathbf{Q}$ denote the constant matrix that is the result of evaluating $\tilde{\mathbf{H}}\mathbf{Z}\tilde{\mathbf{H}}^T$ with a fixed $\tilde{\mathbf{H}}$. It follows that

$$\begin{aligned} &\nabla_{\tilde{\mathbf{H}}^T}\operatorname{Tr}\left(\mathbf{A}^{-1}\tilde{\mathbf{H}}\mathbf{Z}\tilde{\mathbf{H}}^T\right) \\ = \; &\nabla_{\tilde{\mathbf{H}}^T}\operatorname{Tr}\left(\mathbf{P}\tilde{\mathbf{H}}\mathbf{Z}\tilde{\mathbf{H}}^T\right) + \nabla_{\tilde{\mathbf{H}}^T}\operatorname{Tr}\left(\mathbf{A}^{-1}\mathbf{Q}\right) \end{aligned} \tag{30}$$

Using Eqn. 107 from [29], and noting $\mathbf{A}$'s symmetry, we can evaluate the first term in Eqn. 30:

$$\begin{aligned} \nabla_{\tilde{\mathbf{H}}^T}\operatorname{Tr}\left(\mathbf{P}\tilde{\mathbf{H}}\mathbf{Z}\tilde{\mathbf{H}}^T\right) &= \nabla_{\tilde{\mathbf{H}}^T}\operatorname{Tr}\left(\mathbf{Z}\tilde{\mathbf{H}}^T\mathbf{P}\tilde{\mathbf{H}}\right) \\ &= \mathbf{Z}^T\tilde{\mathbf{H}}^T\mathbf{A}^{-1} + \mathbf{Z}\tilde{\mathbf{H}}^T\mathbf{A}^{-1} \\ &= (\mathbf{Z} + \mathbf{Z}^T)\tilde{\mathbf{H}}^{\ddagger}. \end{aligned} \tag{31}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

Using Eqn. 114 from [29] we can evaluate the second term in Eqn. 30:

$$\begin{aligned}&\nabla_{\tilde{\mathbf{H}}^T}\operatorname{Tr}\left(\mathbf{A}^{-1}\mathbf{Q}\right)\\ =\;& -\tilde{\mathbf{H}}^T\mathbf{A}^{-1}(\tilde{\mathbf{H}}\mathbf{Z}\tilde{\mathbf{H}}^T+\tilde{\mathbf{H}}\mathbf{Z}^T\tilde{\mathbf{H}}^T)\mathbf{A}^{-1}\\ =\;& -\tilde{\mathbf{H}}^{\ddagger}\tilde{\mathbf{H}}(\mathbf{Z}+\mathbf{Z}^T)\tilde{\mathbf{H}}^{\ddagger}. \qquad (32)\end{aligned}$$

Substituting Eqns. 31 and 32 into Eqn. 30 completes the first part of the lemma. The second part is trivial: for symmetric $\mathbf{Z}$ we have $\mathbf{Z}+\mathbf{Z}^T=2\mathbf{Z}$. □

Lemma A.2:

$$\begin{aligned}&\nabla_{\tilde{\mathbf{H}}^T}\operatorname{Tr}\left(\mathbf{T}\tilde{\mathbf{H}}^{\ddagger}\tilde{\mathbf{H}}\tilde{\mathbf{H}}^T\tilde{\mathbf{H}}^{\ddagger T}\mathbf{T}^T\right)\\ =\;& 2(\mathbf{I}-\tilde{\mathbf{H}}^{\ddagger}\tilde{\mathbf{H}})\tilde{\mathbf{H}}^T\tilde{\mathbf{H}}^{\ddagger T}\mathbf{T}^T\mathbf{T}\tilde{\mathbf{H}}^{\ddagger}\\ &+2(\mathbf{I}-\tilde{\mathbf{H}}^{\ddagger}\tilde{\mathbf{H}})\mathbf{T}^T\mathbf{T}\tilde{\mathbf{H}}^{\ddagger}\tilde{\mathbf{H}}\tilde{\mathbf{H}}^{\ddagger}. \qquad (33)\end{aligned}$$

