The Deep Tensor Neural Network With Applications to Large Vocabulary Speech Recognition

Dong Yu, *Senior Member, IEEE*, Li Deng, *Fellow, IEEE*, and Frank Seide, *Member, IEEE*

***Abstract*—The recently proposed context-dependent deep neural network hidden Markov models (CD-DNN-HMMs) have been proved highly promising for large vocabulary speech recognition. In this paper, we develop a more advanced type of DNN, which we call the deep tensor neural network (DTNN). The DTNN extends the conventional DNN by replacing one or more of its layers with a double-projection (DP) layer, in which each input vector is projected into two nonlinear subspaces, and a tensor layer, in which two subspace projections interact with each other and jointly predict the next layer in the deep architecture. In addition, we describe an approach to map the tensor layers to the conventional sigmoid layers so that the former can be treated and trained in a similar way to the latter. With this mapping we can consider a DTNN as the DNN augmented with DP layers so that not only the BP learning algorithm of DTNNs can be cleanly derived but also new types of DTNNs can be more easily developed. Evaluation on Switchboard tasks indicates that DTNNs can outperform the already high-performing DNNs with 4–5% and 3% relative word error reduction, respectively, using 30-hr and 309-hr training sets.**

***Index Terms*—Automatic speech recognition, CD-DNN-HMM, large vocabulary, tensor deep neural networks.**

I. Introduction

RECENTLY, the context-dependent deep neural network hidden Markov model (CD-DNN-HMM) was developed for large vocabulary speech recognition (LVSR) and has been successfully applied to a variety of large scale tasks by a number of research groups worldwide [2]–[9]. The CD-DNN-HMM adopts and extends the earlier artificial neural network (ANN) HMM hybrid system framework [10]–[12]. In CD-DNN-HMMs, DNNs—multilayer perceptrons (MLPs) with many hidden layers—replace Gaussian mixture models (GMMs) and directly approximate the emission probabilities of the tied triphone states (also called senones). In the first set of successful experiments, CD-DNN-HMMs were shown to achieve 16% [2], [3] and 33% [4]–[6] relative recognition error reduction over strong, discriminatively trained CD-GMM-HMMs, respectively, on a large-vocabulary voice search (VS) task [13] and the Switchboard (SWB) phone-call transcription task [14]. Subsequent work on Google voice search and YouTube data [7] and on Broadcast News [8], [9] confirmed the effectiveness of the CD-DNN-HMMs for large vocabulary speech recognition.

In this work, we extend the DNN to a novel deep tensor neural network (DTNN) in which one or more layers are double-projection (DP) and tensor layers (see Section III for the explanation). The basic idea of the DTNN comes from the motivation and assumption that the underlying factors, such as the spoken words, the speaker identity, noise and channel distortion, and so on, which affect the observed acoustic signals of speech can be factorized and be approximately represented as interactions between two nonlinear subspaces. This type of multi-way interaction was hypothesized and explored in neuroscience as a model for the central nervous system [15], which conceptually features brain function as comprising functional geometries via metric tensors in the internal central nervous system representation-spaces, both in sensorimotor and connected manifolds.

In DTNN, we represent the hidden, underlying factors by projecting the input onto two separate subspaces through a double-projection (DP) layer in the otherwise conventional DNN. We subsequently model the interactions among these two subspaces and the output neurons through a tensor with three-way connections. We propose a novel approach to reduce the tensor layer to a conventional sigmoid layer so that the model can be better understood and the decoding and learning algorithms can be cleanly developed. Based on this reduction, we also introduce alternative types of DTNNs. We empirically compare the conventional DNN and the new DTNN on the MNIST handwritten digit recognition task and the SWB phone-call transcription task [14]. The experimental results demonstrate that the DTNN generally outperforms the conventional DNN.

This paper is organized as follows. We briefly review the related work in Section II and introduce the general architecture of the DTNN in Section III, in which the detailed components of the DTNN and the forward computations are also described. Section IV is dedicated to the algorithms we developed in this work for learning DTNN weight matrices and tensors. The experimental results on MNIST digit recognition task and SWB task are presented and analyzed in Section V. We conclude the paper in Section VI.

II. Related Work

In recent years, an extension from matrix to tensor has been proposed to model three-way interactions and to improve the

Manuscript received May 29, 2012; revised September 01, 2012 and October 24, 2012; accepted November 03, 2012. Date of publication nulldate; date of current version nulldate. This work significantly extends and completes the preliminary work described in [1]. The associate editor coordinating the review of this manuscript and approving it for publication was Mark J. F. Gales.

D. Yu and L. Deng are with Microsoft Research, Redmond, WA 98052 USA (e-mail: dongyu@microsoft.com; deng@microsoft.com).

F. Seide is with Microsoft Research Asia, Beijing 100080, China (e-mail: fseide@microsoft.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TASL.2012.2227738

modeling power of neural networks. In this section we briefly survey the related work.

The tensor-based restricted Boltzmann machine (RBM) was proposed to model three-way interactions among pixels in [16] and to model pixel means and covariances in [17]. This three-way RBM is different from ours in two ways. First, it is a pure generative model, although it may be discriminatively fine-tuned, while our DTNN is a discriminative model in nature. Second, the three-way RBM has only one hidden layer and the mechanism in its architecture design does not permit the use of tensor weights in more than one layer stacked one on top of another. In contrast, our DTNN is designed very differently, with the goal of naturally embedding the tensor weights in many stacked hidden layers.

The tensor-based RBM was later extended to the tensor recurrent neural network (RNN) [18]. The tensor-RNN as discussed in [18], however, is also mainly used as a generative model.

In a separate study reported in [19], a softmax layer was gated with a hidden factor layer and a tensor was subsequently used to model the joint interaction among the hidden factors, the inputs, and the labels. However, the gated softmax network investigated in [19] is a less effective shallow model with no additional hidden layer other than the hidden factor layer. In addition, the work of [19] adopted the mixture model which predicts the classes by summing over all possible hidden factor combinations. The DTNN that we will present in this paper, however, is a deep network and it predicts the upper layer directly through the tensor connections as shown in (2) in Section III.

More recent work [20] replaced the single sigmoid hidden layer with a tensor layer in a deep network where blocks of shallow networks are used to construct the stacking deep architecture and each block in the stacking network consists of only one hidden layer. In contrast, in the DTNN, there are many hidden layers, one after the other. In fact any sigmoid layer in the conventional DNN may be replaced with a tensor layer. While the technique of converting the tensor layer to a conventional sigmoid layer in [20] has motivated and facilitated the development of DTNN here, it is worth noting that the deep architecture in [20] had difficulty for large vocabulary speech recognition tasks since the output units are often limited to a moderate size due to the special requirement for convexity in part of the network. The DTNN presented in this paper is free from such a restriction, combining the virtue of DNN in handling large vocabulary speech recognition tasks that require large senone output units and the effective technique of handling tensors developed from [20].

The most recent work reported in [21] presented two versions of a tensor-based DNN. The first version extended the gated softmax network of [19] by incorporating the gated softmax layer into DNNs. However, much like the softmax network in [19], this version also used a mixture model. The second version that also explored tensors as proposed in [21] is closer to the DTNN to be described in this paper. The main difference is that the gating factor in [21] was estimated completely separately from the main network and was only applied at the output layer. In contrast, the DTNN integrates all estimation steps of all parameters including the gating factors in a single, consistent framework.

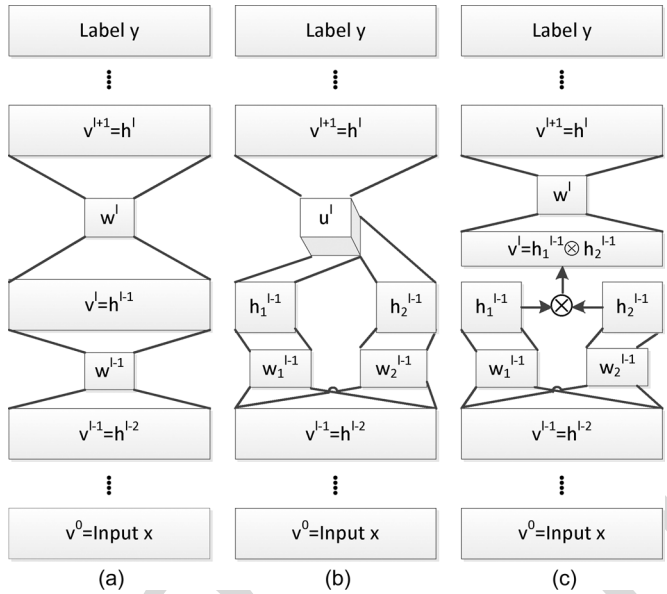


Fig. 1. Architectural illustrations of a conventional DNN and the corresponding DTNN. (a) DNN. (b) DTNN: hidden layer h^{l-1} consists of two parts: h_1^{l-1} and h_2^{l-1}. Hidden layer h^l is a tensor layer to which the connection weights u^l form a three-way tensor. (c) An alternative representation of (b): tensor u^l is replaced with matrix w^l when v^l is defined as the Kronecker product of h_1^{l-1} and h_2^{l-1}.

In summary, the DTNN presented in this paper differentiates itself from previous work in that we use DP layers to automatically factorize information which is later combined through the tensor layers. The distinction also lies in the more flexible incorporation of the DP layers and tensor layers into an otherwise conventional DNN architecture. In addition, our work provides a unified framework to train DNN, DTNN, and their variants (which we will call quasi-DTNN; see Fig. 3 in Section III-B) by mapping the input feature of each layer to a vector and the tensor to a matrix.

III. Architectures of the Deep Tensor Neural Network

The deep tensor neural network (DTNN) is a new type of DNN. It extends the conventional DNN by replacing one or more layers with double-projection and tensor layers, which we will define shortly. In this section we describe the general architecture of the DTNN.

A. DTNN With Double-Projection and Tensor Layers

Fig. 1 illustrates and compares the conventional DNN with the DTNN. Fig. 1(a) shows a conventional DNN, whose input is denoted by x, an $I \times 1$ vector, and the output is y, a $C \times 1$ vector. Subscript l is the layer index. In this conventional DNN, each hidden layer h^{l-1} connects to the next upper layer h^l through a weight matrix w^l and a bias a^l as

$$h^l_{(k)} = \sigma\left(\sum_i w^l_{(i,k)} h^{l-1}_{(i)} + a^l_{(k)}\right), \qquad (1)$$

where i, k are indexes of the hidden units in layers h^{l-1} and h^l, respectively, and $\sigma(x) = 1/(1+\exp(-x))$ is the sigmoid function applied element-wise.

Fig. 1(b) is the corresponding DTNN in which hidden layer h^{l-1} is separated into two parts: h_1^{l-1} (a $K_1^{l-1} \times 1$ vector) and h_2^{l-1} (a $K_2^{l-1} \times 1$ vector). These two parts connect to hidden layer h^l (a $K^l \times 1$ vector) through the three-way tensor u^l [22] of dimension $K_1^{l-1} \times K_2^{l-1} \times K^l$, which is represented with a cube in the figure, according to

$$h_{(k)}^l = \sigma\left(\sum_{i,j} u_{(i,j,k)}^l h_{1(i)}^{l-1} h_{2(j)}^{l-1} + a_{(k)}^l\right), \qquad (2)$$

where i, j, k are indexes of the hidden units in layers h_1^{l-1}, h_2^{l-1} and h^l, respectively. If h_1^{l-1} were to function as a speaker detector and h_2^{l-1} were to function as a spectrum pattern detector, (2) means that for different combinations of speaker $h_{1(i)}^{l-1}$ and spectrum pattern $h_{2(j)}^{l-1}$ a different weight $u_{(i,j,k)}^l$ is assigned for the same detector $h_{(k)}^l$ at next layer.

We call the hidden layer h^{l-1} a double-projection (DP) layer since the information from the previous layer h^{l-2} is projected into two separate subspaces at layer h^{l-1} as h_1^{l-1} and h_2^{l-1}. In this specific case, the DP layer h^{l-1} can be considered as a normal layer where

$$h^{l-1} = \begin{bmatrix} h_1^{l-1} \\ h_2^{l-1} \end{bmatrix} = \sigma\left(\begin{bmatrix} w_1^{l-1} \\ w_2^{l-1} \end{bmatrix} h^{l-2} + \begin{bmatrix} a_1^{l-1} \\ a_2^{l-1} \end{bmatrix}\right). \qquad (3)$$

Here w_1^{l-1} and w_2^{l-1} are weight matrices connecting the hidden layer h^{l-2} with the DP layer parts h_1^{l-1} and h_2^{l-1}, respectively, and a_1^{l-1} and a_2^{l-1} are the corresponding bias terms. As we will see later in this section, however, this is not required and actually is not the case if all layers are DP layers. The only requirement for the DP layer is that it is split into two parts.

The hidden layer h^l is called a tensor layer since the previous layer h^{l-1} is a DP layer that connects with h^l through the weight tensor u^l.

Fig. 1(c) is an alternative view of the same DTNN shown in Fig. 1(b). By defining v^l, the input to the layer l, as

$$v^l = \mathrm{vec}\left(h_1^{l-1} \otimes h_2^{l-1}\right) = \mathrm{vec}\left(h_1^{l-1}\left(h_2^{l-1}\right)^T\right), \qquad (4)$$

where $\otimes$ is the Kronecker product, and $\mathrm{vec}(\cdot)$ is the column-vectorized representation of the matrix, we can organize and rewrite tensor u^l into matrix w^l as represented by a rectangle in Fig. 1(c). In other words, we now have

$$h_{(k)}^l = \sigma\left(\sum_{\mathrm{i}} w_{(i,k)}^l v_{(i)}^l + a_{(k)}^l\right). \qquad (5)$$

This rewriting allows us to reduce and convert *tensor* layers into conventional *matrix* layers and to define the same interface in describing these two different types of layers. For example, in Fig. 1(c) hidden layer h^l can now be considered as a conventional layer as in Fig. 1(a) and can be learned using the conventional backpropagation (BP) algorithm. This rewriting also indicates that the tensor layer can be considered as a conventional layer whose input comprises the cross product of the values passed from the previous layer.

