Tensor-Factorized Neural Networks

Jen-Tzung Chien, *Senior Member, IEEE*, and Yi-Ting Bao

***Abstract*— The growing interests in multiway data analysis and deep learning have drawn tensor factorization (TF) and neural network (NN) as the crucial topics. Conventionally, the NN model is estimated from a set of one-way observations. Such a vectorized NN is *not* generalized for learning the representation from multiway observations. The classification performance using vectorized NN is constrained, because the temporal or spatial information in neighboring ways is disregarded. More parameters are required to learn the complicated data structure. This paper presents a new tensor-factorized NN (TFNN), which tightly integrates TF and NN for multiway feature extraction and classification under a *unified discriminative* objective. This TFNN is seen as a generalized NN, where the affine transformation in an NN is replaced by the multilinear and multiway factorization for tensor-based NN. The multiway information is preserved through layerwise factorization. Tucker decomposition and nonlinear activation are performed in each hidden layer. The tensor-factorized error backpropagation is developed to train TFNN with the limited parameter size and computation time. This TFNN can be further extended to realize the convolutional TFNN (CTFNN) by looking at small subtensors through the factorized convolution. Experiments on real-world classification tasks demonstrate that TFNN and CTFNN attain substantial improvement when compared with an NN and a convolutional NN, respectively.**

***Index Terms*— Neural network (NN), pattern classification, tensor factorization (TF), and tensor-factorized error backpropagation.**

I. Introduction

TENSOR factorization (TF) and *deep learning* have been rapidly arising as the core technologies in various scientific fields ranging from psychology, chemistry, neuroscience, signal processing, computer vision, and bioinformatics to data mining. TF is developed as a versatile tool for multiway data reconstruction through factorizing a tensor or a multidimensional array with multiple modalities [1] in a least-squares sense. Multiple ways, such as trials, conditions, channels, spaces, times, and frequencies, are ubiquitous in real-world data collection. The tensor analysis can be used to extract the salient features from multilinear subspace according to the multilinear algebra [2]. Therefore, the multiway component analysis [3] was performed for blind source separation under a variety of data, including speech [4], [5], music [6], image [2], video, text, social network, chemical data, electroencephalography [7], magnetoencephalography, and functional magnetic resonance imaging, to name a few. In [4], the nonnegative TF (NTF) was implemented for single channel source separation. The frequency domain was incorporated as additional way in factorization procedure. Considering the multiway feature extraction, a multiview intact space learning [8] was proposed to discover the complementary features from multiple views based on a Cauchy loss with the desirable stability and generalization. In [9], a tensor canonical correlation analysis was developed to extract the multiview features by maximizing the canonical correlation of multiple views, where the higher order correlation instead of pairwise correlation was optimized. In [10], a tensor discriminant analysis was developed to extract the multiway discriminative features from input tensors by maximizing the tensor-variate Fisher criterion. Basically, the above-mentioned methods extracted the multiway features from different perspectives based on different objectives. These methods [8], [9] were designed for unsupervised feature representation, which did *not* involve classification problem.

On the other hand, deep learning based on the artificial neural network (NN) is run under a deep architecture, consisting of multiple hidden layers, which captures the high-level abstraction behind data and characterizes the complex nonlinear relationship between inputs and targets. Such a deep NN has been successfully applied for a number of regression and classification systems, including speech recognition [11], image classification [12], source separation, and natural language processing [13], [14]. However, a traditional NN was trained from a set of input *vectors*. For speech recognition, the data structure of a temporal-frequency *matrix* input was flattened into a high-dimensional vector. Two-way data structure was smeared. The contextual or structural information across different ways was disregarded. This paper deals with the classification problem, which involves high-dimensional observations with multiple ways. The observed tensors are factorized and forwarded through a multilayer perceptron (MLP), which is trained via the tensor-factorized error backpropagation.

A. Related Work

Several works have been proposed to strengthen the modeling capability from different perspectives via integration of TF and NN. In [15], a factored three-way restricted Botlzmann machine (RBM) allowed the factors or states of hidden units in the RBM to modulate the pairwise interactions between visible units in natural images. The weights in the RBM were parameterized as a tensor. In [13], the tensor-based RBM was extended to the tensor recurrent NN (RNN) for

Manuscript received June 24, 2016; revised October 27, 2016 and February 10, 2017; accepted March 26, 2017. This work was supported in part by the Ministry of Science and Technology, Taiwan, under Contract MOST 105-2221-E-009-137-MY2. *(Corresponding Author: Jen-Tzung Chien.)*

The authors are with the Department of Electrical and Computer Engineering, National Chiao Tung University, Hsinchu 30010, Taiwan (e-mail: jtchien@nctu.edu.tw).

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/TNNLS.2017.2690379

language modeling, where the hidden-to-hidden weights for different words were seen as a three-way tensor, which was factorized to reduce memory requirement. In [16], a deep tensor NN (TNN) was constructed by cascading the double-projection (DP) layer and the tensor layer, so as to learn the complimentary features from two separate hidden vectors. In [17], a bilinear transformation [18] was applied to form a TNN, which extracted the relational information of discrete entities in question answering.

In the above-mentioned models, the three-way tensor weights were estimated to capture the relation between features or neurons. The inputs and latent features were constrained as one-way vectors, which were fed into an NN through the tensor weights. These methods did not consider multiway inputs, which have been existing in many practical systems, where observations are acquired from different trials, subjects, channels, spaces, times, frequencies, and so on. Convolutional NN (CNN) [19]–[21] was developed to extract spatial and temporal features from raw images and videos. The CNN was built by an architecture, which was composed of convolution layer followed by subsampling or pooling layer, where *no* factorization was performed. In [22], the tensor-variate RBM was proposed as an unsupervised machine, which was trained by using tensor inputs. TF was used to represent the weight tensor. Hidden layer was still formed by a vector. No classification network was learned. In [23], a low rank approximation based on the singular value decomposition (SVD) was applied to speed up the evaluation of large-scale CNN. Decomposition and training of the CNN were separately performed by using different objectives.

B. Motivation and Main Idea

The NN model is estimated from the vector inputs. For classification of speech or music signals, we need to concatenate a set of spectral vectors from a contextual window and adopt this temporal-frequency input vector in a mini-batch training procedure. However, the association information inside this context window is ignored through unfolding into one-way vector. The structural observations are disregarded during NN training. The number of the required parameters and training samples should be increased to compensate the model redundancy. Although 2-D CNN [19], [20] or 3-D CNN [21] could be extended to multidimensional CNN, where the inputs and features in different layers were seen as multiway tensors, the convolution was not factorized and the components in different ways were not characterized. Here, we are motivated to build the factorized classification networks, which are trained from a set of *multiway* observations. Factorization and training of an NN are *jointly* performed under the unified discriminative objective [24].

This paper tightly incorporates TF into the construction of an NN model and proposes a *compact model* based on tensor-factorized NN (TFNN), which is learned efficiently from tensor inputs. Our idea is to preserve the tensor structure and build a *tensor-factorized MLP*, which is trained through the tensor-factorized error backpropagation procedure consisting of a forward pass and a backward pass. In forward pass, we transform the input tensors into hidden tensors via TF and nonlinear activation and then forward these tensor features layer by layer toward the softmax output layer as the class posteriors. TF performs the multilinear transformation. In backward pass, the factor matrices in different *ways* and *layers* are estimated through an error backpropagation, which minimizes the cross-entropy error function for classification. We present a general formulation to implement an N-way tensor-factorized MLP, which can be reduced to one-way MLP [25], where inputs and hidden units in different layers are all one-way vectors. The proposed N-way TFNN is different from 2-D or 3-D CNN in [19]–[21], which did not involve factorization toward different ways and the TNN in [16] and [18], where the tensor inputs were not allowed. Furthermore, the TFNN is extended to the convolutional TFNN (CTFNN), where the small portions or receptive fields of input tensors are modeled through the TFNN.