Proof: Let $\mathbf{P}$ denote the constant matrix that is the result of evaluating $\mathbf{T}\tilde{\mathbf{H}}^{\ddagger}\tilde{\mathbf{H}}$ for a fixed $\tilde{\mathbf{H}}$, then using Lemma A.1,

$$\begin{aligned}&\nabla_{\tilde{\mathbf{H}}^T}\operatorname{Tr}\left(\mathbf{T}\tilde{\mathbf{H}}^{\ddagger}\tilde{\mathbf{H}}\tilde{\mathbf{H}}^T\tilde{\mathbf{H}}^{\ddagger T}\mathbf{T}^T\right)\\ =\;& \nabla_{\tilde{\mathbf{H}}^T}2\operatorname{Tr}\left(\mathbf{P}^T\mathbf{T}\tilde{\mathbf{H}}^{\ddagger}\tilde{\mathbf{H}}\right)\\ =\;& \nabla_{\tilde{\mathbf{H}}^T}2\operatorname{Tr}\left(\mathbf{A}^{-1}\tilde{\mathbf{H}}\mathbf{P}^T\mathbf{T}\tilde{\mathbf{H}}^T\right)\\ =\;& 2(\mathbf{I}-\tilde{\mathbf{H}}^{\ddagger}\tilde{\mathbf{H}})\tilde{\mathbf{H}}^T\tilde{\mathbf{H}}^{\ddagger T}\mathbf{T}^T\mathbf{T}\tilde{\mathbf{H}}^{\ddagger}\\ &+2(\mathbf{I}-\tilde{\mathbf{H}}^{\ddagger}\tilde{\mathbf{H}})\mathbf{T}^T\mathbf{T}\tilde{\mathbf{H}}^{\ddagger}\tilde{\mathbf{H}}\tilde{\mathbf{H}}^{\ddagger}.\end{aligned}$$

□

Lemma A.3:

$$\nabla_{\tilde{\mathbf{H}}^T}\operatorname{Tr}\left(\mathbf{T}\tilde{\mathbf{H}}^{\ddagger}\tilde{\mathbf{H}}\mathbf{T}^T\right)=2(\mathbf{I}-\tilde{\mathbf{H}}^{\ddagger}\tilde{\mathbf{H}})\mathbf{T}^T\mathbf{T}\tilde{\mathbf{H}}^{\ddagger} \qquad (34)$$

Proof: This follows from Lemma A.2 and the fact that $\operatorname{Tr}(PQ)=\operatorname{Tr}(QP)$:

$$\begin{aligned}\nabla_{\tilde{\mathbf{H}}^T}\operatorname{Tr}\left(\mathbf{T}\tilde{\mathbf{H}}^{\ddagger}\tilde{\mathbf{H}}\mathbf{T}^T\right) &= \nabla_{\tilde{\mathbf{H}}^T}\operatorname{Tr}\left(\mathbf{T}\tilde{\mathbf{H}}^T\mathbf{A}^{-1}\tilde{\mathbf{H}}\mathbf{T}^T\right)\\ &= \nabla_{\tilde{\mathbf{H}}^T}\operatorname{Tr}\left(\mathbf{A}^{-1}\tilde{\mathbf{H}}\mathbf{T}^T\mathbf{T}\tilde{\mathbf{H}}^T\right)\\ &= 2(\mathbf{I}-\tilde{\mathbf{H}}^{\ddagger}\tilde{\mathbf{H}})\mathbf{T}^T\mathbf{T}\tilde{\mathbf{H}}^{\ddagger}. \qquad (35)\end{aligned}$$

□

Lemma A.4:

$$\begin{aligned}\nabla_{\tilde{\mathbf{H}}^T}\alpha(\tilde{\mathbf{H}}) &= 2(\mathbf{I}-\tilde{\mathbf{H}}^{\ddagger}\tilde{\mathbf{H}})\tilde{\mathbf{H}}^T\tilde{\mathbf{H}}^{\ddagger T}\mathbf{T}^T\mathbf{T}\tilde{\mathbf{H}}^{\ddagger}\\ &\quad+2(\mathbf{I}-\tilde{\mathbf{H}}^{\ddagger}\tilde{\mathbf{H}})\mathbf{T}^T\mathbf{T}\tilde{\mathbf{H}}^{\ddagger}\tilde{\mathbf{H}}\tilde{\mathbf{H}}^{\ddagger}\\ &\quad-4(\mathbf{I}-\tilde{\mathbf{H}}^{\ddagger}\tilde{\mathbf{H}})\mathbf{T}^T\mathbf{T}\tilde{\mathbf{H}}^{\ddagger} \qquad (36)\end{aligned}$$

Proof: This follows from Eqn. 27 and Lemmas A.2 and A.3 □

Lemma A.5:

$$\nabla_{\tilde{\mathbf{H}}^T}\frac{1}{\lambda}\beta(\tilde{\mathbf{H}})=\nabla_{\tilde{\mathbf{H}}^T}\operatorname{Tr}\left(\mathbf{U}\mathbf{U}^T\right)=$$

$$2\mathbf{T}^T\mathbf{T}\tilde{\mathbf{H}}^{\ddagger}\mathbf{A}^{-1}-2\tilde{\mathbf{H}}^{\ddagger}\left(\tilde{\mathbf{H}}\mathbf{T}^T\mathbf{T}\tilde{\mathbf{H}}^{\ddagger}+\tilde{\mathbf{H}}^{\ddagger T}\mathbf{T}^T\mathbf{T}\tilde{\mathbf{H}}^T\right)\mathbf{A}^{-1} \qquad (37)$$