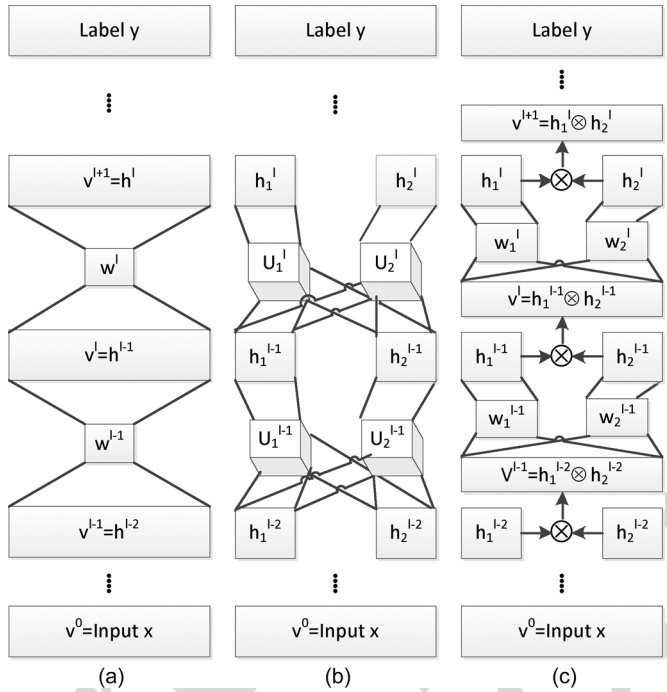


Fig. 2. Comparing conventional DNN and two equivalent views of a DTNN in which all hidden layers are DP tensor layers. (a) DNN; (b) DTNN: normal view where hidden layers are connected through three-way tensors; (c). DTNN: an alternative view where tensors are replaced with matrices when v^l is defined as the Kronecker product of h_1^{l-1} and h_2^{l-1}.

Hidden layer h^{l-1}, however, is still a DP layer that contains two output parts h_1^{l-1} and h_2^{l-1}, which in turn are determined by two separate weight matrices w_1^{l-1} and w_2^{l-1}, in the same way for Fig. 1(b) and Fig. 1(c).

The DTNN shown in Fig. 1 contains only one DP layer. However, nothing prevents other layers from being DP layers. Fig. 2(b) illustrates an example DTNN in which all hidden layers are DP tensor layers. For example, hidden layer h^{l-2} is also separated into two parts h_1^{l-2} and h_2^{l-2} and connects to h_1^{l-1} and h_2^{l-1} through tensors u_1^{l-1} and u_2^{l-1}, respectively. Note that in this DTNN each DP layer projects the input v^l onto two non-linear subspaces h_1^l and h_2^l. The bilinear interaction of these two projections is then combined as the input feature to the adjacent higher layer as quantified by (4). By defining input v^{l-1} to hidden layer h^{l-1} as

$$v^{l-1} = \mathrm{vec}\left(h_1^{l-2} \otimes h_2^{l-2}\right) = \mathrm{vec}\left(h_1^{l-2}\left(h_2^{l-2}\right)^T\right), \qquad (6)$$

tensors u_1^{l-1} and u_2^{l-1} can be rewritten as matrices w_1^{l-1} and w_2^{l-1} as shown in Fig. 2(c). Note that, although all the layers in Fig. 2(b) can be treated as non-tensor layers after this conversion, they are still DP layers since each layer contains two parts.

To summarize, we can represent DTNNs using two types of hidden layers: the conventional sigmoid layer and the DP layer. Each of these hidden layer types can be flexibly placed in the DTNN. For classification tasks the softmax layer that connects the final hidden layer to labels can be used in the DTNN, in the same way as that in the conventional DNN.

Table I summarizes all the forward computations involved in the DTNN, where the input is always converted and written as v^l, a $K_v^l \times 1$ column vector, w^l is the weight matrix, a^l is the

TABLE I
FORWARD COMPUTATIONS IN DTNNS

Condition	Input v^l
first layer	$v^l = v^0 = x$
h^{l-1}: normal layer	$v^l = h^{l-1}$
h^{l-1}: DP layer	$v^l = vec(h_1^{l-1} \otimes h_2^{l-1})$
Condition	**Output h^l**
normal layer	$h^l = \sigma(z^l(v^l)) = \sigma((w^l)^T v^l + a^l)$
DP layer, $i \in \{1,2\}$	$h_i^l = \sigma\left(z_i^l(v^l)\right) = \sigma\left(\left(w_i^l\right)^T v^l + a_i^l\right)$
softmax layer	$p(y\|v^L) = \frac{exp\left(\left(w_y^L\right)^T v^L + a_y^L\right)}{\Sigma_{y'} exp\left(\left(w_{y'}^L\right)^T v^L + a_{y'}^L\right)}$

bias, w_y^L is a column vector of the softmax layer weight matrix w^L, and

$$z^l(v^l) = (w^l)^T v^l + a^l \tag{7}$$

is the activation vector given input v^l.

Note that for the DP layer, the output has two parts

$$h_i^l = \sigma\left(z_i^l(v^l)\right) = \sigma\left(\left(w_i^l\right)^T v^l + a_i^l\right), \tag{8}$$

where $i \in \{1,2\}$ indexes the part number. The two hidden layer vectors h_1^{l-1} and h_2^{l-1} may be augmented with ones when generating the input v^l of the next layer. However, this is unnecessary since the same effect may be achieved by setting weights to 0 and biases to a large positive number for one of the units so that it always outputs 1.

B. Variants of DTNN

The basic DTNN architecture described above can have a number of variants, and we describe two of them here. Fig. 3(a) shows a DTNN variant in which linear activations (i.e., no sigmoid nonlinearity) z_1^{l-1} and z_2^{l-1} are directly connected to layer h^l through tensor u^l. Fig. 3(b) is the equivalent architecture where weight tensor u^l is converted into weight matrix w^l by defining

$$v^l = \text{vec}\left(z_1^{l-1} \otimes z_2^{l-1}\right) = \text{vec}\left(z_1^{l-1}\left(z_2^{l-1}\right)^T\right). \tag{9}$$

Note that the only difference between the architectures of Fig. 3(a), 3(b) and those of Fig. 1(b), 1(c) is that the latter uses a sigmoid non-linearity (as indicated by h_1^{l-1} and h_2^{l-1} instead of z_1^{l-1} and z_2^{l-1} in the DP layer) before connecting to the next layer. This provides numerical stability and also incorporates the former as a special case if the sigmoid function is restricted to the linear range.

Fig. 3(c) shows another variant of the DTNN in which linear DP layers are also used but v^l is redefined as

$$v^l = h^{l-1} = \sigma\left(\text{vec}\left(z_1^{l-1} \otimes z_2^{l-1}\right)\right). \tag{10}$$

The difference between this model and that illustrated in Fig. 1(b), 1(c) is that the sigmoid non-linearity is applied after

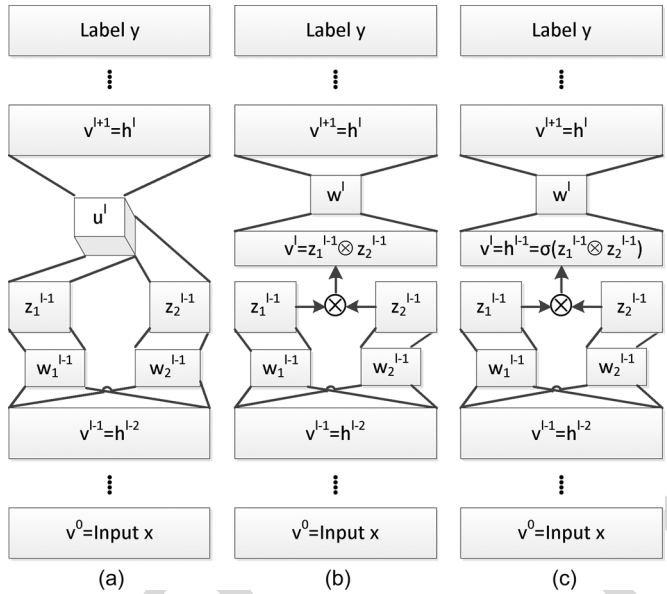


Fig. 3. Two additional types of DTNN. (a) DTNN in which the DP layer is linear (i.e., sigmoid function is not applied). (b) Alternative view of the same DTNN in (a). (c). a quasi-DTNN in which sigmoid non-linearity is applied to the Kronecker product of z_1^{l-1} and z_2^{l-1}. This model, although it models a three-way connection, cannot be represented using a tensor due to the sigmoid non-linearity applied to the Kronecker product of the two input components.

the Kronecker product instead of being applied to the two individual parts of the DP layers. Strictly speaking, the architecture of Fig. 3(c), while also modeling the relations between two subspaces and their upper layer, is not a DTNN since we cannot rewrite and represent it using a tensor. For this reason, we refer to the architecture of Fig. 3(c) as a quasi-DTNN.

IV. LEARNING ALGORITHMS

We optimize the DTNN model parameters by maximizing the negative cross entropy

$$\overline{D} = \frac{1}{N}\sum_x D(x) = \frac{1}{N}\sum_x \sum_y \tilde{p}(y|x)\log p(y|x), \tag{11}$$

commonly used for the conventional DNN, where N is the total number of samples in the training set and $\tilde{p}(y|x)$ is the target probability. When a hard alignment is used $\tilde{p}(y = i|x)$ is 1 if the sample's training label is i and is 0 otherwise. Under that condition, the negative cross entropy is the same as the conditional log-likelihood. The parameters can be learned using the backpropagation (BP) algorithm.

The gradients associated with the softmax layer and the conventional sigmoid layers are the same as that in conventional DNNs. More specifically, for the softmax layer

$$\frac{\partial D(x)}{\partial w^L} = v^L\left(e^L(x)\right)^T, \tag{12}$$

$$\frac{\partial D(x)}{\partial a^L} = e^L(x), \tag{13}$$

where w^L is the $K_v^L \times C$ weight matrix, a^L is the $C \times 1$ bias column vector, and $e^L(x)$ is a $C \times 1$ error column vector with

$$e_i^L(x) = \left(\tilde{p}(y = i|x) - p(y = i|x)\right), \tag{14}$$

where $\tilde{p}(y=i|x)$ is the target probability and $p(y=i|x)$ is the model's predicted probability. For other layers with $l < L$ we define the error signal $e^l(x) = \partial \mathrm{D(x)}/\partial v^{l+1}$.

In the softmax layer, the error can be propagated to the immediately previous layer according to

$$e^{L-1}(\mathrm{x}) = \frac{\partial D(x)}{\partial v^L} = w^L e^L(x). \tag{15}$$

Similarly, for the conventional sigmoid layer, we have

$$\frac{\partial D(x)}{\partial w^l} = v^l \left(\mathrm{diag}\left(\sigma'\left(z^l(v^l)\right)\right) e^l(x)\right)^T, \tag{16}$$

$$\frac{\partial D(x)}{\partial a^l} = \mathrm{diag}\left(\sigma'\left(z^l(v^l)\right)\right) e^l(x), \tag{17}$$

and

$$e^{l-1}(x) = \frac{\partial D(x)}{\partial v^l} = w^l \mathrm{diag}\left(\sigma'\left(z^l(v^l)\right)\right) e^l(x), \tag{18}$$

where $\sigma'(x) = \sigma(x)(1-\sigma(x))$ is the gradient of the sigmoid function applied element-wise, and diag(.) is the diagonal matrix determined by the operand.

However, the gradients are more complicated for the DP layers, which we derive now. Note for the DP layer we have

$$\begin{aligned} v^{l+1} &= \mathrm{vec}\left(h_1^l \left(h_2^l\right)^T\right) \\ &= \left(h_2^l \otimes I_{K_1^l}\right) \mathrm{vec}\left(h_1^l\right) = \left(h_2^l \otimes I_{K_1^l}\right) h_1^l \\ &= \left(I_{K_2^l} \otimes h_1^l\right) \mathrm{vec}\left(\left(h_2^l\right)^T\right) = \left(I_{K_2^l} \otimes h_1^l\right) h_2^l, \end{aligned} \tag{19}$$

where I_K is a $K \times K$ identity matrix. v^{l+1} is thus a $(K_1^l \times K_2^l) \times 1$ column vector whose elements are $v^l_{(j+k\cdot K_1^l)} = h^l_{1,j} h^l_{2,k}$, where we assume matrix and vector index is 0 based. This leads to the gradients

$$\frac{\partial (v^{l+1})^T}{\partial h_1^l} = \frac{\partial \left(\left(h_2^l \otimes I_{K_1^l}\right) h_1^l\right)^T}{\partial h_1^l} = \left(h_2^l\right)^T \otimes I_{K_1^l}, \tag{20}$$

whose $(i, j+k\cdot K_1^l)$-th element is $\delta(i=j)h^l_{2,k}$, and

$$\frac{\partial (v^{l+1})^T}{\partial h_2^l} = \frac{\partial \left(\left(I_{K_2^l} \otimes h_1^l\right) h_2^l\right)^T}{\partial h_2^l} = I_{K_2^l} \otimes \left(h_1^l\right)^T, \tag{21}$$

whose $(i, j+k\cdot K_1^l)$-th element is $\delta(i=k)h^l_{1,j}$.