The rest of this paper is organized as follows. Section II surveys the fundamentals of TF and NN. Section III addresses the tensor feedforward computation and the tensor-factorized error backpropagation for the TFNN. The evaluation for parameter size and time complexity and the extension to CTFNN are illustrated. Section IV reports a series of experiments to evaluate the estimated parameters, computation time, and classification accuracy in the TFNN and the CTFNN. The conclusions drawn from this paper are given in Section V.

II. Tensor Factorization and Neural Network

A. Tensor Factorization

TF provides an effective solution to analyze a multiway structural observation. Here, a general N-way tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ with individual dimension I_n in each way n is considered. There are several TF methods [26], including Tucker decomposition, canonical decomposition/parallel factors (CPs) decomposition, and block term decomposition [1]. Through these factorization models, we can discover the underlying latent features from the observation and identify those common factors across different modalities. In the implementation, we may impose the constraints, such as sparsity, nonnegativity, orthogonality, or discrimination, to meet different situations and extract suitable features for various applications. For example, NTF is realized as a Tucker factorization subject to nonnegativity constraint, which results in a linear combination of outer products of the factor vectors.

In this paper, we use the boldface lowercase letter to denote a vector $\mathbf{a}$, the boldface capital letter to denote a matrix $\mathbf{A}$, and the boldface Euler script letter to denote a multiway tensor $\boldsymbol{\mathcal{A}}$. According to Tucker decomposition, the observed tensor $\mathcal{X}$ is decomposed into a core tensor $\mathcal{A} \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_N}$ accompanied with N factor matrices $\{\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}\}$ via

$$\mathcal{X} = \mathcal{A} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \times_3 \cdots \times_N \mathbf{U}^{(N)} \tag{1}$$

where the upper script in $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times J_n}$ denotes the nth way and $\times_n$ denotes the mode-n product. Each component $\mathcal{X}_{i_1 i_2 \ldots i_N}$ in tensor $\mathcal{X}$ is calculated by

$$\sum_{j_1=1}^{J_1} \sum_{j_2=1}^{J_2} \cdots \sum_{j_N=1}^{J_N} \mathcal{A}_{j_1 j_2 \ldots j_N} u^{(1)}_{i_1 j_1} u^{(2)}_{i_2 j_2} \cdots u^{(N)}_{i_N j_N}. \tag{2}$$

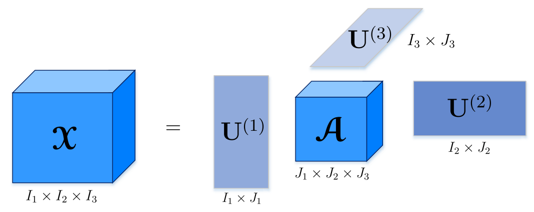


Fig. 1. Tucker decomposition for a three-way tensor.

The factorized core tensor $\mathcal{A}$ is treated as the tensor weights when integrating those factors in different ways. Such a Tucker decomposition is seen as the multilinear SVD [26]. The decomposition is not unique. It is more likely to find a unique solution by imposing constraints. Fig. 1 shows the Tucker decomposition for a three-way tensor. CP decomposition is viewed as a special realization of Tucker decomposition in which the core tensor $\mathcal{A}$ is superdiagonal and the number of components in different ways is equal $J_1 = \cdots = J_N$. Tucker decomposition can be implemented by using the higher order SVD [1]. Tucker or CP model was treated as a solution to low-rank approximation, which was applied to compensate the missing values for tensor completion [27].

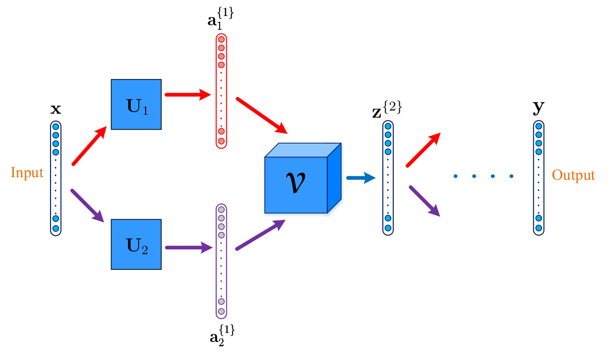
It is important that the *inverse* of Tucker decomposition in (1) can be obtained by

$$\mathcal{A} = \mathcal{X} \times_1 \mathbf{U}^{(1)^\dagger} \times_2 \mathbf{U}^{(2)^\dagger} \times_3 \cdots \times_N \mathbf{U}^{(N)^\dagger} \qquad (3)$$

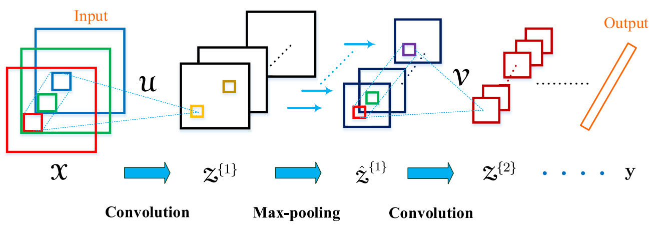
where $\mathbf{U}^{(n)^\dagger} = (\mathbf{U}^{(n)^T}\mathbf{U}^{(n)})^{-1}\mathbf{U}^{(n)^T}$ is the pseudoinverse of $\mathbf{U}^{(n)}$. If $J_1, J_2, \ldots, J_N$ are smaller than $I_1, I_2, \ldots, I_N$, the core tensor $\mathcal{A}$ is viewed as a *compressed* version of the original tensor $\mathcal{X}$. Alternatively, the compressed tensor $\mathcal{A}$ can be interpreted as the feature tensor with the reduced dimensions in different ways, which is extracted by transforming the observed tensor $\mathcal{X}$ using N-way factor matrices $\{\mathbf{U}^{(1)^\dagger}, \ldots, \mathbf{U}^{(N)^\dagger}\}$. This paper adopts this important property to perform multilinear tensor transformation for a factorized classification network based on the tensor-factorized MLP.

B. Neural Network

The NN is recognized as a powerful learning machine. There are two crucial calculations in the construction of an NN model. The first one is the affine transformation and nonlinear activation $h(\cdot)$ at each layer in the forward pass. The hidden vector $\mathbf{z}$ corresponding to an input vector $\mathbf{x}$ is calculated by $\mathbf{z} = h(\mathbf{U}\mathbf{x})$ using the affine matrix $\mathbf{U}$. The second one is the error backpropagation procedure in backward pass, which is implemented to calculate the NN parameters $\mathbf{U}$ in different layers. The performance of an NN model can be improved by involving a deep structure. In addition, we may extend the feedforward NN (FNN) to the RNN [28] where the hidden units of previous sample are augmented in input neurons to capture the temporal information. Traditionally, an FNN and an RNN are trained from one-way inputs. Affine transformation is viewed as the one-way factorization. No tensor inputs and tensor weights are allowed. This may constrain the capability of characterizing the structural information in real-world tasks.



(a)



(b)

Fig. 2. Illustration for (a) TNN and (b) CNN.