Proof: Let $\mathbf{P}$ denote a constant matrix that is the result of evaluating $\mathbf{U}^T\mathbf{A}^{-1}$ with a fixed $\tilde{\mathbf{H}}$. Let $\mathbf{Q}$ denote a constant matrix that is the result of evaluating $\tilde{\mathbf{H}}\mathbf{T}^T$ with a fixed $\tilde{\mathbf{H}}$. It follows that

$$\begin{aligned}&\nabla_{\tilde{\mathbf{H}}^T}\operatorname{Tr}\left(\mathbf{U}\mathbf{U}^T\right)\\ =\;& \nabla_{\tilde{\mathbf{H}}^T}2\operatorname{Tr}\left(\mathbf{P}\tilde{\mathbf{H}}\mathbf{T}^T\right)+\nabla_{\tilde{\mathbf{H}}^T}2\operatorname{Tr}\left(\mathbf{U}^T\mathbf{A}^{-1}\mathbf{Q}\right). \qquad (38)\end{aligned}$$

The first term of Eqn. 38 is

$$\begin{aligned}\nabla_{\tilde{\mathbf{H}}^T}2\operatorname{Tr}\left(\mathbf{P}\tilde{\mathbf{H}}\mathbf{T}^T\right) &= \nabla_{\tilde{\mathbf{H}}^T}2\operatorname{Tr}\left(\tilde{\mathbf{H}}\mathbf{T}^T\mathbf{P}\right)\\ &= 2\mathbf{T}^T\mathbf{T}\tilde{\mathbf{H}}^{\ddagger}\mathbf{A}^{-1}. \qquad (39)\end{aligned}$$

The second term of Eqn. 38 is

$$\begin{aligned}&\nabla_{\tilde{\mathbf{H}}^T}2\operatorname{Tr}\left(\mathbf{U}^T\mathbf{A}^{-1}\mathbf{Q}\right)\\ =\;& -2\tilde{\mathbf{H}}^{\ddagger}(\mathbf{Q}\mathbf{U}^T+\mathbf{U}\mathbf{Q}^T)\mathbf{A}^{-1}\\ =\;& -2\tilde{\mathbf{H}}^{\ddagger}(\tilde{\mathbf{H}}\mathbf{T}^T\mathbf{T}\tilde{\mathbf{H}}^{\ddagger}+\tilde{\mathbf{H}}^{\ddagger T}\mathbf{T}^T\mathbf{T}\tilde{\mathbf{H}}^T)\mathbf{A}^{-1} \qquad (40)\end{aligned}$$

where the first equality comes from Eqn. 114 of [29]. Substituting Eqns. 39 and 40 into Eqn. 38 proves the lemma. □

Theorem A.6:

$$\begin{aligned}\nabla_{\tilde{\mathbf{H}}^T}f &= 2(\mathbf{I}-\tilde{\mathbf{H}}^{\ddagger}\tilde{\mathbf{H}})\tilde{\mathbf{H}}^{\ddagger}\tilde{\mathbf{H}}\mathbf{T}^T\mathbf{T}\tilde{\mathbf{H}}^{\ddagger} \qquad (41)\\ &\quad+2(\mathbf{I}-\tilde{\mathbf{H}}^{\ddagger}\tilde{\mathbf{H}})\mathbf{T}^T\mathbf{T}\tilde{\mathbf{H}}^{\ddagger}\tilde{\mathbf{H}}\tilde{\mathbf{H}}^{\ddagger}\\ &\quad-4(\mathbf{I}-\tilde{\mathbf{H}}^{\ddagger}\tilde{\mathbf{H}})\mathbf{T}^T\mathbf{T}\tilde{\mathbf{H}}^{\ddagger}\\ &\quad+\lambda 2\mathbf{T}^T\mathbf{T}\tilde{\mathbf{H}}^{\ddagger}\mathbf{A}^{-1}\\ &\quad-\lambda 2\tilde{\mathbf{H}}^{\ddagger}\left(\tilde{\mathbf{H}}\mathbf{T}^T\mathbf{T}\tilde{\mathbf{H}}^{\ddagger}+\tilde{\mathbf{H}}^{\ddagger T}\mathbf{T}^T\mathbf{T}\tilde{\mathbf{H}}^T\right)\mathbf{A}^{-1}\end{aligned}$$

Proof: This follows from Eqn. 26 with Lemmas A.4 and A.5. □

We use $\mathbf{\Theta}$ to denote $\nabla_{\tilde{\mathbf{H}}^T}f$ throughout this paper.