Note that for the parts $i \in \{1,2\}$

$$\frac{\partial \left(h_i^l\right)^T}{\partial vec\left(w_i^l\right)} = v^l \otimes \mathrm{diag}\left(\sigma'\left(z_i^l(v^l)\right)\right), \tag{22}$$

$$\frac{\partial \left(h_i^l\right)^T}{\partial a_i^l} = \mathrm{diag}\left(\sigma'\left(z_i^l(v^l)\right)\right), \tag{23}$$

and

$$\frac{\partial \left(h_i^l\right)^T}{\partial v^l} = w_i^l \mathrm{diag}\left(\sigma'\left(z_i^l(v^l)\right)\right). \tag{24}$$

By defining $e_i^l(x) = \partial \mathrm{D(x)}/\partial h_i^l$ we get

$$e_i^l(x) = \frac{\partial D(x)}{\partial h_i^l} = \frac{\partial h^l}{\partial h_i^l}\frac{\partial D(x)}{\partial h^l}. \tag{25}$$

More specifically,

$$e_1^l(x) = \left(\left(h_2^l\right)^T \otimes I_{K_1^l}\right) e^l(x) = \left[e^l(x)\right]_{K_1^l, K_2^l} h_2^l \tag{26}$$

$$e_2^l = \left(I_{K_2^l} \otimes \left(h_1^l\right)^T\right) e^l(x) = \left(\left[e^l(x)\right]_{K_1^l, K_2^l}\right)^T h_1^l, \tag{27}$$

where $[e^l(x)]_{K_1^l, K_2^l}$ reshapes $e^l(x)$ to a $K_1^l \times K_2^l$ matrix. The gradients needed for BP algorithm in the DP layers are thus

$$\begin{aligned} \frac{\partial D(x)}{\partial w_i^l} &= \left[\frac{\partial D(x)}{\partial \mathrm{vec}\left(w_i^l\right)}\right]_{K_v^l, K_i^l} \\ &= \left[\frac{\partial h_i^l}{\partial \mathrm{vec}\left(w_i^l\right)}\frac{\partial D(x)}{\partial h_i^l}\right]_{K_v^l, K_i^l} \\ &= \left[\left(v^l \otimes \mathrm{diag}\left(\sigma'\left(z_i^l(v^l)\right)\right)\right) e_i^l(x)\right]_{K_v^l, K_i^l} \\ &= v^l \left(\mathrm{diag}\left(\sigma'\left(z_i^l(v^l)\right)\right) e_i^l(x)\right)^T, \end{aligned} \tag{28}$$

$$\frac{\partial D(x)}{\partial a_i^l} = \frac{\partial h_i^l}{\partial a_i^l}\frac{\partial D(x)}{\partial h_i^l} = \mathrm{diag}\left(\sigma'\left(z_i^l(v^l)\right)\right) e_i^l(x), \tag{29}$$

and

$$\begin{aligned} e^{l-1}(x) = \frac{\partial D(x)}{\partial v^l} &= \sum_{j\in\{1,2\}} \frac{\partial h_j^l}{\partial v^l}\frac{\partial D(x)}{\partial h_j^l} \\ &= \sum_{j\in\{1,2\}} \frac{\partial h_j^l}{\partial v^l} e_i^l(x) \\ &= \sum_{j\in\{1,2\}} w_i^l \mathrm{diag}\left(\sigma'\left(z_i^l(v^l)\right)\right) e_i^l(x). \end{aligned} \tag{30}$$

The learning algorithm of the quasi-DTNN is very similar to that of the DTNN derived and presented above. The main difference is that for the DP layers in the quasi-DTNN, the gradients now become

$$e_i^l(x) = e^l(x)\mathrm{diag}\left(\sigma'(h^l)\right) z_{1-i}^l(v^l), \tag{31}$$

$$\frac{\partial D(x)}{\partial w_i^l} = v^l \left(e_i^l(x)\right)^T, \tag{32}$$

$$\frac{\partial D(x)}{\partial a_i^l} = e_i^l(x), \tag{33}$$

and

$$e^{l-1}(x) = \sum_{j\in\{1,2\}} w_i^l e_i^l(x). \tag{34}$$

V. Experimental Results

In this section, we compare the DTNN with the conventional DNN on the MNIST handwritten digit recognition task and two Switchboard large vocabulary speech recognition tasks.

To specify a DTNN we use the notation of two numbers enclosed in a pair of parentheses to denote the size of the DP layer. As an example, (96:96) denotes a DP layer with 96 units in each of the two parts. Thus, $(64:64) \times 1 - 2\,\mathrm{k} \times 4$ denotes a DTNN that contains a DP layer with 64 units in each part, followed by 4 conventional sigmoid hidden layers each of which has 2 k units.

TABLE II
COMPARE SINGLE HIDDEN LAYER NEURAL NETWORKS WITH AND WITHOUT USING DOUBLE-PROJECTION AND TENSORS IN THE HIDDEN LAYER ON MNIST DATASET

Configuration	Test Error (%)
Normal 784-130-10	2.33±0.25
Tensor 784-(50:50)-10	1.88±0.21
Quasi-Tensor 784-(50:50)-10	1.86±0.23

A. MNIST Handwritten Digit Recognition Task

The MNIST dataset [23] contains binary images of handwritten digits. The digits have been size-normalized to fit in a 20×20 pixel box while preserving their aspect ratio and centered in a 28×28 image by computing and translating the center of mass of the pixels. 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, out of which we randomly selected 5,000 samples as the cross validation set. The test set has 10,000 patterns. The writers of the training set and test set are disjoint.

Our goal of using the MNIST dataset is to quickly check whether DP and tensor layers indeed have better modeling power than conventional sigmoid layers and to evaluate whether we should choose DTNNs or quasi-DTNNs. For this reason, we have used single hidden layer neural networks with a relatively small number of hidden units. More specifically, we have used a conventional shallow network with the configuration 784-130-10 and the tensor and quasi-tensor shallow networks with the configuration of 784-(50:50)-10. We chose these configurations to ensure that they have a similar number of parameters, which is $(784+1)\times 130+(130+1)\times 10 = 103.4\,\text{K}$ and $(784+1)\times(50+50)+(50\times 50+1)\times 10 = 103.5\,\text{K}$, respectively, for the 784-130-10 and 784-(50:50)-10 configurations.

We initialized weights randomly and ran 10 experiments on each configuration. The training was carried out using stochastic gradient ascent, taking a learning rate of 0.1 per sample for the first 5 sweeps and 0.05 per sample afterwards. The training stops when the error rate measured on the development set increases. The classification results are summarized in Table II. It is clear that both tensor and quasi-tensor layers help reduce the error rate over the conventional sigmoid hidden layers (shaded row in the table). Note that tensor and quasi-tensor layers give similar error rates on this same configuration. However, we have noticed that quasi-tensor layers are in general more likely to diverge in training if model parameters are not correctly initialized or the learning rate is not properly chosen. This is likely because multiplying two real valued numbers may send an unbounded learning signal. For this reason we apply only DTNNs to speech recognition tasks which take much more time to train.

B. SWB 30-hr Speech Recognition Task

The training and development sets in the SWB 30-hr task contain 30 hours and 6.5 hours of data randomly sampled from the 309-hour Switchboard-I training set. The 1831-segment SWB part of the NIST 2000 Hub5 evaluation set (6.5 hours) was used as the test set. To prevent speaker overlap between the training and test sets, speakers occurring in the test set were removed from the training and development sets.

TABLE III
COMPARING THE EFFECT OF DIFFERENT DTNN CONFIGURATIONS ON THE SWB 30-hr TASK. DTNNS WERE TRAINED FOR ONLY 10 SWEEPS, IN WHICH THE FIRST 5 SWEEPS WERE CARRIED OUT USING A LEARNING RATE OF 3×10^{-4} PER SAMPLE AND THE REMAINING 5 SWEEPS WITH A LEARNING RATE 8×10^{-6} PER SAMPLE

Configuration	Hub5'00 WER	Number Params
CD-GMM-HMM (BMMI)	34.8%	4.7M
DTNN: 429-(64:64)x1-2kx4-1504	31.0%	24.1M
DTNN: 429-(96:96)x5-1504	28.5%	21.0M
DNN: 429-2kx5-1504	28.3%	20.7M
JFDNN:429-2kx5-1504x7	28.2%	39.2M
DTNN: 429-2kx2-(64:64)x3-1504	27.9%	12.5M
DTNN: 429-2kx2-(64:64)x1-2kx2-1504	27.6%	21.0M
DTNN: 429-2kx2-(96:96)x3-1504	27.6%	22.9M
DTNN: 429-2kx4-(64:64)x1-1504	27.3%	19.9M
DTNN: 429-2kx4-(96:96)x1-1504	27.0%	27.7M

The system uses a 39-dimensional feature that was reduced using HLDA from mean- and variance-normalized 13-dimensional PLP features and up to third-order derivatives. The common left-to-right 3-state speaker-independent crossword triphones share 1504 CART-tied states determined on the conventional GMM system. The trigram language model (LM) was trained on the 2000 h Fisher-corpus transcripts and interpolated with a written text trigram. The test-set perplexity with a 58 k dictionary is 84. The features, lexicon and LM used in this study are the same as those used in our earlier work [4]–[6].

The GMM-HMM baseline system has a mixture of 40 Gaussians in each HMM state. It was trained with maximum likelihood (ML) and refined discriminatively with the boosted maximum-mutual-information (BMMI) criterion. Using more than 40 Gaussians did not improve the ML result.

Both the CD-DNN-HMM and CD-DTNN-HMM systems replace the Gaussian mixtures with scaled likelihoods derived from the DNN and DTNN posteriors, respectively. The input to the DNN and DTNN contains 11 (5-1-5) frames of the HLDA-transformed features. The baseline DNN uses the architecture of 429-2048 $\times$ 5-1504. A DTNN whose hidden layers are (96:96) $\times$ 5 has 21 million parameters, similar to the total number of parameters in the baseline conventional DNN.

The training was carried out with tied-triphone state labels generated using the ML-trained CD-GMM-HMM system. In our experiments, the conventional DNNs were pre-trained with the DBN-pretraining algorithm [24] before they were fine-tuned using the BP algorithm. However, we have not developed similar pretraining algorithms for DTNNs. DTNNs were thus trained using the BP algorithm presented in Section IV starting from randomly initialized weights. The pretrained DNN model typically outperforms the randomly initialized DNN model, with 0.3%–0.5% absolute WER reduction when the number of hidden layers is 5.

Table III compares the effect of different DTNN configurations on the recognition error rate. To reduce the overall training time we trained DTNNs for only 10 sweeps, in which the first 5 sweeps were carried out using a learning rate of 3×10^{-4} per sample and the remaining 5 sweeps with a learning rate of 8×10^{-6} per sample.

Note that even with this highly sub-optimal learning strategy, a DNN with 5 hidden layers (shaded row in the table) already significantly outperforms the CD-GMM-HMM trained using the BMMI criterion. The results in Table III are organized so that all configurations above the shaded line underperform the conventional DNN and all the configurations below the shaded line outperform DNN.

Examining Table III, we can make three observations. First, configuration (96:96) $\times$ 5 in which all layers are DP tensor layers performs similarly to the DNN baseline that contains a similar number of parameters, even though the DNN was pre-trained while the DTNN was not. Note that due to the nature of the DP layer, the dimension of the hidden layers in the DTNN is much smaller (under two hundred) than comparable conventional layers (a few thousand). Second, the configuration in which only the bottom (first) layer was replaced with the DP layer (configuration $429 - (64 : 64) \times 1 - 2\text{ k} \times 4 - 1504$) performs the worst. We believe this is because much of the information in the real-valued input is lost when the input feature is transformed into a $64 + 64 = 128$ (much smaller than 2048 in the conventional DNN) dimension DP layer. Third, the configurations that replace the top hidden layer with the DP layer (configurations $429 - 2\text{ k} \times 4 - (64 : 64) \times 1 - 1504$ and $429 - 2\text{ k} \times 4 - (96 : 96) \times 1 - 1504$) perform the best and achieve more than 5% relative WER reduction over the DNN. This is because the top hidden layer is more invariant than the input layer and thus the information loss caused by using the low-dimensional DP layer is outweighed by the benefit obtained by using the tensor layer. The DTNN in which only the middle hidden layer is a DP layer (configuration $429 - 2\text{ k} \times 2 - (64 : 64) \times 1 - 2\text{ k} \times 2 - 1504$) performs in between.

In Table III we also included the results achieved with the joint factorized DNN (JFDNN) described in [21]. This is intended to answer the question of whether using a gated softmax layer [19] on top of a DNN is helpful. The experiment used $2^7 = 128$ factors in the gated softmax layer. It can be seen that the JFDNN only slightly outperforms the conventional DNN but with much longer training time.

To eliminate the possibility that the training strategy adopted in Table III may favor DTNNs over DNNs, we tuned the learning strategy, including learning rates and schedule, for DNNs and used this tuned learning strategy to train DTNNs. More specifically, DNNs and DTNNs were trained for 15 sweeps, in which the first 9 sweeps were carried out using a learning rate of 3×10^{-4} per sample and the remaining 6 sweeps with a learning rate 8×10^{-6} per sample. Further increasing the training sweeps does not lead to additional gain on the development set. In addition, we have compared DNNs and DTNNs with 7-hidden layers. The new results are summarized in Table IV. These results further confirm the effectiveness of the DTNN, with 1.2% and 1.0% absolute, or 4.4% and 3.9% relative, WER reduction over the DNNs, respectively, for the five and seven-hidden layer systems.