Fig. 2(a) and (b) shows two advanced NN models, which involve tensor inputs or tensor weights. A deep TNN [16] was proposed for acoustic modeling where the DP layer for a speech input vector $\mathbf{x}$ is formed by DP with two projected vectors $\mathbf{a}_1^{\{1\}}$ and $\mathbf{a}_2^{\{1\}}$. These vectors are obtained by using two affine matrices $\mathbf{U}_1$ and $\mathbf{U}_2$ and then activated by using the nonlinear activation $h(\cdot)$. In the next layer called the tensor layer, the hidden vector $\mathbf{z}^{\{2\}}$ is calculated through a bilinear transformation and a nonlinear activation

$$\mathbf{z}^{\{2\}} = h\big(\mathcal{V}^{\{2\}} \times_1 \mathbf{a}_1^{\{1\}} \times_2 \mathbf{a}_2^{\{1\}}\big). \qquad (4)$$

The parameters in the second layer are formed as a weight tensor $\mathcal{V}^{\{2\}}$. By optimizing the cross-entropy error function between classifier outputs $\mathbf{y}$ and class targets $\mathbf{r}$, TNN parameters in different layers are estimated by an error backpropagation algorithm. Here, the upper scripts $\{1\}$ and $\{2\}$ denote the first and second layers, respectively. This TNN uses the *tensor weights* $\mathcal{V}^{\{2\}}$ to fuse information from two nonlinear subspaces constructed by a DP layer.

The 2-D CNN [19], [20] and 3-D CNN [21] aim to extract the features from spatial horizon and both spatial and temporal horizons for classification problems, respectively. The spatial pattern in visual signals and the temporal dependence in time-series signals are sufficiently characterized. The competitive performance was achieved for image recognition and human action recognition. The CNN can be constructed as a deep stacking model where each stack consists of a convolution layer and a pooling layer. Instead of applying affine transformation in the NN, the 2-D CNN conducts the convolution operation, which captures small partitions of an image and finds the corresponding 2-D feature maps through 2-D convolution weights trained from an error backpropagation algorithm. The nonlinear activation is performed after the convolution. Besides, a subsampling method called the max-pooling is applied to deal with the small shift and rotation in an image and simultaneously alleviate the requirements of memory storage and computation overhead. Such a 2-D CNN has been developed for speech recognition where 2-D temporal-frequency inputs from speech signals were treated as a kind of image data for acoustic modeling [29]. Using the CNN for classification, the multiway inputs and their target classes $\{\boldsymbol{\mathcal{X}}_t, \mathbf{r}_t\}$ are collected for supervised training. In training procedure, the multiway information is conveyed through layers by using multiway convolution weights $\{\boldsymbol{\mathcal{U}}, \boldsymbol{\mathcal{V}}\}$ for the subregions in observation tensor $\boldsymbol{\mathcal{X}}$ and feature tensor $\boldsymbol{\mathcal{Z}}^{\{1\}}$ or subsampled tensor $\hat{\boldsymbol{\mathcal{Z}}}^{\{1\}}$. Fig. 2(b) shows the 2-D CNN by using three-way tensor inputs from color images with pixels of red, green, and blue.

Generally, the TNN uses three-way weight tensor to project two complimentary feature vectors into a unified feature vector. The inputs and features in different layers are all one-way vectors. Although the CNN allows multiway inputs and features in a layerwise model, the max-pooling is required to alleviate the curse of dimensionality. Basically, the TNN and the CNN do not involve multiway factorization for input tensors and feature tensors. Both methods do not extract the factorized components. The multiway components convey the common factors, which are beneficial for pattern classification.

III. Tensor-Factorized Neural Network

This paper proposes a new type of MLP, called the TFNN, which conducts the multiway factorization in a layered model where the inputs consist of N-way observations $\boldsymbol{\mathcal{X}}_t$ and their targets $\mathbf{r}_t$ for posterior classification outputs. The factorization is introduced to learn the common factors along different ways. A compact model can be built by using the factor matrices in different ways and layers. The structure of N-way data is conveyed from inputs to the layered features before calculating the posterior outputs. Convention of notations used in the construction of TFNN is provided in Table I.

A. Model Construction

The idea of integrating TF and NN into an N-way TFNN is originated by incorporating the property of inverse Tucker decomposition in (3) into MLP where the core tensor $\boldsymbol{\mathcal{A}}$ is obtained from an input tensor $\boldsymbol{\mathcal{X}}$ by performing an N-way linear transformation using the factor matrices $\{\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}\}$. Fig. 3 compares the classification networks based on one-way, two-way, and three-way TFNNs. Fig. 4 shows the factorization and activation of a three-way input tensor $\boldsymbol{\mathcal{X}}$ into the core tensor $\boldsymbol{\mathcal{A}}^{\{1\}}$ and then the feature tensor $\boldsymbol{\mathcal{Z}}^{\{1\}}$ in the first hidden layer. In case of $N = 1$, the weights in factor matrix $\mathbf{U}$ are used for affine transformation, which projects input vector $\mathbf{x}$ into a feature vector $\mathbf{z}^{\{1\}}$. Such a linear transformation corresponds to conduct the factor analysis or principal component analysis, which extracts the common factors or principal components from $\mathbf{x}$. This one-way TFNN is equivalent to the standard NN where the weight matrix $\mathbf{U}$ performs the one-way factorization. For $N = 2$, the feature matrix $\mathbf{Z}^{\{1\}}$ in first hidden layer is extracted from input matrix $\mathbf{X}$ by using bilinear or two-way transformation based on the factor matrices along two horizons $\mathbf{U}^{(1)}$ and $\mathbf{U}^{(2)}$. This two-way TFNN is different from 2-D CNN [19], [20] where no factorization was performed over 2-D convolution coefficients. For $N = 3$ in Figs. 3(c) and 4, the three-way TFNN is constructed