A.2.2 *Simplified gradient*

When no regularization is used in the objective (i.e. $\lambda=0$), the gradient can be simplified. Importantly, in this case $\tilde{\mathbf{H}}^{\ddagger}=\tilde{\mathbf{H}}^{\dagger}$ (the pseudo-inverse). Recall that $\tilde{\mathbf{H}}^{\dagger}\tilde{\mathbf{H}}\tilde{\mathbf{H}}^{\dagger}=\tilde{\mathbf{H}}^{\dagger}$ and $\tilde{\mathbf{H}}\tilde{\mathbf{H}}^{\dagger}\tilde{\mathbf{H}}=\tilde{\mathbf{H}}$. Under these conditions,

$$\begin{aligned}\nabla_{\tilde{\mathbf{H}}^T}f &= 2(\mathbf{I}-\tilde{\mathbf{H}}^{\dagger}\tilde{\mathbf{H}})\tilde{\mathbf{H}}^{\dagger}\tilde{\mathbf{H}}\mathbf{T}^T\mathbf{T}\tilde{\mathbf{H}}^{\dagger} \qquad (42)\\ &\quad+2(\mathbf{I}-\tilde{\mathbf{H}}^{\dagger}\tilde{\mathbf{H}})\mathbf{T}^T\mathbf{T}\tilde{\mathbf{H}}^{\dagger}\tilde{\mathbf{H}}\tilde{\mathbf{H}}^{\dagger}\\ &\quad-4(\mathbf{I}-\tilde{\mathbf{H}}^{\dagger}\tilde{\mathbf{H}})\mathbf{T}^T\mathbf{T}\tilde{\mathbf{H}}^{\dagger}\\ &= 2\tilde{\mathbf{H}}^{\dagger}\tilde{\mathbf{H}}\mathbf{T}^T\mathbf{T}\tilde{\mathbf{H}}^{\dagger}-2\mathbf{T}^T\mathbf{T}\tilde{\mathbf{H}}^{\dagger}. \qquad (43)\end{aligned}$$

Note this is the particular form of $\mathbf{\Theta}$ in Eqn. 12 presented in Section 3 earlier.

A.3 Deriving $\nabla_{\mathbf{W}_{(i)}}f$

In order to train our model, we need to compute $\nabla_{\mathbf{W}_{(1)}}f$ and $\nabla_{\mathbf{W}_{(2)}}f$. We employ the chain rule (e.g. as stated in Eqn. 126 of [29]):

$$[\nabla_{\mathbf{Q}}f(\mathbf{P})]_{ij}=\operatorname{Tr}\left((\nabla_{\mathbf{P}}f(\mathbf{P}))^T\frac{\partial\mathbf{P}}{\partial Q_{ij}}\right)=\langle\nabla_{\mathbf{P}}f(\mathbf{P}),\frac{\partial\mathbf{P}}{\partial Q_{ij}}\rangle \qquad (44)$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

The notation $\langle\cdot,\cdot\rangle$ denotes matrix inner product: it is an element-wise multiplication followed by a sum.

First we find $\nabla_{\mathbf{H_{(i)}}} f$. It follows from Eqn. 44 that

$$\left[\nabla_{\mathbf{H}_{(1)}} f(\tilde{\mathbf{H}})\right]_{in} = \langle \mathbf{\Theta}^T, \frac{\partial \tilde{\mathbf{H}}}{\partial H_{(1)in}}\rangle \tag{45}$$

By the definition of the Khatri-Rao product, we get

$$\frac{\partial \tilde{\mathbf{H}}}{\partial H_{(1)in}} = \mathbf{E}_{(i,n)}^{L_1\times N} \odot \mathbf{H}_{(2)} \tag{46}$$

where, as used earlier in the paper, $\mathbf{E}_{in}^{L_1\times N}$ denotes an $L_1 \times N$ matrix that is zero everywhere except 1 in the (i,n)th position. Let $\mathbf{\Psi}_{(1)}$ denote the matrix $\nabla_{\mathbf{H}_{(1)}} f(\tilde{\mathbf{H}})$.