C. SWB 309-hr Speech Recognition Task

In the SWB 309-hr task, we used the 309-hour Switchboard-I training set [14]. The feature extraction process is exactly the same as that described in Section V-B. However, the optimal

TABLE IV
COMPARING DNN AND DIFFERENT CONFIGURATIONS OF DTNN ON THE SWB 30-hr TASK. THE LEARNING STRATEGY WAS TUNED FOR DNN AND APPLIED TO DTNN

Configuration	Hub5'00 WER
CD-GMM-HMM (BMMI)	34.8%
DNN: 429-2kx5-1504	27.4%
DTNN: 429-2kx2-(64:64)x1-2kx2-1504	26.8%
DTNN: 429-2kx4-(64:64)x1-1504	26.4%
DTNN: 429-2kx4-(96:96)x1-1504	26.2%
DNN: 429-2kx7-1504	25.7%
DTNN: 429-2kx6-(64:64)x1-1504	24.8%
DTNN: 429-2kx6-(96:96)x1-1504	24.7%

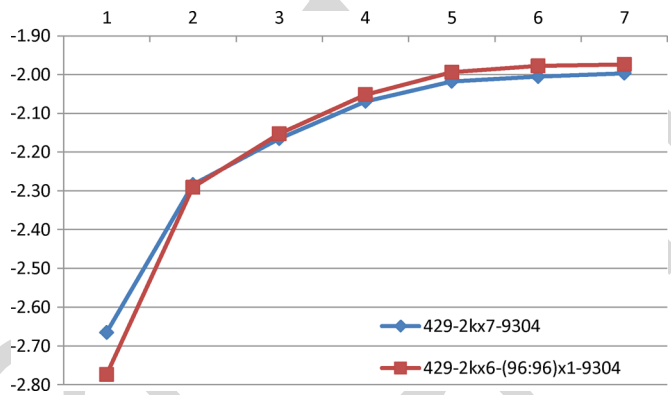


Fig. 4. The change of training set frame-level cross entropy after each sweep of the 309-hr training set.

number of CART-tied triphone states determined by the GMM system is now increased to 9304. We followed the same procedure as described in [4], [5] to train the conventional CD-DNN-HMM with the tied-triphone state alignment generated using the ML-trained CD-GMM-HMM. More specifically, we swept the training data seven times. We used a learning rate of 3×10^{-4} per sample for the first three sweeps and 8×10^{-6} per sample for the remaining four sweeps. The conventional DNN was pre-trained generatively using the DBN-pretraining algorithm, but the DTNN was not, although the discriminative pretraining procedure introduced in [5] could be used. To prevent divergence, we have used a minibatch size of 128 for the first sweep and 1024 afterwards. To investigate the generalization ability we tested the model on the 6.3 h Spring 2003 NIST rich transcription set (RT03S) in addition to the Hub5'00 evaluation set. Different from the best results achieved in [4] which used DNN realignment, the results presented here used only the alignment generated from the GMM-ML system.

Fig. 4 and Fig. 5 illustrate the training set frame-level cross-entropy (CE) and senone prediction accuracy, respectively, over sweeps of the 309-hr training data. It can be seen that initially the DTNN performs worse than the conventional DNN since the weights were not pretrained. However, after three sweeps, the DTNN made up the difference and eventually outperformed the DNN.

Table V summarizes the word error rate (WER) on this task using DNN and DTNN. From Table V we can see that the DTNN still outperforms the DNN, but the gain is smaller with 0.5% absolute or 3% relative WER reduction on the Hub5'00 eval set. This is possibly because a DNN trained with significantly more data can generalize better even without explicit

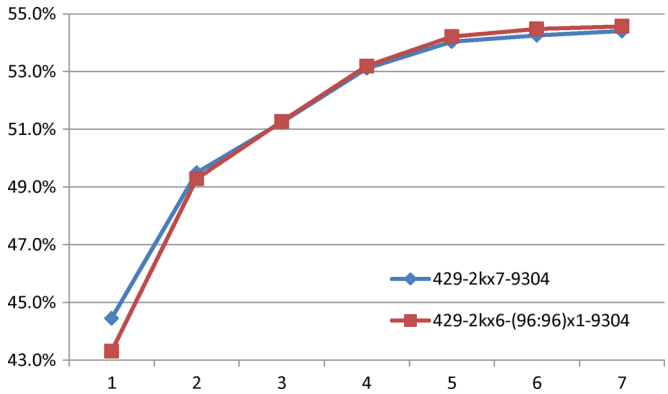


Fig. 5. The change of training set frame-level senone classification accuracy after each sweep of the 309-hr training set.

TABLE V
COMPARING DNN AND DTNN ON THE SWB 309-hr TASK. WER ON HUB5'00 AND RT03S EVALUATION SETS

Configuration	Hub5'00	RT03S	
		FSH	SW
CD-GMM-HMM (BMMI)	23.6%	27.4%	37.6%
DNN: 429-2kx7-9304	17.1%	19.6%	28.4%
DNN: 429-2kx9-9304	17.0%	19.0%	28.1%
DTNN: 429-2kx6-(96:96)x1-9304	16.6%	19.2%	27.9%

modeling of subspaces and their interactions as intended by the DTNN. Table V also indicates that when applied to the RT03S evaluation set, the DTNN outperforms the seven hidden layer DNN with a 0.4% and 0.5% WER reduction on the FSH and SW parts, respectively. Compared to the nine-hidden-layer DNN it performs slightly better on the Hub5'00 evaluation set and SW part of the RT03S set, but slightly worse on the FSH part of the RT03S set.

VI. SUMMARY AND CONCLUSIONS

In this paper we have proposed and implemented a novel deep neural network, the DTNN, which involves tensor interactions among neurons. This work is in part motivated by tensor network theory in neuroscience, where tensor interactions play a role in the central nervous system (e.g., [15]).

In a DTNN, at least one layer in the deep architecture is composed of a DP and a tensor layer. The two subspaces represented by the two parts in the DP layer interact with each other to cover a product space. We have described an approach to map the tensor layers to conventional sigmoid layers so that the former can be treated and trained in a similar way to the latter. With this mapping we can consider a DTNN as a DNN augmented with DP layers. As a result, the BP learning algorithm for DTNNs can be cleanly derived as we presented in Section IV of this paper.

In addition, we have described how the DP and tensor layers can stack up to form a DTNN in which all layers are DP and tensor layers. We have also showed how two variants of the DTNN can be constructed and their weight parameters learned.

We have evaluated different configurations of the DTNN architecture on the MNIST digit recognition task and on two SWB tasks using 30 and 309 hours of training data, respectively. The experimental results demonstrate that when the DP layer is placed at the top hidden layer of the DTNN, it performs the best and it outperforms the corresponding DNN by 4%–5% relative WER reduction on the 30-hr SWB task and 3% on the 309-hr SWB task. Our experiments suggest that the proposed DTNN is especially effective when the training data size is small.

In this work, we have discovered that DTNN is a very powerful deep architecture capable of representing covariance structure of the data in the hidden space and thus may show its potential in modeling noisy speech or speech with high variability. As our future work, we will investigate to what degree the use of speaker adapted features as the input to a DTNN would shrink the gain from using the DTNN over the regular DNN. On the other hand, we have noticed that having small DP layers may hurt the performance especially when the DP layer is at the bottom. However, increasing the DP layer size may significantly increase the overall model size and thus introduce overfitting problems. A possible solution is to factorize the weight tensor using the techniques adopted in [16], [19] to reduce the number of parameters.

ACKNOWLEDGMENT

We would like to thank Brian Hutchinson at The University of Washington for valuable discussions when he was an intern at Microsoft Research.

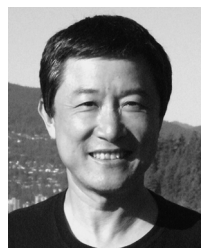
REFERENCES

[1] D. Yu, L. Deng, and F. Seide, "Large vocabulary speech recognition using deep tensor neural networks," in *Proc. Interspeech '12*.
[2] D. Yu, L. Deng, and G. Dahl, "Roles of pretraining and fine-tuning in context-dependent DBN-HMMs for real-world speech recognition," in *Proc. NIPS Workshop Deep Learn. Unsupervised Feature Learn.*, 2010.
[3] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large vocabulary speech recognition," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 20, no. 1, pp. 33–42, Jan. 2012.
[4] F. Seide, G. Li, and D. Yu, "Conversational speech transcription using context-dependent deep neural networks," in *Proc. Interspeech '11*, pp. 437–440.
[5] F. Seide, G. Li, X. Chen, and D. Yu, "Feature engineering in context-dependent deep neural networks for conversational speech transcription," in *Proc. ASRU '11*, pp. 24–29.
[6] D. Yu, F. Seide, G. Li, and L. Deng, "Exploiting sparseness in deep neural networks for large vocabulary speech recognition," in *Proc. ICASSP*, Mar. 2012, pp. 4409–4412.
[7] N. Jaitly, P. Nguyen, A. Senior, and V. Vanhoucke, An application of pretrained deep neural networks to large vocabulary conversational speech recognition Dept. of Comput. Sci., Univ. of Toronto, Tech. Rep. 001, 2012.
[8] T. N. Sainath, B. Kingsbury, and B. Ramabhadran, "Improvements in using deep belief networks for large vocabulary continuous speech recognition," Speech and Language Algorithm Group, IBM, Tech. Rep. UTML TR 2010-003, Feb. 2011, Tech. Rep..
[9] T. N. Sainath, B. Kingsbury, B. Ramabhadran, P. Fousek, P. Novak, and A.-R. Mohamed, "Making deep belief networks effective for large vocabulary continuous speech recognition," in *ASRU '11*, pp. 30–35.
[10] S. Renals, N. Morgan, H. Bourlard, M. Cohen, and H. Franco, "Connectionist probability estimators in HMM speech recognition," *IEEE Trans. Speech Audio Process.*, vol. 2, no. 1, pp. 161–174, Jan. 1994.
[11] A. Mohamed, G. E. Dahl, and G. E. Hinton, "Acoustic modeling using deep belief networks," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 20, no. 1, pp. 14–22, Jan. 2012.
[12] A. Mohamed, D. Yu, and L. Deng, "Investigation of full-sequence training of deep belief networks for speech recognition," in *Proc. Interspeech '10*, pp. 1692–1695.
[13] D. Yu, Y. C. Ju, Y. Y. Wang, G. Zweig, and A. Acero, "Automated directory assistance system—From theory to practice," in *Proc. Interspeech*, 2007, pp. 2709–2711.

[14] J. Godfrey and E. Holliman, "Switchboard-1 release 2," Linguistic Data Consortium. Philadelphia, PA, 1997.
[15] A. Pellionisz and R. Llinas, "Tensor network theory of the meta organization of functional geometries in the central nervous system," *Neuroscience*, vol. 16, pp. 245–273, 1985.
[16] M. Ranzato, A. Krizhevsky, and G. Hinton, "Factored 3-way restricted Boltzmann machines for modeling natural images," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2010, vol. 13.
[17] M. Ranzato and G. Hinton, "Modeling pixel means and covariances using factorized third-order Boltzmann machines," in *Proc. Comput. Vis. Pattern Recognit. Conf. (CVPR '10)*, 2010, pp. 2551–2558.
[18] I. Sutskever, J. Martens, and G. Hinton, "Generating text with recurrent neural networks," in *Proc. ICML*, 2011.
[19] R. Memisevic, C. Zach, G. Hinton, and M. Pollefeys, "Gated softmax classification," in *Proc. NIPS '11*.
[20] B. Hutchinson, L. Deng, and D. Yu, "A deep architecture with bilinear modeling of hidden representations: Applications to phonetic recognition," in *Proc. ICASSP '12*, 2012, pp. 4805–4508.
[21] D. Yu, X. Chen, and L. Deng, "Factorized deep neural networks for adaptive speech recognition," in *Proc. Int. Workshop Statist. Mach. Learn. Speech Process.*, 2012.
[22] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Rev.*, vol. 51, no. 3, pp. 455–500, Sep. 2009.
[23] 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, Nov. 1998.
[24] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

Dong Yu (M'97–SM'06) joined Microsoft Corporation in 1998 and Microsoft Speech Research Group in 2002, where he is currently 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, machine learning, and pattern recognition. He has published over 100 papers in these areas and is the inventor/coinventor of more than 40 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).



Li Deng (M'87–SM'92–F'04) received the Ph.D. from the University of Wisconsin-Madison. He was an Assistant (1989–1992), Associate (1992–1996), and Full Professor (1996–1999) at the University of Waterloo, Ontario, Canada. He then joined Microsoft Research, Redmond, where he is currently a Principal Researcher and where he received Microsoft Research Technology Transfer, Goldstar, and Achievement Awards. Prior to MSR, he also worked or taught at Massachusetts Institute of Technology, ATR Interpreting Telecom. Research Lab. (Kyoto, Japan), and HKUST. He has published over 300 refereed papers in leading journals/conferences and 3 books covering broad areas of human language technology, machine learning, and audio, speech, and signal processing. He is a Fellow of the Acoustical Society of America, a Fellow of the IEEE, and a Fellow of the International Speech Communication Association. He is an inventor or co-inventor of over 50 granted US, Japanese, or international patents. He served on the Board of Governors of the IEEE Signal Processing Society (2008–2010). More recently, he served as Editor-in-Chief for *IEEE Signal Processing Magazine* (2009–2011), which, according to the Thompson Reuters Journal Citation Report released 2010 and 2011, ranks first in both years among all 127 IEEE publications and all 247 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 IEEE TRANSACTIONS ON AUDIO, SPEECH AND LANGUAGE PROCESSING.

Frank Seide (M'97) is a Senior Researcher and Research Manager at Microsoft Research Asia, Beijing, responsible for Research efforts on transcription of phone calls and voicemail and content-based indexing of video and audio.

Frank was born in Hamburg, Germany. In 1993, he received a Master degree in electrical engineering from University of Technology of Hamburg-Harburg. From 1993–97, Frank worked at the speech research group of Philips Research in Aachen, Germany, on spoken-dialogue systems. He then transferred to Taiwan as one of the founding members of Philips Research East-Asia, Taipei, to lead a research project on Mandarin speech recognition. In June 2001, he joined the speech group at Microsoft Research Asia, initially as a Researcher, since 2003 as Project Leader for offline speech applications, and since October 2006 as Research Manager.