TABLE I

Convention of Notations

$\mathbf{r}_t = \{r_{tc}\},\ \mathbf{y}_t = \{y_{tc}\}$	class targets and outputs of sample t
$E(\boldsymbol{\Theta}),\ E_t(\boldsymbol{\Theta})$	error function with parameters $\boldsymbol{\Theta}$
$N_f,\ N_b,\ N_d$	no of multiplications for factorization, backpropagation and differentiation
N-way TFNN	
$\boldsymbol{\mathcal{X}} = \{\mathcal{X}_{i_1 \cdots i_N}\}$	input tensor with dim $I_1, \ldots, I_N$
$\boldsymbol{\Theta} = \{\mathbf{U}^{(n)}, \mathbf{V}^{(n)}, \boldsymbol{\mathcal{W}}\}$	set of parameters
$\mathbf{U}^{(n)} = \{U^{(n)}_{j_n i_n}\}$	n-th way factor matrix in layer $l-2$
$\{\mathbf{u}^{(n)}_{i_n}, \mathbf{u}^{(n)}_{j_n:}\}$	col. and row of factor matrix $\mathbf{U}^{(n)}$
$\mathbf{V}^{(n)} = \{V^{(n)}_{k_n j_n}\}$	n-th way factor matrix in layer $l-1$
$\boldsymbol{\mathcal{W}} = \{\mathcal{W}_{k_1 \cdots k_N c}\}$	par. connecting to output layer l
$K_1, \ldots, K_N$	tensor dim in layer $l-1$
$J_1, \ldots, J_N$	tensor dim in layer $l-2$
$\mathbf{a}^{\{l\}} = \{a^{\{l\}}_c\}$	activations in output layer l
$\boldsymbol{\mathcal{A}}^{\{l-1\}} = \{\mathcal{A}^{\{l-1\}}_{k_1 k_2 \cdots k_N}\}$	activation tensor in layer $l-1$
$\boldsymbol{\mathcal{A}}^{\{l-2\}} = \{\mathcal{A}^{\{l-2\}}_{j_1 j_2 \cdots j_N}\}$	activation tensor in layer $l-2$
$\boldsymbol{\mathcal{Z}}^{\{l-1\}} = \{\mathcal{Z}^{\{l-1\}}_{k_1 k_2 \cdots k_N}\}$	feature tensor in layer $l-1$
$\boldsymbol{\mathcal{Z}}^{\{l-2\}} = \{\mathcal{Z}^{\{l-2\}}_{j_1 j_2 \cdots j_N}\}$	feature tensor in layer $l-2$
$\tilde{\boldsymbol{\mathcal{Z}}}^{\{l-2\}} = \{\tilde{\mathcal{Z}}^{\{l-2\}}_{j_1 j_2 \cdots j_N}\}$	factorized feature tensor of $\boldsymbol{\mathcal{Z}}^{\{l-2\}}$
$\tilde{\boldsymbol{\mathcal{Z}}}^{\{l-3\}} = \{\tilde{\mathcal{Z}}^{\{l-3\}}_{i_1 i_2 \cdots i_N}\}$	factorized feature tensor of $\boldsymbol{\mathcal{Z}}^{\{l-3\}}$
$\mathbf{d}^{\{l\}} = \{d^{\{l\}}_c\}$	local gradients in output layer l
$\boldsymbol{\mathcal{D}}^{\{l-1\}} = \{\mathcal{D}^{\{l-1\}}_{k_1 k_2 \cdots k_N}\}$	local gradient tensor in layer $l-1$
$\boldsymbol{\mathcal{D}}^{\{l-2\}} = \{\mathcal{D}^{\{l-2\}}_{j_1 j_2 \cdots j_N}\}$	local gradient tensor in layer $l-2$
Three-way TFNN	
$\boldsymbol{\mathcal{X}} = \{\mathcal{X}_{ijk}\}$	input tensor with dim I, J, K
$\boldsymbol{\Theta} = \{\mathbf{U}, \mathbf{V}, \mathbf{W}, \boldsymbol{\mathcal{W}}\}$	set of parameters
$\mathbf{U} = \{U_{li}\} = \{\mathbf{u}_i, \mathbf{u}_{l:}\}$	col. and row in 1st way factor matrix
$\mathbf{V} = \{V_{mj}\} = \{\mathbf{v}_j, \mathbf{v}_{m:}\}$	col. and row in 2nd way factor matrix
$\mathbf{W} = \{W_{nk}\} = \{\mathbf{w}_k, \mathbf{w}_{n:}\}$	col. and row in 3rd way factor matrix
$\boldsymbol{\mathcal{W}} = \{\mathcal{W}_{lmnc}\}$	par. connecting to output layer l
$\boldsymbol{\mathcal{A}}^{\{l-1\}} = \{\mathcal{A}^{\{l-1\}}_{lmn}\}$	activation tensor in layer $l-1$
$\boldsymbol{\mathcal{A}}^{\{l-2\}} = \{\mathcal{A}^{\{l-2\}}_{ijk}\}$	activation tensor in layer $l-2$
$\boldsymbol{\mathcal{Z}}^{\{l-1\}} = \{\mathcal{Z}^{\{l-1\}}_{lmn}\}$	feature tensor in layer $l-1$
$\boldsymbol{\mathcal{Z}}^{\{l-2\}} = \{\mathcal{Z}^{\{l-2\}}_{ijk}\}$	feature tensor in layer $l-2$
$\boldsymbol{\mathcal{D}}^{\{l-1\}} = \{\mathcal{D}^{\{l-1\}}_{lmn}\}$	local gradient tensor in layer $l-1$
$\boldsymbol{\mathcal{D}}^{\{l-2\}} = \{\mathcal{D}^{\{l-2\}}_{ijk}\}$	local gradient tensor in layer $l-2$

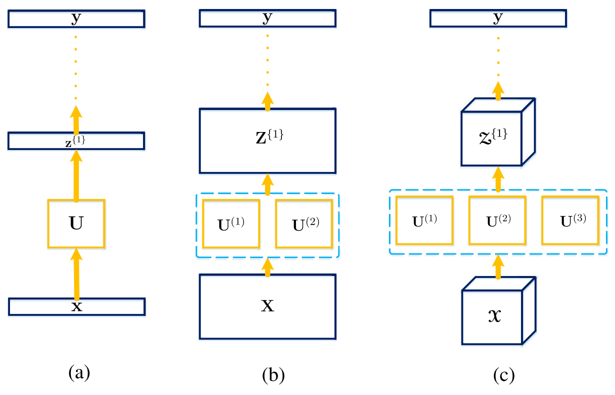


Fig. 3. Comparison of (a) one-way, (b) two-way, and (c) three-way TFNNs for classification.

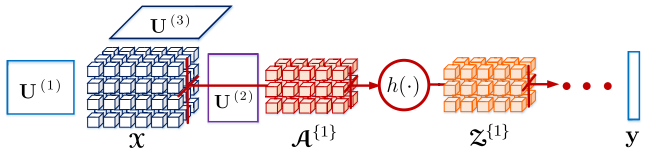


Fig. 4. Factorization and activation of an input tensor in three-way TFNN.

by factorizing and activating an input tensor $\mathcal{X}$ into the feature tensor $\mathcal{Z}^{\{1\}}$ by using three-way factor matrices $\{\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \mathbf{U}^{(3)}\}$ and activation function $h(\cdot)$. Such a factorized tensor $\mathcal{Z}^{\{1\}}$ is forwarded and further factorized to produce deep feature tensors $\{\mathcal{Z}^{\{2\}}, \mathcal{Z}^{\{3\}}, \ldots\}$ toward C posterior classification outputs $\mathbf{y} = \{y_c\}$, which are formed by using a softmax function $s(\cdot)$. A weight tensor $\mathcal{W}$ is introduced to project the hidden tensor into a vector of class posteriors. More generally, this model is feasible to factorize an N-way tensor input for the construction of an N-way TFNN by using different N values. The training procedure is fulfilled to estimate the parameters containing the factor matrices $\{\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}\}$ in different hidden layers and the weight tensor $\mathcal{W}$ in output layer by using the tensor-factorized error backprogagation.

B. Factorization and Feedforward Computation

First of all, we address the tensor feedforward computation for hidden layer and output layer [30].

1) Hidden Layer: Each hidden layer involves the computations for TF as well as nonlinear activation. The N-way input tensor $\mathcal{X} = \{\mathcal{X}_{i_1\ i_2 \cdots i_N}\}$ is first multilinearly transformed from $\mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ to $\mathbb{R}^{J_1 \times J_2 \times \cdots \times J_N}$ to obtain a core tensor

$$\begin{aligned}\mathcal{A}^{\{1\}} &= \mathcal{X} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \times_3 \cdots \times_N \mathbf{U}^{(N)} \\ &= \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_N=1}^{I_N} \mathcal{X}_{i_1 i_2 \cdots i_N} \left(\mathbf{u}_{i_1}^{(1)} \circ \mathbf{u}_{i_2}^{(2)} \circ \cdots \circ \mathbf{u}_{i_N}^{(N)}\right) \quad (5)\end{aligned}$$

where $\mathbf{U}^{(n)} = \{U_{j_n i_n}^{(n)}\} \in \mathbb{R}^{J_n \times I_n}$ denotes the nth factor matrix. In this multiplication, we sum up different tensors based on the outer products of N factors $\{\mathbf{u}_{i_1}^{(1)}, \mathbf{u}_{i_2}^{(2)}, \ldots, \mathbf{u}_{i_N}^{(N)}\}$ from individual columns of weight matrices $\{\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}\}$. This transformation connects the entries between input tensor and core tensor along different ways. These connections provide mutual interaction among different dimensions and modalities. A representative core tensor $\mathcal{A}^{\{1\}} = \{\mathcal{A}_{j_1 j_2 \cdots j_N}^{\{1\}}\}$ is used to approximate the input tensor $\mathcal{X}$ based on a set of factor matrices $\{\mathbf{U}^{(n)}\}$. The feature tensor $\mathcal{Z}^{\{1\}}$ in the first hidden layer is extracted through the nonlinear activation $\mathcal{Z}^{\{1\}} = h(\mathcal{A}^{\{1\}})$.