Similarly, we use $\mathbf{\Psi}_{(2)}$ to denote $\nabla_{\mathbf{H}_{(2)}} f(\tilde{\mathbf{H}})$, where

$$\left[\nabla_{\mathbf{H}_{(2)}} f(\tilde{\mathbf{H}})\right]_{jn} = \langle \mathbf{\Theta}^T, \mathbf{H}_{(1)} \odot \mathbf{E}_{(j,n)}^{L_2\times N}\rangle. \tag{47}$$

Let $\mathbf{Z}_{(i)}$ denote $\mathbf{W}_{(i)}^T\mathbf{X}$. By another application of the chain rule, we get

$$\left[\nabla_{\mathbf{Z}_{(i)}} f(\mathbf{H_{(i)}})\right]_{jk} = \langle \mathbf{\Psi}_{(i)}, \frac{\partial \mathbf{H}_{(i)}}{\partial Z_{(i)jk}}\rangle. \tag{48}$$

Recall that $\mathbf{H}_{(i)} = \sigma(\mathbf{Z}_{(i)}^T)$. The partial derivative is

$$\frac{\partial \mathbf{H}_{(i)}}{\partial Z_{(i)jk}} = \mathbf{E}_{jk}^{L_i\times N} \circ \mathbf{H}_{(i)} \circ (\mathbf{1} - \mathbf{H}_{(i)}) \tag{49}$$

where $\mathbf{1} \in \mathbb{R}^{L_I\times N}$ is the matrix of all ones. Since this fully decomposes over elements, we have

$$\nabla_{\mathbf{Z}_{(i)}} f(\mathbf{H}_{(i)}) = \mathbf{H}_{(i)} \circ (\mathbf{1} - \mathbf{H}_{(i)}) \circ \mathbf{\Psi}_{(i)} \tag{50}$$

Finally, let $\mathbf{\Omega}_{(i)}$ denote $\nabla_{\mathbf{Z}_{(i)}} f(\mathbf{H}_{(i)})$. Then

$$\left[\nabla_{\mathbf{W}_{(i)}} f(\mathbf{Z}_{(i)})\right]_{jk} = \langle \mathbf{\Omega}_{(i)}, \frac{\partial \mathbf{Z}_{(i)}}{\partial W_{(i)jk}}\rangle. \tag{51}$$

The matrix $\frac{\partial \mathbf{Z}_{(i)}}{\partial W_{(i)jk}}$ is an $L_i \times N$ matrix that is zero everywhere except for the jth row, which contains the kth row of $\mathbf{X}$. Thus taking the element-wise product of $\mathbf{\Omega}$ and $\frac{\partial \mathbf{Z}_{(i)}}{\partial W_{(i)jk}}$ and then summing is equivalent to taking the inner product between the jth row of $\mathbf{X}$ and the k row of $\mathbf{\Omega}$ (i.e. the kth column of $\mathbf{\Omega}^T$). Repeating this for all j and k can be expressed succinctly as a matrix-matrix product:

$$\nabla_{\mathbf{W}_{(i)}} f = \mathbf{X}\mathbf{\Omega}^T = \mathbf{X}\left(\mathbf{H}_{(i)}^T \circ (\mathbf{1} - \mathbf{H}_{(i)})^T \circ \mathbf{\Psi}_{(i)}^T\right). \tag{52}$$

This gives us an expression for the gradients $\nabla_{\mathbf{W}_{(i)}} f$, which we can use to train a T-DSN block.

Appendix B
T-DSN and DSN Gradient Equivalence

Under the conditions described in Sec. 3.1, namely that $\mathbf{W}$ has dimension $D\times 1$ and is entry-wise zero, we show that the gradients $\nabla_{\mathbf{W}_{(2)}} f$ and $\nabla_{\mathbf{W}} f$ are equivalent. Let $\hat{\mathbf{\Theta}} = \nabla_{\hat{\mathbf{H}}} f$ be defined analogously to Eqn. 12. Using the simplified gradient, it follows that

$$\begin{aligned}\hat{\mathbf{\Theta}} &= 2\left(\hat{\mathbf{H}}^\dagger(\hat{\mathbf{H}}\mathbf{T}^T)(\mathbf{T}\hat{\mathbf{H}}^\dagger) - \mathbf{T}^T(\mathbf{T}\hat{\mathbf{H}}^\dagger)\right)\\ &= 4\left(\mathbf{H}^\dagger(\mathbf{H}\mathbf{T}^T)(\mathbf{T}\mathbf{H}^\dagger) - \mathbf{T}^T(\mathbf{T}\mathbf{H}^\dagger)\right) = 2\mathbf{\Theta}.\end{aligned} \tag{53}$$

Then because each entry of $\hat{H}_{(1)}$ is 1/2,

$$\hat{\mathbf{\Psi}}_{(2)in} = \langle \hat{\mathbf{\Theta}}, \hat{\mathbf{H}}_{(1)} \odot \mathbf{E}_{(i,n)}^{L_2\times N}\rangle \Rightarrow \hat{\mathbf{\Psi}}_{(2)} = \frac{1}{2}\hat{\mathbf{\Theta}} = \mathbf{\Theta} \tag{54}$$

Which gives us the following gradient

$$\nabla_{\mathbf{W}_{(2)}} f = \mathbf{X}(\hat{\mathbf{H}}_{(2)}^T \circ (\mathbf{1} - \hat{\mathbf{H}}_{(2)}^T) \circ \mathbf{\Theta}) = \nabla_{\mathbf{W}} f. \tag{55}$$

The gradients are identical under these conditions. For them to remain identical, $\mathbf{W}_{(1)}$ must be clamped to zero.