The Deep Tensor Neural Network With Applications to Large Vocabulary Speech Recognition

Dong Yu, *Senior Member, IEEE*, Li Deng, *Fellow, IEEE*, and Frank Seide, *Member, IEEE*

***Abstract*—The recently proposed context-dependent deep neural network hidden Markov models (CD-DNN-HMMs) have been proved highly promising for large vocabulary speech recognition. In this paper, we develop a more advanced type of DNN, which we call the deep tensor neural network (DTNN). The DTNN extends the conventional DNN by replacing one or more of its layers with a double-projection (DP) layer, in which each input vector is projected into two nonlinear subspaces, and a tensor layer, in which two subspace projections interact with each other and jointly predict the next layer in the deep architecture. In addition, we describe an approach to map the tensor layers to the conventional sigmoid layers so that the former can be treated and trained in a similar way to the latter. With this mapping we can consider a DTNN as the DNN augmented with DP layers so that not only the BP learning algorithm of DTNNs can be cleanly derived but also new types of DTNNs can be more easily developed. Evaluation on Switchboard tasks indicates that DTNNs can outperform the already high-performing DNNs with 4–5% and 3% relative word error reduction, respectively, using 30-hr and 309-hr training sets.**

***Index Terms*—Automatic speech recognition, CD-DNN-HMM, large vocabulary, tensor deep neural networks.**

I. Introduction

RECENTLY, the context-dependent deep neural network hidden Markov model (CD-DNN-HMM) was developed for large vocabulary speech recognition (LVSR) and has been successfully applied to a variety of large scale tasks by a number of research groups worldwide [2]–[9]. The CD-DNN-HMM adopts and extends the earlier artificial neural network (ANN) HMM hybrid system framework [10]–[12]. In CD-DNN-HMMs, DNNs—multilayer perceptrons (MLPs) with many hidden layers—replace Gaussian mixture models (GMMs) and directly approximate the emission probabilities of the tied triphone states (also called senones). In the first set of successful experiments, CD-DNN-HMMs were shown to achieve 16% [2], [3] and 33% [4]–[6] relative recognition error reduction over strong, discriminatively trained CD-GMM-HMMs, respectively, on a large-vocabulary voice search (VS) task [13] and the Switchboard (SWB) phone-call transcription task [14]. Subsequent work on Google voice search and YouTube data [7] and on Broadcast News [8], [9] confirmed the effectiveness of the CD-DNN-HMMs for large vocabulary speech recognition.

In this work, we extend the DNN to a novel deep tensor neural network (DTNN) in which one or more layers are double-projection (DP) and tensor layers (see Section III for the explanation). The basic idea of the DTNN comes from the motivation and assumption that the underlying factors, such as the spoken words, the speaker identity, noise and channel distortion, and so on, which affect the observed acoustic signals of speech can be factorized and be approximately represented as interactions between two nonlinear subspaces. This type of multi-way interaction was hypothesized and explored in neuroscience as a model for the central nervous system [15], which conceptually features brain function as comprising functional geometries via metric tensors in the internal central nervous system representation-spaces, both in sensorimotor and connected manifolds.

In DTNN, we represent the hidden, underlying factors by projecting the input onto two separate subspaces through a double-projection (DP) layer in the otherwise conventional DNN. We subsequently model the interactions among these two subspaces and the output neurons through a tensor with three-way connections. We propose a novel approach to reduce the tensor layer to a conventional sigmoid layer so that the model can be better understood and the decoding and learning algorithms can be cleanly developed. Based on this reduction, we also introduce alternative types of DTNNs. We empirically compare the conventional DNN and the new DTNN on the MNIST handwritten digit recognition task and the SWB phone-call transcription task [14]. The experimental results demonstrate that the DTNN generally outperforms the conventional DNN.

This paper is organized as follows. We briefly review the related work in Section II and introduce the general architecture of the DTNN in Section III, in which the detailed components of the DTNN and the forward computations are also described. Section IV is dedicated to the algorithms we developed in this work for learning DTNN weight matrices and tensors. The experimental results on MNIST digit recognition task and SWB task are presented and analyzed in Section V. We conclude the paper in Section VI.

II. Related Work

In recent years, an extension from matrix to tensor has been proposed to model three-way interactions and to improve the

Manuscript received May 29, 2012; revised September 01, 2012 and October 24, 2012; accepted November 03, 2012. Date of publication nulldate; date of current version nulldate. This work significantly extends and completes the preliminary work described in [1]. The associate editor coordinating the review of this manuscript and approving it for publication was Mark J. F. Gales.

D. Yu and L. Deng are with Microsoft Research, Redmond, WA 98052 USA (e-mail: dongyu@microsoft.com; deng@microsoft.com).

F. Seide is with Microsoft Research Asia, Beijing 100080, China (e-mail: fseide@microsoft.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TASL.2012.2227738

modeling power of neural networks. In this section we briefly survey the related work.

The tensor-based restricted Boltzmann machine (RBM) was proposed to model three-way interactions among pixels in [16] and to model pixel means and covariances in [17]. This three-way RBM is different from ours in two ways. First, it is a pure generative model, although it may be discriminatively fine-tuned, while our DTNN is a discriminative model in nature. Second, the three-way RBM has only one hidden layer and the mechanism in its architecture design does not permit the use of tensor weights in more than one layer stacked one on top of another. In contrast, our DTNN is designed very differently, with the goal of naturally embedding the tensor weights in many stacked hidden layers.

The tensor-based RBM was later extended to the tensor recurrent neural network (RNN) [18]. The tensor-RNN as discussed in [18], however, is also mainly used as a generative model.

In a separate study reported in [19], a softmax layer was gated with a hidden factor layer and a tensor was subsequently used to model the joint interaction among the hidden factors, the inputs, and the labels. However, the gated softmax network investigated in [19] is a less effective shallow model with no additional hidden layer other than the hidden factor layer. In addition, the work of [19] adopted the mixture model which predicts the classes by summing over all possible hidden factor combinations. The DTNN that we will present in this paper, however, is a deep network and it predicts the upper layer directly through the tensor connections as shown in (2) in Section III.

More recent work [20] replaced the single sigmoid hidden layer with a tensor layer in a deep network where blocks of shallow networks are used to construct the stacking deep architecture and each block in the stacking network consists of only one hidden layer. In contrast, in the DTNN, there are many hidden layers, one after the other. In fact any sigmoid layer in the conventional DNN may be replaced with a tensor layer. While the technique of converting the tensor layer to a conventional sigmoid layer in [20] has motivated and facilitated the development of DTNN here, it is worth noting that the deep architecture in [20] had difficulty for large vocabulary speech recognition tasks since the output units are often limited to a moderate size due to the special requirement for convexity in part of the network. The DTNN presented in this paper is free from such a restriction, combining the virtue of DNN in handling large vocabulary speech recognition tasks that require large senone output units and the effective technique of handling tensors developed from [20].

The most recent work reported in [21] presented two versions of a tensor-based DNN. The first version extended the gated softmax network of [19] by incorporating the gated softmax layer into DNNs. However, much like the softmax network in [19], this version also used a mixture model. The second version that also explored tensors as proposed in [21] is closer to the DTNN to be described in this paper. The main difference is that the gating factor in [21] was estimated completely separately from the main network and was only applied at the output layer. In contrast, the DTNN integrates all estimation steps of all parameters including the gating factors in a single, consistent framework.

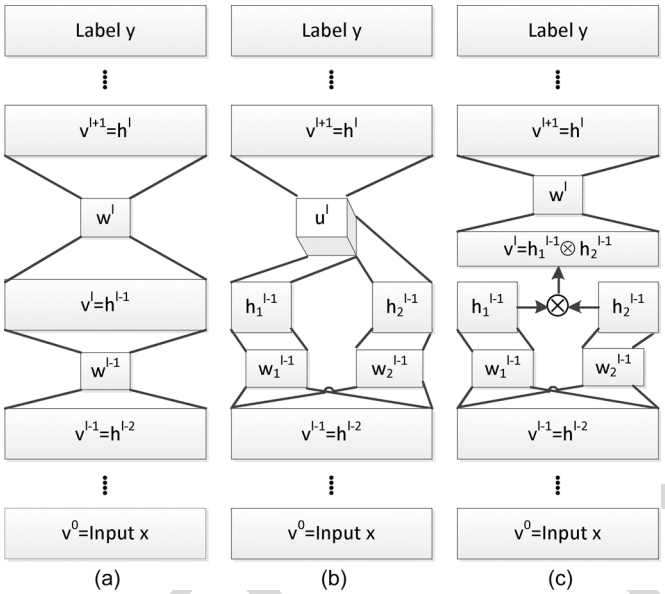


Fig. 1. Architectural illustrations of a conventional DNN and the corresponding DTNN. (a) DNN. (b) DTNN: hidden layer h^{l-1} consists of two parts: h_1^{l-1} and h_2^{l-1}. Hidden layer h^l is a tensor layer to which the connection weights u^l form a three-way tensor. (c) An alternative representation of (b): tensor u^l is replaced with matrix w^l when v^l is defined as the Kronecker product of h_1^{l-1} and h_2^{l-1}.

In summary, the DTNN presented in this paper differentiates itself from previous work in that we use DP layers to automatically factorize information which is later combined through the tensor layers. The distinction also lies in the more flexible incorporation of the DP layers and tensor layers into an otherwise conventional DNN architecture. In addition, our work provides a unified framework to train DNN, DTNN, and their variants (which we will call quasi-DTNN; see Fig. 3 in Section III-B) by mapping the input feature of each layer to a vector and the tensor to a matrix.

III. Architectures of the Deep Tensor Neural Network

The deep tensor neural network (DTNN) is a new type of DNN. It extends the conventional DNN by replacing one or more layers with double-projection and tensor layers, which we will define shortly. In this section we describe the general architecture of the DTNN.

A. DTNN With Double-Projection and Tensor Layers

Fig. 1 illustrates and compares the conventional DNN with the DTNN. Fig. 1(a) shows a conventional DNN, whose input is denoted by x, an $I \times 1$ vector, and the output is y, a $C \times 1$ vector. Subscript l is the layer index. In this conventional DNN, each hidden layer h^{l-1} connects to the next upper layer h^l through a weight matrix w^l and a bias a^l as

$$h^l_{(k)} = \sigma\left(\sum_i w^l_{(i,k)} h^{l-1}_{(i)} + a^l_{(k)}\right), \qquad (1)$$

where i, k are indexes of the hidden units in layers h^{l-1} and h^l, respectively, and $\sigma(x) = 1/(1+\exp(-x))$ is the sigmoid function applied element-wise.

Fig. 1(b) is the corresponding DTNN in which hidden layer h^{l-1} is separated into two parts: h_1^{l-1} (a $K_1^{l-1} \times 1$ vector) and h_2^{l-1} (a $K_2^{l-1} \times 1$ vector). These two parts connect to hidden layer h^l (a $K^l \times 1$ vector) through the three-way tensor u^l [22] of dimension $K_1^{l-1} \times K_2^{l-1} \times K^l$, which is represented with a cube in the figure, according to

$$h_{(k)}^l = \sigma\left(\sum_{i,j} u_{(i,j,k)}^l h_{1(i)}^{l-1} h_{2(j)}^{l-1} + a_{(k)}^l\right), \qquad (2)$$

where i, j, k are indexes of the hidden units in layers h_1^{l-1}, h_2^{l-1} and h^l, respectively. If h_1^{l-1} were to function as a speaker detector and h_2^{l-1} were to function as a spectrum pattern detector, (2) means that for different combinations of speaker $h_{1(i)}^{l-1}$ and spectrum pattern $h_{2(j)}^{l-1}$ a different weight $u_{(i,j,k)}^l$ is assigned for the same detector $h_{(k)}^l$ at next layer.

We call the hidden layer h^{l-1} a double-projection (DP) layer since the information from the previous layer h^{l-2} is projected into two separate subspaces at layer h^{l-1} as h_1^{l-1} and h_2^{l-1}. In this specific case, the DP layer h^{l-1} can be considered as a normal layer where

$$h^{l-1} = \begin{bmatrix} h_1^{l-1} \\ h_2^{l-1} \end{bmatrix} = \sigma\left(\begin{bmatrix} w_1^{l-1} \\ w_2^{l-1} \end{bmatrix} h^{l-2} + \begin{bmatrix} a_1^{l-1} \\ a_2^{l-1} \end{bmatrix}\right). \qquad (3)$$

Here w_1^{l-1} and w_2^{l-1} are weight matrices connecting the hidden layer h^{l-2} with the DP layer parts h_1^{l-1} and h_2^{l-1}, respectively, and a_1^{l-1} and a_2^{l-1} are the corresponding bias terms. As we will see later in this section, however, this is not required and actually is not the case if all layers are DP layers. The only requirement for the DP layer is that it is split into two parts.

The hidden layer h^l is called a tensor layer since the previous layer h^{l-1} is a DP layer that connects with h^l through the weight tensor u^l.

Fig. 1(c) is an alternative view of the same DTNN shown in Fig. 1(b). By defining v^l, the input to the layer l, as

$$v^l = \mathrm{vec}\left(h_1^{l-1} \otimes h_2^{l-1}\right) = \mathrm{vec}\left(h_1^{l-1}\left(h_2^{l-1}\right)^T\right), \qquad (4)$$

where $\otimes$ is the Kronecker product, and $\mathrm{vec}(\cdot)$ is the column-vectorized representation of the matrix, we can organize and rewrite tensor u^l into matrix w^l as represented by a rectangle in Fig. 1(c). In other words, we now have

$$h_{(k)}^l = \sigma\left(\sum_i w_{(i,k)}^l v_{(i)}^l + a_{(k)}^l\right). \qquad (5)$$

This rewriting allows us to reduce and convert *tensor* layers into conventional *matrix* layers and to define the same interface in describing these two different types of layers. For example, in Fig. 1(c) hidden layer h^l can now be considered as a conventional layer as in Fig. 1(a) and can be learned using the conventional backpropagation (BP) algorithm. This rewriting also indicates that the tensor layer can be considered as a conventional layer whose input comprises the cross product of the values passed from the previous layer.