Alternatively, each entry of the core tensor in (5) $\mathcal{A}_{j_1 j_2 \cdots j_N}^{\{1\}}$ can be expressed by an inner product of the input tensor $\mathcal{X}$ and the weight tensor, which is formed by the outer product of the rows $\{\mathbf{u}_{j_1:}^{(1)}, \mathbf{u}_{j_2:}^{(2)}, \ldots, \mathbf{u}_{j_N:}^{(N)}\}$ of the corresponding factor matrices $\{\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}\}$

$$\begin{aligned}\mathcal{A}_{j_1 j_2 \cdots j_N}^{\{1\}} &= \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_N=1}^{I_N} \mathcal{X}_{i_1 i_2 \cdots i_N} U_{j_1 i_1}^{(1)} U_{j_2 i_2}^{(2)} \cdots U_{j_N i_N}^{(N)} \\ &= \langle \mathcal{X}, (\mathbf{u}_{j_1:}^{(1)} \circ \mathbf{u}_{j_2:}^{(2)} \circ \cdots \circ \mathbf{u}_{j_N:}^{(N)}) \rangle. \quad (6)\end{aligned}$$

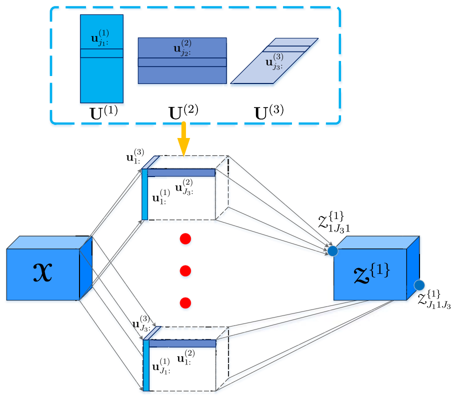

$\mathbf{u}_{j_1:}^{(1)}$
$\mathbf{u}_{j_2:}^{(2)}$
$\mathbf{u}_{j_3:}^{(3)}$
$\mathbf{U}^{(1)}$
$\mathbf{U}^{(2)}$
$\mathbf{U}^{(3)}$
$\mathcal{X}$
$\mathcal{Z}^{\{1\}}$
$\mathcal{Z}_{1 J_3 1}^{\{1\}}$
$\mathcal{Z}_{J_1 1 J_3}^{\{1\}}$

Fig. 5. Three-way factorization and activation of an input tensor $\mathcal{X}$ into feature tensor $\mathcal{Z}^{\{1\}}$. The outer product of the rows of three factor matrices (shown inside the dashed rectangle) $\{\mathbf{u}_{j_1:}^{(1)}, \mathbf{u}_{j_2:}^{(2)}, \mathbf{u}_{j_3:}^{(3)}\}$ is seen as a three-way weight tensor to calculate an entry $\mathcal{Z}_{j_1 j_2 j_3}^{\{1\}}$ in feature tensor.

The factorization of an input tensor $\mathcal{X}$ into a core tensor $\mathcal{A}^{\{1\}} \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_N}$ is realized as a kind of tensor transformation by using $J_1 \cdot J_2 \cdots J_N$ rank-one weight tensors, which mutually characterize different row factors in N factor matrices with N horizons. The hidden tensor in the first layer $\mathcal{Z}^{\{1\}}$ is then propagated layer by layer to build an architecture with the layered tensors $\mathcal{Z}^{\{2\}}, \mathcal{Z}^{\{3\}}, \ldots$. The weight tensors in the TFNN come from a shared set of row vectors from N factor matrices $\{\mathbf{U}^{(n)}\}$. With the *sharing* of weight factors or common bases along different ways, the TFNN can *efficiently* capture the data structure with much fewer parameters than the NN without factorization. As a result, the tensor-factorized model is more likely to use sparser parameters to represent the multiway inputs. The factor matrices $\{\mathbf{U}^{(n)}\}$ are utilized to extract a compact feature tensor $\mathcal{Z}^{\{1\}}$ from input tensor $\mathcal{X}$. Fig. 5 shows the feedforward calculation for $\mathcal{Z}^{\{1\}}$ in the first hidden layer when $N = 3$. This feature tensor $\mathcal{Z}^{\{1\}}$ is treated as an input for TF to calculate the core tensor $\mathcal{A}^{\{2\}}$ and the hidden tensor $\mathcal{Z}^{\{2\}} = h(\mathcal{A}^{\{2\}}) \in \mathbb{R}^{K_1 \times K_2 \times \cdots \times K_N}$ in the second hidden layer by using the factor matrices $\{\mathbf{V}^{(n)}\} = \{V_{k_n j_n}^{(n)}\}$ where $\mathbf{V}^{(n)} \in \mathbb{R}^{K_n \times J_n}$. Notably, the number of ways in different layers is assumed to be the same, but can also be different in different layers. We use the notations $\mathbf{U}^{(n)}$ and $\mathbf{V}^{(n)}$ for factor matrices in front and back layers, respectively. TF in the second hidden layer is done by

$$\mathcal{A}^{\{2\}} = \mathcal{Z}^{\{1\}} \times_1 \mathbf{V}^{(1)} \times_2 \mathbf{V}^{(2)} \times_3 \cdots \times_N \mathbf{V}^{(N)}. \quad (7)$$

2) Output Layer: For a classification problem, a softmax function $s(\cdot)$ is used to calculate the posterior outputs $\mathbf{y} = \{y_c\}$ over C classes corresponding to an input tensor $\mathcal{X}$ as

$$\mathbf{y} = s(\mathbf{a}^{\{l\}}) = \frac{\exp(\mathbf{a}^{\{l\}})}{\sum_{c=1}^{C} \exp\left(a_c^{\{l\}}\right)} \quad (8)$$

where $a_c^{\{l\}}$ is an entry of vector $\mathbf{a}^{\{l\}} \in \mathbb{R}^C$ in output layer l, which is obtained by applying the tensor transformation using the hidden tensor $\mathcal{Z}^{\{l-1\}}$ in previous layer $l-1$ and the parameter tensor $\mathcal{W}$ via $a_c^{\{l\}} = \langle \mathcal{W}_{:,:,\cdots,:,c}, \mathcal{Z}^{\{l-1\}} \rangle$. Here, $\mathcal{W} \in \mathbb{R}^{K_1 \times K_2 \times \cdots \times K_N \times C}$ is an $(N+1)$-way tensor which is used to connect the last hidden layer to output layer. Notation $\mathcal{W}_{:,:,\cdots,:c} \in \mathbb{R}^{K_1\times K_2\times\cdots\times K_N}$ means a reduced N-way tensor from $\mathcal{W}$, which is only connected to the cth output neuron. TFNN parameters are formed by the factor matrices in different ways and layers and the weight tensor in output layer $\boldsymbol{\Theta} = \{\mathbf{U}^{(n)}, \mathbf{V}^{(n)}, \mathcal{W}\}$. Given a set of training tensors and their class targets $\{\mathcal{X}_t, \mathbf{r}_t\}$, the cross-entropy error function is calculated by

$$E(\boldsymbol{\Theta}) = \sum_{t=1}^{T} E_t(\boldsymbol{\Theta}) = -\sum_{t=1}^{T}\sum_{c=1}^{C} r_{tc}\ln y_{tc} \tag{9}$$

where $\mathbf{y}_t = \{y_{tc}\}$ is the output vector and $\mathbf{r}_t = \{r_{tc}\}$ is the target vector corresponding to the tth input tensor $\mathcal{X}_t$ based on the 1-of-C coding scheme

$$r_{tc} = \begin{cases} 1, & \text{if } \mathcal{X}_t \text{ corresponds to class } c \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

The TFNN tightly integrates TF and an NN. The optimum of the unified parameters for factorization and classification is achieved in accordance with the class posteriors $\mathbf{y}$, which is seen as a discriminative objective. The TFNN aims to extract the multiway deep features for pattern classification. The structural learning in the TFNN is different from the purpose of rapid computation in large-scale NN training based on the low-rank decomposition [23] where the decomposition was separately trained from an NN according to a generative objective. In addition, the TFNN can also be extended to build a *regression network* in the presence of different ways of tensor inputs and regression outputs, where the sum-of-squares error function is minimized. The topology and the formulation for TFNN classification are similar to those for the TFNN regression in case that the one-way target outputs or a set of scalar targets are collected in regression task.