Appendix C
Complementarity of the Hidden Views

In the T-DSN, an input vector $\mathbf{x} \in \mathbb{R}^D$ is mapped simultaneously to two different hidden representations, $\mathbf{h}_1 \in \mathbb{R}^{L_1}$ and $\mathbf{h}_2 \in \mathbb{R}^{L_2}$, via

$$\mathbf{h}_1 = \sigma(\mathbf{W}_1^T\mathbf{x}), \qquad \mathbf{h}_2 = \sigma(\mathbf{W}_2^T\mathbf{x}).$$

In all cases considered in this work, $L_1 = L_2 \ll D$, so the two linear maps $\mathbf{W}_i^T \in \mathbb{R}^{L_i\times D}$ have full row rank, but a $D - L_i$ dimensional null space. The D-dimensional row spaces of $\mathbf{W}_i$, which we'll denote $\mathcal{R}_i$, can be used to characterize the linear map - it identifies which dimensions of the input space are preserved (and implicitly, which are destroyed). To assess the degree to which these two hidden representations capture different views of the data, we took a pair of matrices $\mathbf{W}_1$ and $\mathbf{W}_2$ trained on the TIMIT task (Section 5.2), where $L_1 = L_2 = 90$ and used two notions of similarity of subspaces to infer whether the T-DSN does indeed push towards extracting different views of the data.

C.1 Measuring "Mass" Lost

Method

To measure the similarity between the two row spaces, we first consider the following score:

$$\mathrm{sim}(\mathbf{W}_1, \mathbf{W}_2) = \sum_i \sqrt{\sum_j \left(\mathbf{B}_{1i}^T\mathbf{B}_{2j}\right)^2}$$

Where $\mathbf{B}_{1i} \in \mathbb{B}^D$ denotes the ith basis vector in some basis of $\mathcal{R}_1$, and $\mathbf{B}_{2j} \in \mathbb{R}^D$ likewise denotes the jth basis vector in some basis of $\mathcal{R}_2$. Assuming $\mathbf{W}_i$ has singular value decomposition $\mathbf{U}_i\mathbf{\Sigma}_i\mathbf{V}_i^T$, then in detail the process of computing sim is:

1) Take the ith basis vector $\mathbf{B}_{1i}$ and translate it into $\mathcal{R}_2$ via $\mathbf{U}_2$, producing $\mathbf{U}_2^T\mathbf{B}_{1i}$.
2) Take the ℓ_2 norm of $\mathbf{U}_2^T\mathbf{B}_{1i}$. If $\|\mathbf{U}_2^T\mathbf{B}_{1i}\|_2$ equals 1, then no mass has been lost during the change of basis; i.e. $\mathbf{B}_{1i}$ is in the rowspace, $\mathcal{R}_2$. If it is equal

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

to 0, then all mass has been lost; i.e. $\mathbf{B}_{1i}$ is in the null space of $\mathbf{W}_2$.

3) sim is the sum over i of the scores computed in (2) above, aggregating the amount of "mass" that was preserved or lost in the change of basis, and thus characterizing the similarity between the two spaces.

The maximum value sim can attain is D, if $\mathcal{R}_1 = \mathcal{R}_2$; the minimum value it can attain is 0, if the two spaces are orthogonal.

Results

For our pair of learned matrices, $\mathbf{W}_1, \mathbf{W}_2 \in \mathbb{R}^{785\times 90}$, we measured sim and obtained the result of 30.41 (out of 90 possible). We also randomly generated $10,000$ pairs of orthonormal basis for 90-dimensional subspaces of $\mathbb{R}^{785}$, and computed the corresponding sim values. On average, sim was 31.09, with a standard deviation of 0.21, putting sim$(\mathbf{W}_1, \mathbf{W}_2)$ over three standard deviations below the mean. Thus, while the two mappings do not capture mutually exclusive information from in the input, they extract more dissimilar information that would be expected by random chance, suggesting that they do in fact push towards extracting complementary views of the data.

C.2 Measuring Canonical Angles

We can also use canonical angles to assess subspace similarity.