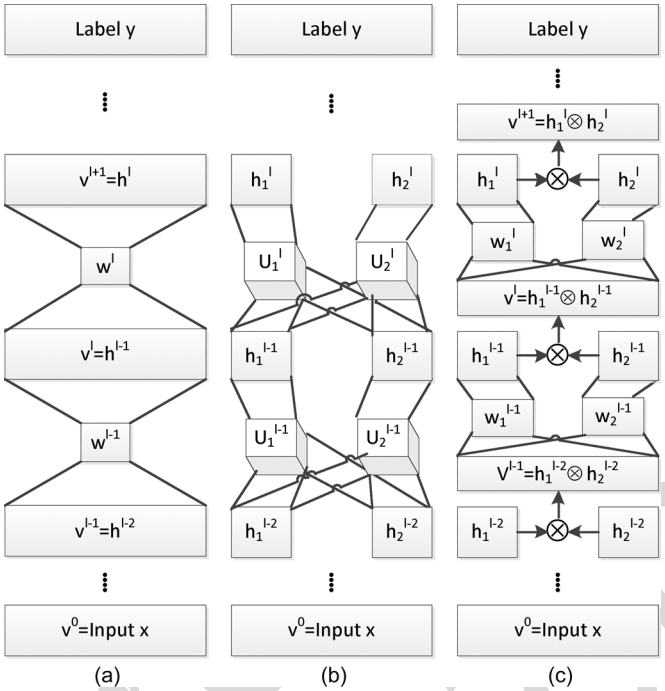


Fig. 2. Comparing conventional DNN and two equivalent views of a DTNN in which all hidden layers are DP tensor layers. (a) DNN; (b) DTNN: normal view where hidden layers are connected through three-way tensors; (c). DTNN: an alternative view where tensors are replaced with matrices when v^l is defined as the Kronecker product of h_1^{l-1} and h_2^{l-1}.

Hidden layer h^{l-1}, however, is still a DP layer that contains two output parts h_1^{l-1} and h_2^{l-1}, which in turn are determined by two separate weight matrices w_1^{l-1} and w_2^{l-1}, in the same way for Fig. 1(b) and Fig. 1(c).

The DTNN shown in Fig. 1 contains only one DP layer. However, nothing prevents other layers from being DP layers. Fig. 2(b) illustrates an example DTNN in which all hidden layers are DP tensor layers. For example, hidden layer h^{l-2} is also separated into two parts h_1^{l-2} and h_2^{l-2} and connects to h_1^{l-1} and h_2^{l-1} through tensors u_1^{l-1} and u_2^{l-1}, respectively. Note that in this DTNN each DP layer projects the input v^l onto two non-linear subspaces h_1^l and h_2^l. The bilinear interaction of these two projections is then combined as the input feature to the adjacent higher layer as quantified by (4). By defining input v^{l-1} to hidden layer h^{l-1} as

$$v^{l-1} = \mathrm{vec}\left(h_1^{l-2} \otimes h_2^{l-2}\right) = \mathrm{vec}\left(h_1^{l-2}\left(h_2^{l-2}\right)^T\right), \qquad (6)$$

tensors u_1^{l-1} and u_2^{l-1} can be rewritten as matrices w_1^{l-1} and w_2^{l-1} as shown in Fig. 2(c). Note that, although all the layers in Fig. 2(b) can be treated as non-tensor layers after this conversion, they are still DP layers since each layer contains two parts.

To summarize, we can represent DTNNs using two types of hidden layers: the conventional sigmoid layer and the DP layer. Each of these hidden layer types can be flexibly placed in the DTNN. For classification tasks the softmax layer that connects the final hidden layer to labels can be used in the DTNN, in the same way as that in the conventional DNN.

Table I summarizes all the forward computations involved in the DTNN, where the input is always converted and written as v^l, a $K_v^l \times 1$ column vector, w^l is the weight matrix, a^l is the

TABLE I
FORWARD COMPUTATIONS IN DTNNS

Condition	Input v^l
first layer	$v^l = v^0 = x$
h^{l-1}: normal layer	$v^l = h^{l-1}$
h^{l-1}: DP layer	$v^l = vec(h_1^{l-1} \otimes h_2^{l-1})$
Condition	**Output h^l**
normal layer	$h^l = \sigma\left(z^l(v^l)\right) = \sigma((w^l)^T v^l + a^l)$
DP layer, $i \in \{1,2\}$	$h_i^l = \sigma\left(z_i^l(v^l)\right) = \sigma\left(\left(w_i^l\right)^T v^l + a_i^l\right)$
softmax layer	$p(y\|v^L) = \frac{exp\left(\left(w_y^L\right)^T v^L + a_y^L\right)}{\Sigma_{y'} exp\left(\left(w_{y'}^L\right)^T v^L + a_{y'}^L\right)}$

bias, w_y^L is a column vector of the softmax layer weight matrix w^L, and

$$z^l(v^l) = (w^l)^T v^l + a^l \tag{7}$$

is the activation vector given input v^l.

Note that for the DP layer, the output has two parts

$$h_i^l = \sigma\left(z_i^l(v^l)\right) = \sigma\left(\left(w_i^l\right)^T v^l + a_i^l\right), \tag{8}$$

where $i \in \{1, 2\}$ indexes the part number. The two hidden layer vectors h_1^{l-1} and h_2^{l-1} may be augmented with ones when generating the input v^l of the next layer. However, this is unnecessary since the same effect may be achieved by setting weights to 0 and biases to a large positive number for one of the units so that it always outputs 1.

B. Variants of DTNN

The basic DTNN architecture described above can have a number of variants, and we describe two of them here. Fig. 3(a) shows a DTNN variant in which linear activations (i.e., no sigmoid nonlinearity) z_1^{l-1} and z_2^{l-1} are directly connected to layer h^l through tensor u^l. Fig. 3(b) is the equivalent architecture where weight tensor u^l is converted into weight matrix w^l by defining

$$v^l = \text{vec}\left(z_1^{l-1} \otimes z_2^{l-1}\right) = \text{vec}\left(z_1^{l-1}\left(z_2^{l-1}\right)^T\right). \tag{9}$$

Note that the only difference between the architectures of Fig. 3(a), 3(b) and those of Fig. 1(b), 1(c) is that the latter uses a sigmoid non-linearity (as indicated by h_1^{l-1} and h_2^{l-1} instead of z_1^{l-1} and z_2^{l-1} in the DP layer) before connecting to the next layer. This provides numerical stability and also incorporates the former as a special case if the sigmoid function is restricted to the linear range.

Fig. 3(c) shows another variant of the DTNN in which linear DP layers are also used but v^l is redefined as

$$v^l = h^{l-1} = \sigma\left(\text{vec}\left(z_1^{l-1} \otimes z_2^{l-1}\right)\right). \tag{10}$$

The difference between this model and that illustrated in Fig. 1(b), 1(c) is that the sigmoid non-linearity is applied after the Kronecker product instead of being applied to the two individual parts of the DP layers. Strictly speaking, the architecture of Fig. 3(c), while also modeling the relations between two subspaces and their upper layer, is not a DTNN since we cannot rewrite and represent it using a tensor. For this reason, we refer to the architecture of Fig. 3(c) as a quasi-DTNN.

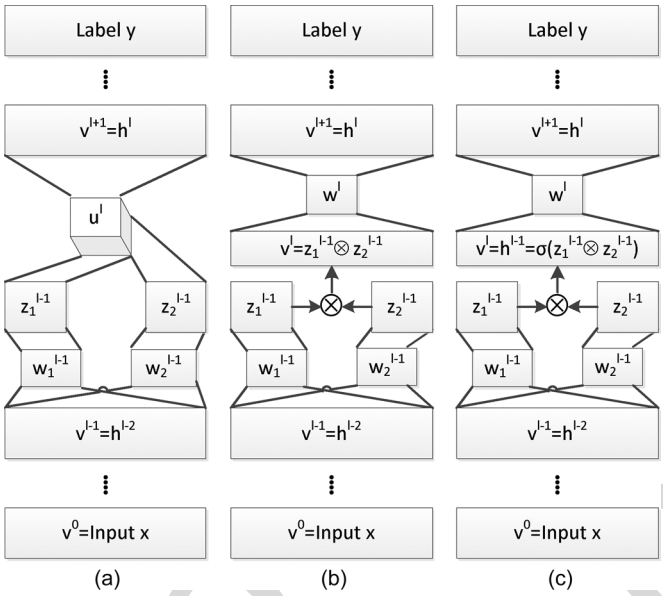


Fig. 3. Two additional types of DTNN. (a) DTNN in which the DP layer is linear (i.e., sigmoid function is not applied). (b) Alternative view of the same DTNN in (a). (c). a quasi-DTNN in which sigmoid non-linearity is applied to the Kronecker product of z_1^{l-1} and z_2^{l-1}. This model, although it models a three-way connection, cannot be represented using a tensor due to the sigmoid non-linearity applied to the Kronecker product of the two input components.

IV. LEARNING ALGORITHMS

We optimize the DTNN model parameters by maximizing the negative cross entropy

$$\overline{D} = \frac{1}{N}\sum_x D(x) = \frac{1}{N}\sum_x\sum_y \tilde{p}(y|x)\log p(y|x), \tag{11}$$

commonly used for the conventional DNN, where N is the total number of samples in the training set and $\tilde{p}(y|x)$ is the target probability. When a hard alignment is used $\tilde{p}(y = i|x)$ is 1 if the sample's training label is i and is 0 otherwise. Under that condition, the negative cross entropy is the same as the conditional log-likelihood. The parameters can be learned using the backpropagation (BP) algorithm.

The gradients associated with the softmax layer and the conventional sigmoid layers are the same as that in conventional DNNs. More specifically, for the softmax layer

$$\frac{\partial D(x)}{\partial w^L} = v^L\left(e^L(x)\right)^T, \tag{12}$$

$$\frac{\partial D(x)}{\partial a^L} = e^L(x), \tag{13}$$

where w^L is the $K_v^L \times C$ weight matrix, a^L is the $C \times 1$ bias column vector, and $e^L(x)$ is a $C \times 1$ error column vector with

$$e_i^L(x) = \left(\tilde{p}(y = i|x) - p(y = i|x)\right), \tag{14}$$

where $\tilde{p}(y=i|x)$ is the target probability and $p(y=i|x)$ is the model's predicted probability. For other layers with $l < L$ we define the error signal $e^l(x) = \partial \mathrm{D(x)}/\partial v^{l+1}$.

In the softmax layer, the error can be propagated to the immediately previous layer according to

$$e^{L-1}(\mathrm{x}) = \frac{\partial D(x)}{\partial v^L} = w^L e^L(x). \tag{15}$$

Similarly, for the conventional sigmoid layer, we have

$$\frac{\partial D(x)}{\partial w^l} = v^l \left(\mathrm{diag}\left(\sigma'\left(z^l(v^l)\right)\right) e^l(x)\right)^T, \tag{16}$$

$$\frac{\partial D(x)}{\partial a^l} = \mathrm{diag}\left(\sigma'\left(z^l(v^l)\right)\right) e^l(x), \tag{17}$$

and

$$e^{l-1}(x) = \frac{\partial D(x)}{\partial v^l} = w^l \mathrm{diag}\left(\sigma'\left(z^l(v^l)\right)\right) e^l(x), \tag{18}$$

where $\sigma'(x) = \sigma(x)(1-\sigma(x))$ is the gradient of the sigmoid function applied element-wise, and diag(.) is the diagonal matrix determined by the operand.

However, the gradients are more complicated for the DP layers, which we derive now. Note for the DP layer we have

$$\begin{aligned} v^{l+1} &= \mathrm{vec}\left(h_1^l \left(h_2^l\right)^T\right) \\ &= \left(h_2^l \otimes I_{K_1^l}\right) \mathrm{vec}\left(h_1^l\right) = \left(h_2^l \otimes I_{K_1^l}\right) h_1^l \\ &= \left(I_{K_2^l} \otimes h_1^l\right) \mathrm{vec}\left(\left(h_2^l\right)^T\right) = \left(I_{K_2^l} \otimes h_1^l\right) h_2^l, \end{aligned} \tag{19}$$

where I_K is a $K \times K$ identity matrix. v^{l+1} is thus a $(K_1^l \times K_2^l) \times 1$ column vector whose elements are $v^l_{(j+k\cdot K_1^l)} = h^l_{1,j} h^l_{2,k}$, where we assume matrix and vector index is 0 based. This leads to the gradients

$$\frac{\partial (v^{l+1})^T}{\partial h_1^l} = \frac{\partial \left(\left(h_2^l \otimes I_{K_1^l}\right) h_1^l\right)^T}{\partial h_1^l} = \left(h_2^l\right)^T \otimes I_{K_1^l}, \tag{20}$$

whose $(i, j+k\cdot K_1^l)$-th element is $\delta(i=j)h^l_{2,k}$, and

$$\frac{\partial (v^{l+1})^T}{\partial h_2^l} = \frac{\partial \left(\left(I_{K_2^l} \otimes h_1^l\right) h_2^l\right)^T}{\partial h_2^l} = I_{K_2^l} \otimes \left(h_1^l\right)^T, \tag{21}$$

whose $(i, j+k\cdot K_1^l)$-th element is $\delta(i=k)h^l_{1,j}$.