C. Tensor-Factorized Error Backpropagation

Different from stand-alone TF, where the factor matrices are estimated in an unsupervised fashion by minimizing the reconstruction error for a single tensor $\mathcal{X}$, the proposed TFNN conducts the supervised learning, integrates the TF and NN, and estimates the factor matrices and weight tensor from a set of training tensors and their targets $\{\mathcal{X}_t, \mathbf{r}_t\}$. The tensor-factorized error backpropagation procedure is developed through the stochastic gradient descent (SGD) algorithm, which is implemented by calculating the gradients of $E_t(\boldsymbol{\Theta})$ with respect to individual parameters in $\boldsymbol{\Theta} = \{U^{(n)}_{j_n i_n}, V^{(n)}_{k_n j_n}, \mathcal{W}_{k_1\ k_2\cdots k_N c}\}$ from output layer to different hidden layers in a backward manner. We derive a general formulation as follows.

1) Output Layer: Output layer l acts as a softmax layer. We first calculate the *local gradient* for individual output neuron c as

$$\frac{\partial E_t}{\partial a_c^{\{l\}}} = \sum_{k=1}^{C} \frac{\partial E_t}{\partial y_{tk}}\frac{\partial y_{tk}}{\partial a_c^{\{l\}}} = y_{tc} - r_{tc} \triangleq d_c^{\{l\}} \tag{11}$$

and then find the derivative for updating the individual entry $\mathcal{W}_{k_1k_2\cdots k_N c}$ in parameter tensor $\mathcal{W}$ by

$$\frac{\partial E_t}{\partial \mathcal{W}_{k_1k_2\cdots k_N c}} = \frac{\partial E_t}{\partial a_c^{\{l\}}}\frac{\partial a_c^{\{l\}}}{\partial \mathcal{W}_{k_1k_2\cdots k_N c}} = d_c^{\{l\}}\mathcal{Z}^{\{l-1\}}_{k_1k_2\cdots k_N}. \tag{12}$$

More generally, updating the parameter tensor $\mathcal{W}_{:,:,\cdots,:c}$ for the cth output in softmax layer is performed by using $\nabla_{\mathcal{W}_{:,:,\cdots,:c}} E_t = d_c^{\{l\}}\mathcal{Z}^{\{l-1\}}$, which is a scalar product of the local gradient $d_c^{\{l\}}$ of class c in layer l and the hidden tensor in the previous layer $\mathcal{Z}^{\{l-1\}} \in \mathbb{R}^{K_1\times K_2\times\cdots\times K_N}$.

After updating the parameter tensor $\mathcal{W}$, we propagate the local gradient from $\mathbf{d}^{\{l\}} = \{d_c^{\{l\}}\}$ in layer l back to $\mathcal{D}^{\{l-1\}} = \{\mathcal{D}^{\{l-1\}}_{k_1k_2\cdots k_N}\}$ in layer $l-1$ through a weighted mode-$(N+1)$ product of an $(N+1)$-way parameter tensor $\mathcal{W}$ and a gradient vector $\mathbf{d}^{\{l\}}$ with a weighting tensor $h'(\mathcal{A}^{\{l-1\}}) \in \mathbb{R}^{K_1\times K_2\times\cdots\times K_N}$

$$\mathcal{D}^{\{l-1\}} = h'(\mathcal{A}^{\{l-1\}}) * (\mathcal{W} \times_{(N+1)} \mathbf{d}^{\{l\}}) \tag{13}$$

where $*$ denotes the elementwise product. Equation (13) is derived according to

$$\begin{aligned}\frac{\partial E_t}{\partial \mathcal{A}^{\{l-1\}}_{k_1k_2\cdots k_N}} &= \sum_{c=1}^{C}\frac{\partial E_t}{\partial a_c^{\{l\}}}\frac{\partial a_c^{\{l\}}}{\partial \mathcal{Z}^{\{l-1\}}_{k_1k_2\cdots k_N}}\frac{\partial \mathcal{Z}^{\{l-1\}}_{k_1k_2\cdots k_N}}{\partial \mathcal{A}^{\{l-1\}}_{k_1k_2\cdots k_N}} \\ &= h'(\mathcal{A}^{\{l-1\}}_{k_1k_2\cdots k_N})\sum_{c=1}^{C} d_c^{\{l\}}\mathcal{W}_{k_1k_2\cdots k_N c} \triangleq \mathcal{D}^{\{l-1\}}_{k_1k_2\cdots k_N}.\end{aligned} \tag{14}$$

This derivation illustrates that each local gradient $\mathcal{D}^{\{l-1\}}_{k_1k_2\cdots k_N}$ in layer $l-1$ is proportional to the derivative $h'(\mathcal{A}^{\{l-1\}}_{k_1k_2\cdots k_N})$ and the weighted sum of local gradients $\{d_c^{\{l\}}\}$ in output layer l where the synapses $\{\mathcal{W}_{k_1k_2\cdots k_N c}\}$ from C outputs are treated as the weights.

2) Hidden Layer: Having the local gradient tensor $\mathcal{D}^{\{l-1\}}$ in layer $l-1$, the updating of each entry $V^{(n)}_{k_n j_n}$ in individual factor matrix in layer $l-1$ is derived by finding the derivative

$$\begin{aligned}&\frac{\partial E_t}{\partial V^{(n)}_{k_n j_n}} \\ &= \sum_{k_1=1}^{K_1}\cdots\sum_{k_{n-1}=1}^{K_{n-1}}\sum_{k_{n+1}=1}^{K_{n+1}}\cdots\sum_{k_N=1}^{K_N}\frac{\partial E_t}{\partial \mathcal{A}^{\{l-1\}}_{k_1k_2\cdots k_N}}\frac{\partial \mathcal{A}^{\{l-1\}}_{k_1k_2\cdots k_N}}{\partial V^{(n)}_{k_n j_n}} \\ &= \sum_{k_1=1}^{K_1}\cdots\sum_{k_{n-1}=1}^{K_{n-1}}\sum_{k_{n+1}=1}^{K_{n+1}}\cdots\sum_{k_N=1}^{K_N}\mathcal{D}^{\{l-1\}}_{k_1k_2\cdots k_N} \\ &\quad\times\left(\sum_{j_1=1}^{J_1}\cdots\sum_{j_{n-1}=1}^{J_{n-1}}\sum_{j_{n+1}=1}^{J_{n+1}}\cdots\sum_{j_N=1}^{J_N}\mathcal{Z}^{\{l-2\}}_{j_1j_2\cdots j_N}V^{(1)}_{k_1j_1}\cdots V^{(N)}_{k_Nj_N}\right)\end{aligned} \tag{15}$$

which is obtained by referring (6) and (7). This derivative can be written as an inner product of a subtensor of $\mathcal{D}^{(l-1)}$ with fixed dimension k_n in layer $l-1$ and a subtensor of $\tilde{\mathcal{Z}}^{\{l-2\}}$ with fixed dimension j_n in previous layer $l-2$

$$\frac{\partial E_t}{\partial V^{(n)}_{k_n j_n}} = \langle \mathcal{D}^{\{l-1\}}_{:,\cdots,:k_n:,\cdots,:}, \tilde{\mathcal{Z}}^{\{l-2\}}_{:,\cdots,:j_n:,\cdots,:}\rangle \tag{16}$$

where

$$\begin{aligned}\tilde{\mathcal{Z}}^{\{l-2\}}_{:,\cdots,:j_n:,\cdots,:} &\triangleq \mathcal{Z}^{\{l-2\}}_{:,\cdots,:j_n:,\cdots,:}\times_1\mathbf{V}^{(1)}\times_2\cdots\times_{n-1}\mathbf{V}^{(n-1)} \\ &\quad\times_{n+1}\mathbf{V}^{(n+1)}\times_{n+2}\cdots\times_N\mathbf{V}^{(N)}\end{aligned} \tag{17}$$

is seen as an $(N-1)$-way TF of the output tensor $\mathcal{Z}^{\{l-2\}}$ in layer $l-2$ by using all the factor matrices in layer $l-1$ except the nth factor matrix $\mathbf{V}^{(n)}$.