Method

The cosine of the kth canonical angle can be defined recursively as the largest inner product between a basis vector of $\mathcal{R}_1$ and a basis vector of $\mathcal{R}_2$, excluding all basis vectors that have already been "used" in previous steps (e.g. were part of a larger inner product). The cosine of canonical angles $\theta_1, \theta_2, \dots \theta_D$ between two D-dimensional subspaces can be computed as the singular values of $\mathbf{U}_1^T\mathbf{U}_2$, where the columns of $\mathbf{U}_i$ form a basis of $\mathcal{R}_i$. One way to measure similarity is to sum the cosines of canonical angles; this notion of similarity too has a maximum value of D (if the spaces are equal) and a minimum value of 0 (if the spaces are orthogonal).

Results

For the same particular pair of estimated matrices, $W_1, W_2 \in \mathbb{R}^{785\times 90}$, we measured the sum of canonical angles and obtained the result of 26.20 (out of 90 possible). We also randomly generated $10,000$ pairs of orthonormal basis for 90-dimensional subspaces of $\mathbb{R}^{785}$, and computed the corresponding sum-of-canonical-angles values. On average, this was 26.71, with a standard deviation of 0.20, putting sim$(\mathbf{W}_1, \mathbf{W}_2)$ roughly 2.5 deviations below the mean. This also suggests that the two subspaces are being pushed to capture different information, but again, they are far from orthogonal.

C.3 Discussion

This analysis provides evidence in support of the notion that the T-DSN learns "different" views of the data. These different views capture a more diverse set of dimensions in the input that would occur by random chance, but are far from orthogonal.

Acknowledgments

The authors would like to thank Karthik Mohan for his valuable feedback on the analysis of the T-DSN model.

References

[1] L. Deng and D. Yu, "Deep convex networks: A scalable architecture for speech pattern classification," in *Proc. Interspeech*, August 2011.

[2] L. Deng and D. Yu, "Deep convex networks for image and speech classification," in *ICML Workshop on LearningArchitectures*, June 2011.

[3] L. Deng, D. Yu, and J. Platt, "Scalable stacking and learning for building deep architectures," in *Proc. ICASSP*, Kyoto, Japan, 2012.

[4] D.H. Wolpert, "Stacked generalization," *Neural Networks*, vol. 5, no. 2, pp. 241–259, 1992.

[5] G. Dahl, M. Ranzato, A. Mohamed, and G. Hinton, "Phone recognition with the mean-covariance restricted bolzmann machine," in *Proc. NIPS*, December 2010.

[6] M. Ranzato, A. Krizhevsky, and G. Hinton, "Factored 3-way restricted Boltzmann machines for modeling natural images," in *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2010, vol. 13.

[7] M. Ranzato and G. Hinton, "Modeling pixel means and covariances using factorized third-order Boltzmann machines," in *Proc. of Computer Vision and Pattern Recognition Conference (CVPR 2010)*, 2010, pp. 2551–2558.

[8] B. Hutchinson, L. Deng, and D. Yu, "A deep architecture with bilinear modeling of hidden representations: applications to phonetic recognition," in *Proc. ICASSP*, Kyoto, Japan, 2012.

[9] G. Hinton and R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5768, pp. 504–507, 2006.

[10] 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*, January 2012.

[11] A. Mohamed, G. Dahl, and G. Hinton, "Acoustic modeling using deep belief networks," *IEEE Transactions on Audio, Speech, and Language Processing*, January 2012.

[12] Q. Le, M. Ranzato, R. Monga, M. Devin, G. Corrado, K. Chen, J. Dean, and A. Ng, "Building high-level features using large scale unsupervised learning," in *Proc. ICML*, 2012.

[13] Y. Bengio, "Learning deep architectures for AI," *Foundations and Trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[14] D. Yu and L. Deng, "Accelerated parallelizable neural network learning algorithm for speech recognition," in *Proc. Interspeech*, August 2011.

[15] E. Weisstein, "Symmetric bilinear form," in *MathWorld and Wikipedia*, 2012.

[16] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, September 2009.

[17] D. M. Dunlavy, T. G. Kolda, and E. Acar, "Poblano v1.0: A matlab toolbox for gradient-based optimization," Tech. Rep. SAND2010-1422, Sandia National Laboratories, Albuquerque, NM and Livermore, CA, Mar. 2010.

[18] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[19] D. Ciresan, U. Meier, J. Masci, L. Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," in *Proc. IJCAI*, 2011.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

[20] A. Mohamed, D. Yu, and L. Deng, "Investigation of full-sequence training of deep belief networks for speech recognition," in *Proc. Interspeech*, September 2010.