Note that for the parts $i \in \{1,2\}$

$$\frac{\partial \left(h_i^l\right)^T}{\partial vec\left(w_i^l\right)} = v^l \otimes \mathrm{diag}\left(\sigma'\left(z_i^l(v^l)\right)\right), \tag{22}$$

$$\frac{\partial \left(h_i^l\right)^T}{\partial a_i^l} = \mathrm{diag}\left(\sigma'\left(z_i^l(v^l)\right)\right), \tag{23}$$

and

$$\frac{\partial \left(h_i^l\right)^T}{\partial v^l} = w_i^l \mathrm{diag}\left(\sigma'\left(z_i^l(v^l)\right)\right). \tag{24}$$

By defining $e_i^l(x) = \partial \mathrm{D(x)}/\partial h_i^l$ we get

$$e_i^l(x) = \frac{\partial D(x)}{\partial h_i^l} = \frac{\partial h^l}{\partial h_i^l}\frac{\partial D(x)}{\partial h^l}. \tag{25}$$

More specifically,

$$e_1^l(x) = \left(\left(h_2^l\right)^T \otimes I_{K_1^l}\right) e^l(x) = \left[e^l(x)\right]_{K_1^l, K_2^l} h_2^l \tag{26}$$

$$e_2^l = \left(I_{K_2^l} \otimes \left(h_1^l\right)^T\right) e^l(x) = \left(\left[e^l(x)\right]_{K_1^l, K_2^l}\right)^T h_1^l, \tag{27}$$

where $[e^l(x)]_{K_1^l, K_2^l}$ reshapes $e^l(x)$ to a $K_1^l \times K_2^l$ matrix. The gradients needed for BP algorithm in the DP layers are thus

$$\begin{aligned} \frac{\partial D(x)}{\partial w_i^l} &= \left[\frac{\partial D(x)}{\partial \mathrm{vec}\left(w_i^l\right)}\right]_{K_v^l, K_i^l} \\ &= \left[\frac{\partial h_i^l}{\partial \mathrm{vec}\left(w_i^l\right)}\frac{\partial D(x)}{\partial h_i^l}\right]_{K_v^l, K_i^l} \\ &= \left[\left(v^l \otimes \mathrm{diag}\left(\sigma'\left(z_i^l(v^l)\right)\right)\right) e_i^l(x)\right]_{K_v^l, K_i^l} \\ &= v^l \left(\mathrm{diag}\left(\sigma'\left(z_i^l(v^l)\right)\right) e_i^l(x)\right)^T, \end{aligned} \tag{28}$$

$$\frac{\partial D(x)}{\partial a_i^l} = \frac{\partial h_i^l}{\partial a_i^l}\frac{\partial D(x)}{\partial h_i^l} = \mathrm{diag}\left(\sigma'\left(z_i^l(v^l)\right)\right) e_i^l(x), \tag{29}$$

and

$$\begin{aligned} e^{l-1}(x) = \frac{\partial D(x)}{\partial v^l} &= \sum_{j\in\{1,2\}} \frac{\partial h_j^l}{\partial v^l}\frac{\partial D(x)}{\partial h_j^l} \\ &= \sum_{j\in\{1,2\}} \frac{\partial h_j^l}{\partial v^l} e_i^l(x) \\ &= \sum_{j\in\{1,2\}} w_i^l \mathrm{diag}\left(\sigma'\left(z_i^l(v^l)\right)\right) e_i^l(x). \end{aligned} \tag{30}$$

The learning algorithm of the quasi-DTNN is very similar to that of the DTNN derived and presented above. The main difference is that for the DP layers in the quasi-DTNN, the gradients now become

$$e_i^l(x) = e^l(x)\mathrm{diag}\left(\sigma'(h^l)\right) z_{1-i}^l(v^l), \tag{31}$$

$$\frac{\partial D(x)}{\partial w_i^l} = v^l \left(e_i^l(x)\right)^T, \tag{32}$$

$$\frac{\partial D(x)}{\partial a_i^l} = e_i^l(x), \tag{33}$$

and

$$e^{l-1}(x) = \sum_{j\in\{1,2\}} w_i^l e_i^l(x). \tag{34}$$

V. Experimental Results

In this section, we compare the DTNN with the conventional DNN on the MNIST handwritten digit recognition task and two Switchboard large vocabulary speech recognition tasks.

To specify a DTNN we use the notation of two numbers enclosed in a pair of parentheses to denote the size of the DP layer. As an example, (96:96) denotes a DP layer with 96 units in each of the two parts. Thus, $(64:64) \times 1 - 2\,\mathrm{k} \times 4$ denotes a DTNN that contains a DP layer with 64 units in each part, followed by 4 conventional sigmoid hidden layers each of which has 2 k units.

TABLE II
COMPARE SINGLE HIDDEN LAYER NEURAL NETWORKS WITH AND WITHOUT USING DOUBLE-PROJECTION AND TENSORS IN THE HIDDEN LAYER ON MNIST DATASET

Configuration	Test Error (%)
Normal 784-130-10	2.33±0.25
Tensor 784-(50:50)-10	1.88±0.21
Quasi-Tensor 784-(50:50)-10	1.86±0.23

A. MNIST Handwritten Digit Recognition Task

The MNIST dataset [23] contains binary images of handwritten digits. The digits have been size-normalized to fit in a 20×20 pixel box while preserving their aspect ratio and centered in a 28×28 image by computing and translating the center of mass of the pixels. 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, out of which we randomly selected 5,000 samples as the cross validation set. The test set has 10,000 patterns. The writers of the training set and test set are disjoint.

Our goal of using the MNIST dataset is to quickly check whether DP and tensor layers indeed have better modeling power than conventional sigmoid layers and to evaluate whether we should choose DTNNs or quasi-DTNNs. For this reason, we have used single hidden layer neural networks with a relatively small number of hidden units. More specifically, we have used a conventional shallow network with the configuration 784-130-10 and the tensor and quasi-tensor shallow networks with the configuration of 784-(50:50)-10. We chose these configurations to ensure that they have a similar number of parameters, which is $(784+1)\times130+(130+1)\times10 = 103.4$ K and $(784+1)\times(50+50)+(50\times50+1)\times10 = 103.5$ K, respectively, for the 784-130-10 and 784-(50:50)-10 configurations.

We initialized weights randomly and ran 10 experiments on each configuration. The training was carried out using stochastic gradient ascent, taking a learning rate of 0.1 per sample for the first 5 sweeps and 0.05 per sample afterwards. The training stops when the error rate measured on the development set increases. The classification results are summarized in Table II. It is clear that both tensor and quasi-tensor layers help reduce the error rate over the conventional sigmoid hidden layers (shaded row in the table). Note that tensor and quasi-tensor layers give similar error rates on this same configuration. However, we have noticed that quasi-tensor layers are in general more likely to diverge in training if model parameters are not correctly initialized or the learning rate is not properly chosen. This is likely because multiplying two real valued numbers may send an unbounded learning signal. For this reason we apply only DTNNs to speech recognition tasks which take much more time to train.

B. SWB 30-hr Speech Recognition Task

The training and development sets in the SWB 30-hr task contain 30 hours and 6.5 hours of data randomly sampled from the 309-hour Switchboard-I training set. The 1831-segment SWB part of the NIST 2000 Hub5 evaluation set (6.5 hours) was used as the test set. To prevent speaker overlap between the training and test sets, speakers occurring in the test set were removed from the training and development sets.

TABLE III
COMPARING THE EFFECT OF DIFFERENT DTNN CONFIGURATIONS ON THE SWB 30-hr TASK. DTNNS WERE TRAINED FOR ONLY 10 SWEEPS, IN WHICH THE FIRST 5 SWEEPS WERE CARRIED OUT USING A LEARNING RATE OF 3×10^{-4} PER SAMPLE AND THE REMAINING 5 SWEEPS WITH A LEARNING RATE 8×10^{-6} PER SAMPLE

Configuration	Hub5'00 WER	Number Params
CD-GMM-HMM (BMMI)	34.8%	4.7M
DTNN: 429-(64:64)x1-2kx4-1504	31.0%	24.1M
DTNN: 429-(96:96)x5-1504	28.5%	21.0M
DNN: 429-2kx5-1504	28.3%	20.7M
JFDNN:429-2kx5-1504x7	28.2%	39.2M
DTNN: 429-2kx2-(64:64)x3-1504	27.9%	12.5M
DTNN: 429-2kx2-(64:64)x1-2kx2-1504	27.6%	21.0M
DTNN: 429-2kx2-(96:96)x3-1504	27.6%	22.9M
DTNN: 429-2kx4-(64:64)x1-1504	27.3%	19.9M
DTNN: 429-2kx4-(96:96)x1-1504	27.0%	27.7M

The system uses a 39-dimensional feature that was reduced using HLDA from mean- and variance-normalized 13-dimensional PLP features and up to third-order derivatives. The common left-to-right 3-state speaker-independent crossword triphones share 1504 CART-tied states determined on the conventional GMM system. The trigram language model (LM) was trained on the 2000 h Fisher-corpus transcripts and interpolated with a written text trigram. The test-set perplexity with a 58 k dictionary is 84. The features, lexicon and LM used in this study are the same as those used in our earlier work [4]–[6].

The GMM-HMM baseline system has a mixture of 40 Gaussians in each HMM state. It was trained with maximum likelihood (ML) and refined discriminatively with the boosted maximum-mutual-information (BMMI) criterion. Using more than 40 Gaussians did not improve the ML result.

Both the CD-DNN-HMM and CD-DTNN-HMM systems replace the Gaussian mixtures with scaled likelihoods derived from the DNN and DTNN posteriors, respectively. The input to the DNN and DTNN contains 11 (5-1-5) frames of the HLDA-transformed features. The baseline DNN uses the architecture of 429-2048 $\times$ 5-1504. A DTNN whose hidden layers are (96:96) $\times$ 5 has 21 million parameters, similar to the total number of parameters in the baseline conventional DNN.

The training was carried out with tied-triphone state labels generated using the ML-trained CD-GMM-HMM system. In our experiments, the conventional DNNs were pre-trained with the DBN-pretraining algorithm [24] before they were fine-tuned using the BP algorithm. However, we have not developed similar pretraining algorithms for DTNNs. DTNNs were thus trained using the BP algorithm presented in Section IV starting from randomly initialized weights. The pretrained DNN model typically outperforms the randomly initialized DNN model, with 0.3%–0.5% absolute WER reduction when the number of hidden layers is 5.

Table III compares the effect of different DTNN configurations on the recognition error rate. To reduce the overall training time we trained DTNNs for only 10 sweeps, in which the first 5 sweeps were carried out using a learning rate of 3×10^{-4} per sample and the remaining 5 sweeps with a learning rate of 8×10^{-6} per sample.

Note that even with this highly sub-optimal learning strategy, a DNN with 5 hidden layers (shaded row in the table) already significantly outperforms the CD-GMM-HMM trained using the BMMI criterion. The results in Table III are organized so that all configurations above the shaded line underperform the conventional DNN and all the configurations below the shaded line outperform DNN.

Examining Table III, we can make three observations. First, configuration (96:96) $\times$ 5 in which all layers are DP tensor layers performs similarly to the DNN baseline that contains a similar number of parameters, even though the DNN was pre-trained while the DTNN was not. Note that due to the nature of the DP layer, the dimension of the hidden layers in the DTNN is much smaller (under two hundred) than comparable conventional layers (a few thousand). Second, the configuration in which only the bottom (first) layer was replaced with the DP layer (configuration $429 - (64 : 64) \times 1 - 2\text{ k} \times 4 - 1504$) performs the worst. We believe this is because much of the information in the real-valued input is lost when the input feature is transformed into a $64 + 64 = 128$ (much smaller than 2048 in the conventional DNN) dimension DP layer. Third, the configurations that replace the top hidden layer with the DP layer (configurations $429 - 2\text{ k} \times 4 - (64 : 64) \times 1 - 1504$ and $429 - 2\text{ k} \times 4 - (96 : 96) \times 1 - 1504$) perform the best and achieve more than 5% relative WER reduction over the DNN. This is because the top hidden layer is more invariant than the input layer and thus the information loss caused by using the low-dimensional DP layer is outweighed by the benefit obtained by using the tensor layer. The DTNN in which only the middle hidden layer is a DP layer (configuration $429 - 2\text{ k} \times 2 - (64 : 64) \times 1 - 2\text{ k} \times 2 - 1504$) performs in between.

In Table III we also included the results achieved with the joint factorized DNN (JFDNN) described in [21]. This is intended to answer the question of whether using a gated softmax layer [19] on top of a DNN is helpful. The experiment used $2^7 = 128$ factors in the gated softmax layer. It can be seen that the JFDNN only slightly outperforms the conventional DNN but with much longer training time.

To eliminate the possibility that the training strategy adopted in Table III may favor DTNNs over DNNs, we tuned the learning strategy, including learning rates and schedule, for DNNs and used this tuned learning strategy to train DTNNs. More specifically, DNNs and DTNNs were trained for 15 sweeps, in which the first 9 sweeps were carried out using a learning rate of 3×10^{-4} per sample and the remaining 6 sweeps with a learning rate 8×10^{-6} per sample. Further increasing the training sweeps does not lead to additional gain on the development set. In addition, we have compared DNNs and DTNNs with 7-hidden layers. The new results are summarized in Table IV. These results further confirm the effectiveness of the DTNN, with 1.2% and 1.0% absolute, or 4.4% and 3.9% relative, WER reduction over the DNNs, respectively, for the five and seven-hidden layer systems.