After updating the entries of factor matrices $\{V_{k_n j_n}^{(n)}\}$ in hidden layer $l-1$, we have to propagate the local gradient tensor from $\mathcal{D}^{\{l-1\}} \in \mathbb{R}^{K_1 \times K_2 \times \cdots \times K_N}$ in layer $l-1$ back to $\mathcal{D}^{\{l-2\}} \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_N}$ in layer $l-2$ via

$$\mathcal{D}^{\{l-2\}} = h'(\mathcal{A}^{\{l-2\}}) * \Big(\mathcal{D}^{\{l-1\}} \times_1 \mathbf{V}^{(1)^T} \times_2 \mathbf{V}^{(2)^T} \times_3 \cdots \times_N \mathbf{V}^{(N)^T}\Big) \tag{18}$$

which is derived by

$$\begin{aligned}\frac{\partial E_t}{\partial \mathcal{A}_{j_1 j_2 \cdots j_N}^{\{l-2\}}} &= \sum_{k_1=1}^{K_1}\sum_{k_2=1}^{K_2}\cdots\sum_{k_N=1}^{K_N} \frac{\partial E_t}{\partial \mathcal{A}_{k_1 k_2 \cdots k_N}^{\{l-1\}}} \frac{\partial \mathcal{A}_{k_1 k_2 \cdots k_N}^{\{l-1\}}}{\partial \mathcal{Z}_{j_1 j_2 \cdots j_N}^{\{l-2\}}} \\ &\quad\times \frac{\partial \mathcal{Z}_{j_1 j_2 \cdots j_N}^{\{l-2\}}}{\partial \mathcal{A}_{j_1 j_2 \cdots j_N}^{\{l-2\}}} \\ &= h'\big(\mathcal{A}_{j_1 j_2 \cdots j_N}^{\{l-2\}}\big) \sum_{k_1=1}^{K_1}\sum_{k_2=1}^{K_2}\cdots\sum_{k_N=1}^{K_N} \mathcal{D}_{k_1 k_2 \cdots k_N}^{\{l-1\}} \\ &\quad\times V_{k_1 j_1}^{(1)} V_{k_2 j_2}^{(2)} \cdots V_{k_N j_N}^{(N)} \triangleq \mathcal{D}_{j_1 j_2 \cdots j_N}^{\{l-2\}}.\end{aligned} \tag{19}$$

The backpropagation from $\mathcal{D}^{\{l-1\}}$ to $\mathcal{D}^{\{l-2\}}$, shown in (18), is expressed in a style of the *weighted TF*, where the factor matrices are all transposed and the derivative tensor $h'(\mathcal{A}^{\{l-2\}}) \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_N}$ is treated as the weights. Such a procedure is performed to obtain local gradients in different layers $\{\mathbf{d}^{\{l\}}, \mathcal{D}^{\{l-1\}}, \mathcal{D}^{\{l-2\}}, \ldots, \mathcal{D}^{\{1\}}\}$ in a backward manner. Having the local gradient tensor $\mathcal{D}^{\{l-2\}} \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_N}$, we similarly calculate the derivative for updating the factor matrices in layer $l-2$ by

$$\frac{\partial E_t}{\partial U_{j_n i_n}^{(n)}} = \big\langle \mathcal{D}_{:\cdots:j_n:\cdots:}^{\{l-2\}}, \tilde{\mathcal{Z}}_{:\cdots:i_n:\cdots:}^{\{l-3\}} \big\rangle \tag{20}$$

where

$$\tilde{\mathcal{Z}}_{:\cdots:i_n:\cdots:}^{\{l-3\}} \triangleq \mathcal{Z}_{:\cdots:i_n:\cdots:}^{\{l-3\}} \times_1 \mathbf{U}^{(1)} \times_2 \cdots \times_{n-1} \mathbf{U}^{(n-1)} \times_{n+1} \mathbf{U}^{(n+1)} \times_{n+2} \cdots \times_N \mathbf{U}^{(N)} \tag{21}$$

and $\mathcal{Z}^{\{l-3\}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$. Accordingly, the differentiations of E_t with respect to individual TFNN parameters from $\{\mathcal{W}_{k_1\ k_2 \cdots k_N c}\}$ in layer l, $\{V_{j_n i_n}^{(n)}\}$ in layer $l-1$ to $\{U_{k_n j_n}^{(n)}\}$ in layer $l-2$ are derived for implementation.

Algorithm 1 illustrates the tensor-factorized error backpropagation. The formulas for updating the parameters between two layers are consistently expressed as a product of hidden tensor $\mathcal{Z}^{\{l-1\}}$, $\tilde{\mathcal{Z}}^{\{l-2\}}$ or $\tilde{\mathcal{Z}}^{\{l-3\}}$ and local gradient $\mathbf{d}^{\{l\}}$, $\mathcal{D}^{\{l-1\}}$ or $\mathcal{D}^{\{l-2\}}$. Different from the traditional error backpropagation, the derivatives $\{\partial E_t / \partial U_{j_n i_n}^{(n)}, \partial E_t / \partial V_{k_n j_n}^{(n)}\}$ are calculated by using the hidden tensor $\{\tilde{\mathcal{Z}}_{:\cdots:i_n:\cdots:}^{\{l-3\}}, \tilde{\mathcal{Z}}_{:\cdots:j_n:\cdots:}^{\{l-2\}}\}$ *after* $(N-1)$*-way factorization*. The tensor-factorized error backpropagation is proposed as a general solution to N-way TFNN. This algorithm can be realized to train model parameters for one-way TFNN, two-way TFNN, and three-way TFNN from training data based on vectors, matrices and tensors, respectively. The Appendix shows the realization of tensor-factorized error backpropagation for a three-way TFNN.