[21] A. Mohamed, G. Dahl, and G. Hinton, "Deep belief networks for phone recognition," in *NIPS Workshop on Deep Learning for Speech Recognition and Related Applications*, December 2009.

[22] K. F. Lee and H. W. Hon, "Speaker-independent phone recognition using hidden Markov models," *IEEE Transactions on Audio, Speech & Language Processing*, vol. 37, no. 11, pp. 1641–1648, 1989.

[23] G. Hinton, L. Deng, D. Yu, G. Dahl, A.Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition," *IEEE Signal Processing Magazine*, vol. 29, no. 6, 2012.

[24] L. Deng, D. Yu, and A. Acero, "Structured speech modeling," *IEEE Trans. Audio, Speech and Langauge Proc.*, vol. 14, no. 5, pp. 1492–1504, 2006.

[25] D. B. Paul and J. M. Baker, "The design for the wall street journal-based csr corpus," in *ICSLP*, 1992.

[26] D. Yu, L. Deng, and F. Seide, "Large vocabulary speech recognition using deep tensor neural networks," in *Proc. Interspeech*, 2012.

[27] G. Tur, L. Deng, D. Hakkani-Tur, and X. He, "Toward deeper understanding: Deep convex networks for semantic utterance classification," in *Proc. ICASSP*, Kyoto, Japan, 2012.

[28] D. Yu, S. Siniscalchi, L. Deng, and C. Lee, "Boosting attribute and phone estimation accuracies with deep neural networks for detection-based speech recognition," in *Proc. ICASSP*, Kyoto, Japan, 2012.

[29] K. B. Petersen and M. S. Pedersen, "The matrix cookbook," in *http: matrixbook.com*, 2008.

Brian Hutchinson is currently a Ph.D candidate at the University of Washington, where he earlier received his Master's degree in Electrical Engineering. He also holds degrees in Computer Science (BS, MS) and Linguistics (BA) from Western Washington University. He worked at Microsoft Research, Redmond as a Ph.D. intern in 2010 and 2011.

His research interests include speech and language processing, optimization, pattern classification and machine learning. In particular, he is interested in matrix and tensor rank minimization in the context of language processing.

Li Deng (M87 SM92 F04) Li Deng received the Ph.D. degrees from the University of Wisconsin-Madison. He joined Dept. Electrical and Computer Engineering, University of Waterloo, Ontario, Canada in 1989 as an Assistant Professor, where he became a Full Professor with tenure in 1996. In 1999, he joined Microsoft Research, Redmond, WA as a Senior Researcher, where he is currently a Principal Researcher. Since 2000, he has also been an Affiliate Full Professor and graduate committee member in the Department of Electrical Engineering at University of Washington, Seattle. Prior to MSR, he also worked or taught at Massachusetts Institute of Technology, ATR Interpreting Telecom. Research Lab. (Kyoto, Japan), and HKUST. In broad areas of speech/language processing, signal processing, and machine learning, he has published over 300 refereed papers in leading journals and conferences and 3 books. His recent technical work (since 2009) on industry-scale deep learning with colleagues and collaborators has created significant impact on speech recognition, signal processing, and related applications with high practical value. He has been granted over 60 US or international patents. He is a Fellow of the Acoustical Society of America, a Fellow of the IEEE, and a Fellow of ISCA. He served on the Board of Governors of the IEEE Signal Processing Society (2008-2010). More recently, he served as Editor-in-Chief for the IEEE Signal Processing Magazine (2009-2011), which ranked first in both years among all publications within the Electrical and Electronics Engineering Category worldwide in terms of its impact factor and for which he received the 2011 IEEE SPS Meritorious Service Award. He currently serves as Editor-in-Chief for the IEEE Transactions on Audio, Speech and Language Processing.

Dong Yu (M97 SM06) joined Microsoft Corporation in 1998 and Microsoft Speech Research Group in 2002, where he currently is a senior researcher. He holds a Ph.D. degree in computer science from University of Idaho, an MS degree in computer science from Indiana University at Bloomington, an MS degree in electrical engineering from Chinese Academy of Sciences, and a BS degree (with honor) in electrical engineering from Zhejiang University (China). His current research interests include speech processing, robust speech recognition, discriminative training, spoken dialog system, machine learning, and pattern recognition. He has published more than 100 papers in these areas and is the inventor/coinventor of close to 50 granted/pending patents.

Dr. Dong Yu is currently serving as an associate editor of IEEE transactions on audio, speech, and language processing (2011-) and has served as an associate editor of IEEE signal processing magazine (2008-2011) and the lead guest editor of IEEE transactions on audio, speech, and language processing - special issue on deep learning for speech and language processing (2010-2011).