C. SWB 309-hr Speech Recognition Task

In the SWB 309-hr task, we used the 309-hour Switchboard-I training set [14]. The feature extraction process is exactly the same as that described in Section V-B. However, the optimal number of CART-tied triphone states determined by the GMM system is now increased to 9304. We followed the same procedure as described in [4], [5] to train the conventional CD-DNN-HMM with the tied-triphone state alignment generated using the ML-trained CD-GMM-HMM. More specifically, we swept the training data seven times. We used a learning rate of 3×10^{-4} per sample for the first three sweeps and 8×10^{-6} per sample for the remaining four sweeps. The conventional DNN was pre-trained generatively using the DBN-pretraining algorithm, but the DTNN was not, although the discriminative pretraining procedure introduced in [5] could be used. To prevent divergence, we have used a minibatch size of 128 for the first sweep and 1024 afterwards. To investigate the generalization ability we tested the model on the 6.3 h Spring 2003 NIST rich transcription set (RT03S) in addition to the Hub5'00 evaluation set. Different from the best results achieved in [4] which used DNN realignment, the results presented here used only the alignment generated from the GMM-ML system.

TABLE IV
COMPARING DNN AND DIFFERENT CONFIGURATIONS OF DTNN ON THE SWB 30-hr TASK. THE LEARNING STRATEGY WAS TUNED FOR DNN AND APPLIED TO DTNN

Configuration	Hub5'00 WER
CD-GMM-HMM (BMMI)	34.8%
DNN: 429-2kx5-1504	27.4%
DTNN: 429-2kx2-(64:64)x1-2kx2-1504	26.8%
DTNN: 429-2kx4-(64:64)x1-1504	26.4%
DTNN: 429-2kx4-(96:96)x1-1504	26.2%
DNN: 429-2kx7-1504	25.7%
DTNN: 429-2kx6-(64:64)x1-1504	24.8%
DTNN: 429-2kx6-(96:96)x1-1504	24.7%

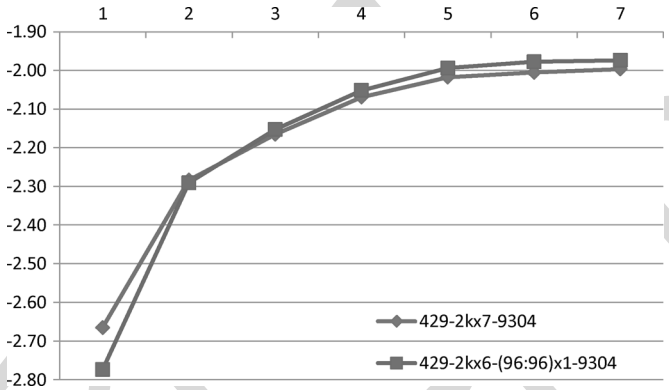


Fig. 4. The change of training set frame-level cross entropy after each sweep of the 309-hr training set.

Fig. 4 and Fig. 5 illustrate the training set frame-level cross-entropy (CE) and senone prediction accuracy, respectively, over sweeps of the 309-hr training data. It can be seen that initially the DTNN performs worse than the conventional DNN since the weights were not pretrained. However, after three sweeps, the DTNN made up the difference and eventually outperformed the DNN.

Table V summarizes the word error rate (WER) on this task using DNN and DTNN. From Table V we can see that the DTNN still outperforms the DNN, but the gain is smaller with 0.5% absolute or 3% relative WER reduction on the Hub5'00 eval set. This is possibly because a DNN trained with significantly more data can generalize better even without explicit

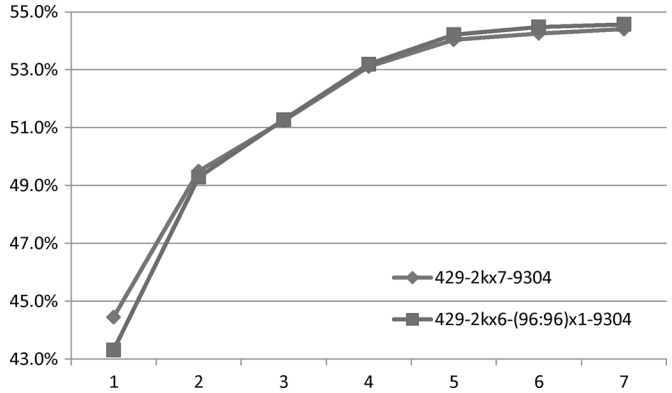


Fig. 5. The change of training set frame-level senone classification accuracy after each sweep of the 309-hr training set.

TABLE V
COMPARING DNN AND DTNN ON THE SWB 309-hr TASK. WER ON HUB5'00 AND RT03S EVALUATION SETS

Configuration	Hub5'00	RT03S	
		FSH	SW
CD-GMM-HMM (BMMI)	23.6%	27.4%	37.6%
DNN: 429-2kx7-9304	17.1%	19.6%	28.4%
DNN: 429-2kx9-9304	17.0%	19.0%	28.1%
DTNN: 429-2kx6-(96:96)x1-9304	16.6%	19.2%	27.9%

modeling of subspaces and their interactions as intended by the DTNN. Table V also indicates that when applied to the RT03S evaluation set, the DTNN outperforms the seven hidden layer DNN with a 0.4% and 0.5% WER reduction on the FSH and SW parts, respectively. Compared to the nine-hidden-layer DNN it performs slightly better on the Hub5'00 evaluation set and SW part of the RT03S set, but slightly worse on the FSH part of the RT03S set.

VI. SUMMARY AND CONCLUSIONS

In this paper we have proposed and implemented a novel deep neural network, the DTNN, which involves tensor interactions among neurons. This work is in part motivated by tensor network theory in neuroscience, where tensor interactions play a role in the central nervous system (e.g., [15]).

In a DTNN, at least one layer in the deep architecture is composed of a DP and a tensor layer. The two subspaces represented by the two parts in the DP layer interact with each other to cover a product space. We have described an approach to map the tensor layers to conventional sigmoid layers so that the former can be treated and trained in a similar way to the latter. With this mapping we can consider a DTNN as a DNN augmented with DP layers. As a result, the BP learning algorithm for DTNNs can be cleanly derived as we presented in Section IV of this paper.

In addition, we have described how the DP and tensor layers can stack up to form a DTNN in which all layers are DP and tensor layers. We have also showed how two variants of the DTNN can be constructed and their weight parameters learned.

We have evaluated different configurations of the DTNN architecture on the MNIST digit recognition task and on two SWB tasks using 30 and 309 hours of training data, respectively. The experimental results demonstrate that when the DP layer is placed at the top hidden layer of the DTNN, it performs the best and it outperforms the corresponding DNN by 4%–5% relative WER reduction on the 30-hr SWB task and 3% on the 309-hr SWB task. Our experiments suggest that the proposed DTNN is especially effective when the training data size is small.

In this work, we have discovered that DTNN is a very powerful deep architecture capable of representing covariance structure of the data in the hidden space and thus may show its potential in modeling noisy speech or speech with high variability. As our future work, we will investigate to what degree the use of speaker adapted features as the input to a DTNN would shrink the gain from using the DTNN over the regular DNN. On the other hand, we have noticed that having small DP layers may hurt the performance especially when the DP layer is at the bottom. However, increasing the DP layer size may significantly increase the overall model size and thus introduce overfitting problems. A possible solution is to factorize the weight tensor using the techniques adopted in [16], [19] to reduce the number of parameters.

ACKNOWLEDGMENT

We would like to thank Brian Hutchinson at The University of Washington for valuable discussions when he was an intern at Microsoft Research.

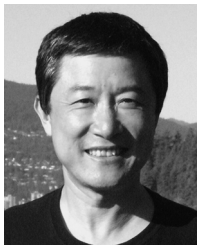
REFERENCES

[1] D. Yu, L. Deng, and F. Seide, "Large vocabulary speech recognition using deep tensor neural networks," in *Proc. Interspeech '12*.
[2] D. Yu, L. Deng, and G. Dahl, "Roles of pretraining and fine-tuning in context-dependent DBN-HMMs for real-world speech recognition," in *Proc. NIPS Workshop Deep Learn. Unsupervised Feature Learn.*, 2010.
[3] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large vocabulary speech recognition," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 20, no. 1, pp. 33–42, Jan. 2012.
[4] F. Seide, G. Li, and D. Yu, "Conversational speech transcription using context-dependent deep neural networks," in *Proc. Interspeech '11*, pp. 437–440.
[5] F. Seide, G. Li, X. Chen, and D. Yu, "Feature engineering in context-dependent deep neural networks for conversational speech transcription," in *Proc. ASRU '11*, pp. 24–29.
[6] D. Yu, F. Seide, G. Li, and L. Deng, "Exploiting sparseness in deep neural networks for large vocabulary speech recognition," in *Proc. ICASSP*, Mar. 2012, pp. 4409–4412.
[7] N. Jaitly, P. Nguyen, A. Senior, and V. Vanhoucke, An application of pretrained deep neural networks to large vocabulary conversational speech recognition Dept. of Comput. Sci., Univ. of Toronto, Tech. Rep. 001, 2012.
[8] T. N. Sainath, B. Kingsbury, and B. Ramabhadran, "Improvements in using deep belief networks for large vocabulary continuous speech recognition," Speech and Language Algorithm Group, IBM, Tech. Rep. UTML TR 2010-003, Feb. 2011, Tech. Rep..
[9] T. N. Sainath, B. Kingsbury, B. Ramabhadran, P. Fousek, P. Novak, and A.-R. Mohamed, "Making deep belief networks effective for large vocabulary continuous speech recognition," in *ASRU '11*, pp. 30–35.
[10] S. Renals, N. Morgan, H. Bourlard, M. Cohen, and H. Franco, "Connectionist probability estimators in HMM speech recognition," *IEEE Trans. Speech Audio Process.*, vol. 2, no. 1, pp. 161–174, Jan. 1994.
[11] A. Mohamed, G. E. Dahl, and G. E. Hinton, "Acoustic modeling using deep belief networks," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 20, no. 1, pp. 14–22, Jan. 2012.
[12] A. Mohamed, D. Yu, and L. Deng, "Investigation of full-sequence training of deep belief networks for speech recognition," in *Proc. Interspeech '10*, pp. 1692–1695.
[13] D. Yu, Y. C. Ju, Y. Y. Wang, G. Zweig, and A. Acero, "Automated directory assistance system—From theory to practice," in *Proc. Interspeech*, 2007, pp. 2709–2711.

[14] J. Godfrey and E. Holliman, "Switchboard-1 release 2," Linguistic Data Consortium. Philadelphia, PA, 1997.
[15] A. Pellionisz and R. Llinas, "Tensor network theory of the meta organization of functional geometries in the central nervous system," *Neuroscience*, vol. 16, pp. 245–273, 1985.
[16] M. Ranzato, A. Krizhevsky, and G. Hinton, "Factored 3-way restricted Boltzmann machines for modeling natural images," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2010, vol. 13.
[17] M. Ranzato and G. Hinton, "Modeling pixel means and covariances using factorized third-order Boltzmann machines," in *Proc. Comput. Vis. Pattern Recognit. Conf. (CVPR '10)*, 2010, pp. 2551–2558.
[18] I. Sutskever, J. Martens, and G. Hinton, "Generating text with recurrent neural networks," in *Proc. ICML*, 2011.
[19] R. Memisevic, C. Zach, G. Hinton, and M. Pollefeys, "Gated softmax classification," in *Proc. NIPS '11*.
[20] B. Hutchinson, L. Deng, and D. Yu, "A deep architecture with bilinear modeling of hidden representations: Applications to phonetic recognition," in *Proc. ICASSP '12*, 2012, pp. 4805–4508.
[21] D. Yu, X. Chen, and L. Deng, "Factorized deep neural networks for adaptive speech recognition," in *Proc. Int. Workshop Statist. Mach. Learn. Speech Process.*, 2012.
[22] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Rev.*, vol. 51, no. 3, pp. 455–500, Sep. 2009.
[23] 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, Nov. 1998.
[24] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

Dong Yu (M'97–SM'06) joined Microsoft Corporation in 1998 and Microsoft Speech Research Group in 2002, where he is currently 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, machine learning, and pattern recognition. He has published over 100 papers in these areas and is the inventor/coinventor of more than 40 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).



Li Deng (M'87–SM'92–F'04) received the Ph.D. from the University of Wisconsin-Madison. He was an Assistant (1989–1992), Associate (1992–1996), and Full Professor (1996–1999) at the University of Waterloo, Ontario, Canada. He then joined Microsoft Research, Redmond, where he is currently a Principal Researcher and where he received Microsoft Research Technology Transfer, Goldstar, and Achievement Awards. Prior to MSR, he also worked or taught at Massachusetts Institute of Technology, ATR Interpreting Telecom. Research Lab. (Kyoto, Japan), and HKUST. He has published over 300 refereed papers in leading journals/conferences and 3 books covering broad areas of human language technology, machine learning, and audio, speech, and signal processing. He is a Fellow of the Acoustical Society of America, a Fellow of the IEEE, and a Fellow of the International Speech Communication Association. He is an inventor or co-inventor of over 50 granted US, Japanese, or international patents. He served on the Board of Governors of the IEEE Signal Processing Society (2008–2010). More recently, he served as Editor-in-Chief for *IEEE Signal Processing Magazine* (2009–2011), which, according to the Thompson Reuters Journal Citation Report released 2010 and 2011, ranks first in both years among all 127 IEEE publications and all 247 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 IEEE TRANSACTIONS ON AUDIO, SPEECH AND LANGUAGE PROCESSING.

Frank Seide (M'97) is a Senior Researcher and Research Manager at Microsoft Research Asia, Beijing, responsible for Research efforts on transcription of phone calls and voicemail and content-based indexing of video and audio.

Frank was born in Hamburg, Germany. In 1993, he received a Master degree in electrical engineering from University of Technology of Hamburg-Harburg. From 1993–97, Frank worked at the speech research group of Philips Research in Aachen, Germany, on spoken-dialogue systems. He then transferred to Taiwan as one of the founding members of Philips Research East-Asia, Taipei, to lead a research project on Mandarin speech recognition. In June 2001, he joined the speech group at Microsoft Research Asia, initially as a Researcher, since 2003 as Project Leader for offline speech applications, and since October 2006 as Research Manager.