Algorithm 1 Tensor-Factorized Error Backpropagation

```
Input: Training tensors and class targets $\{\mathcal{X}_t, \mathbf{r}_t\}_{t=1}^T$
Output: Parameters $\Theta = \{U_{j_n i_n}^{(n)}, V_{k_n j_n}^{(n)}, \mathcal{W}_{k_1\ k_2 \cdots k_N c}\}$
for $t = 1, \ldots, T$ do
    Feedforward the network through factorization
        Eqs. (5)(7) and activation $h(\cdot)$ to find $\mathcal{A}$ and $\mathcal{Z}$
    Evaluate $d_c^{(l)}$ using Eq. (11) and backpropagate to
        $\{\mathcal{D}_{k_1 k_2 \cdots k_N}^{\{l-1\}}, \mathcal{D}_{j_1 j_2 \cdots j_N}^{\{l-2\}}\}$ using Eqs. (14)(19)
    Update $\{\mathcal{W}_{k_1\ k_2 \cdots k_N c}, V_{k_n j_n}^{(n)}, U_{j_n i_n}^{(n)}\}$ using
        the derivatives in Eqs. (12)(16)(20)
end
Repeat the whole process until $\Theta$ converged
```

D. Number of Parameters and Time Complexity

TFNN preserves the tensor structure from inputs to hidden features or even to outputs through the factorization over different ways and layers, while NN vectorizes the N-way input tensor $\mathcal{X}_t \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ into a one-way high-dimensional vector $\mathbf{x}_t \in \mathbb{R}^{I_1\ I_2 \cdots I_N}$. It is interesting to investigate the number of parameters of using the vectorized NN and TFNN. The tensor-based NN using TNN [16] is also evaluated. Considering the TFNN with two hidden layers constructed by N ways of factorized matrices $\mathbf{U}^{(n)} \in \mathbb{R}^{J_n \times I_n}$ and $\mathbf{V}^{(n)} \in \mathbb{R}^{K_n \times J_n}$ and one softmax tensor $\mathcal{W} \in \mathbb{R}^{K_1 \times K_2 \times \cdots \times K_N \times C}$, the number of parameters is determined by

$$\underbrace{\sum_{n=1}^N I_n J_n}_{\{\mathbf{U}^{(n)}\}} + \underbrace{\sum_{n=1}^N J_n K_n}_{\{\mathbf{V}^{(n)}\}} + \underbrace{C \prod_{n=1}^N K_n}_{\mathcal{W}}. \tag{22}$$

But, the number of NN parameters is largely increased to

$$\prod_{n=1}^N I_n J_n + \prod_{n=1}^N J_n K_n + C \prod_{n=1}^N K_n \tag{23}$$

where the first and second terms are replaced by using the products. Using TNN with the projected vector $\mathbf{a}_1^{\{1\}}$ with dimension $\mathbb{J}_1$ and $\mathbf{a}_2^{\{1\}}$ with dimension $\mathbb{J}_2$, the number of parameters with two hidden layers is calculated by $\prod_{n=1}^N I_n(\mathbb{J}_1 + \mathbb{J}_2) + \mathbb{J}_1 \mathbb{J}_2 \prod_{n=1}^N K_n + C \prod_{n=1}^N K_n$. Here, $\mathbb{J}_1$ and $\mathbb{J}_2$ should be much larger than J_1 and J_2 to catch the required multiway information from the vectorized input vector $\mathbf{x}$, respectively. The parameter size of a TNN is typically smaller than that of an NN but much larger than that of the TFNN. For a model with more parameters, we require more memory for storage [31] and more data for reliable estimation.

Next, we examine the time complexity. This information can be measured by the number of multiplications in different processing units in error backpropagation algorithm. For ease of expression, we assume that the number of dimensions is the same in different ways for hidden tensors $\mathcal{Z}^{\{l-2\}} \in \mathbb{R}^{J \times \cdots \times J}$ and $\mathcal{Z}^{\{l-1\}} \in \mathbb{R}^{K \times \cdots \times K}$. Only the calculation for the parameters in last hidden layer is considered. The multiplications come from three processing units including the following.

1) the feedforward TF (N_f);
2) the backpropagation of local gradients (N_b);
3) the calculation of derivatives (N_d).

Using the vectorized NN, we have $N_f = J^N K^N$, $N_b = J^N K^N$, and $N_d = J^N K^N$. The number of multiplications is 3 $J^N K^N$. Using the TFNN, the time complexity in factorization via (7) and backpropagation via (18) is the same as calculated by

$$N_f = N_b = \underbrace{J^N K}_{\text{mode-1 product}} + \underbrace{J^{N-1}K^2}_{\text{mode-2 product}} + \cdots + \underbrace{JK^N}_{\text{mode-}N\text{ product}}. \tag{24}$$

The complexity for calculation of derivatives is given by

$$\begin{aligned} N_d = &\underbrace{(NJ)}_{\text{no of TFs}} \underbrace{(J^{N-1}K + J^{N-2}K^2 + \cdots + JK^{N-1})}_{\text{Eq. (17): } (N-1)\text{-way TF}} \\ &+ \underbrace{(NJK)}_{\text{no of parameters}} \underbrace{K^{N-1}}_{\text{Eq. (16): inner product}} = N \cdot N_f. \end{aligned} \tag{25}$$

Therefore, the number of multiplications is greatly reduced from 3 $J^N K^N$ using an NN to $(N+2)(J^N K + J^{N-1}K^2 + \cdots + JK^N)$ using the TFNN. The more the ways N are observed in tensor data, the more significant the time complexity is reduced. The TFNN is compact and efficient in model training. Using the TNN, the number of multiplications for feedforward and backpropagation is determined by $N_f = N_b = \mathbb{J}^2\ K + \mathbb{J}K^2$ due to the bilinear transformation in (4) and that of derivatives is obtained by $N_d = 2\ N_f$. Here, we assume the same dimension $\mathbb{J}_1 = \mathbb{J}_2 = \mathbb{J}$ in two projected vectors $\mathbf{a}_1^{\{1\}}$ and $\mathbf{a}_2^{\{1\}}$. The time complexity of a TNN is given by $4(\mathbb{J}^2\ K + \mathbb{J}K^2)$, which is much smaller than that of an NN. The difference between the complexities of an NN and a TNN depends on how $\mathbb{J}$ is larger than J to catch the N-way information by using a DP layer.

E. Convolutional Tensor-Factorized Neural Network

The TFNN factorizes the multiway input tensors and hidden tensors in a layerwise structural model toward the desired target outputs for classification. The CNN [19]–[21] performs the feature extraction and classification, which capture the subregions of multiway tensors by convolving with multiple filters for finding the feature maps where the weights are shared in each filter. These filters extract multiple features but ignore the factorized structural information in different ways. Interestingly, the rank-one weight tensors of the TFNN as shown in Fig. 5 are also seen as a series of filters, which convolve with the whole input tensor $\mathcal{X}$ to obtain the feature tensor $\mathcal{Z}^{\{1\}}$, where the factorized structural information in different ways is preserved. These filters are constructed by the rows of factor matrices and used to conduct the factorized convolution over different bases or factors in different ways. Factorized features are extracted for classification.

Here, we are motivated by integrating the spirits of the CNN and the TFNN. The CTFNN is established by capturing local spatial and temporal information, which is crucial for image recognition and video recognition. The idea of CTFNN is to factorize the *subregion* of an N-way input tensor or hidden tensor as shown in Fig. 6. Similar to the CNN, the CTFNN identifies *multiple* feature maps where each map is obtained by an N-way factorized convolution. Factorized structural information from N ways is preserved. Using the CNN [20] in Fig. 2(b), the three-way input tensor $\mathcal{X}$ from the pixel values of red, green, and blue of color images is convolved with a set of three-way weight tensors $\mathcal{U}$, while *two*-way spatial information from 2-D images and the channels of red, green, and blue are captured but *not* factorized in structural learning. The proposed CTFNN does not only factorize two-way spatial features, but also color feature in the third way.

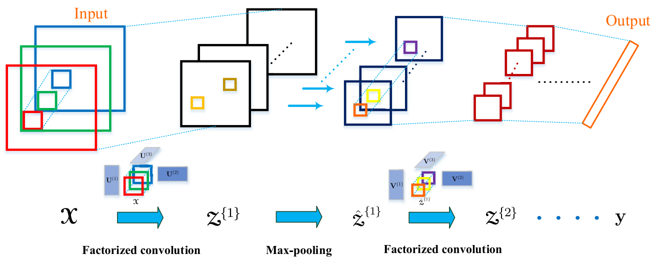


Fig. 6. Illustration for CTFNN.

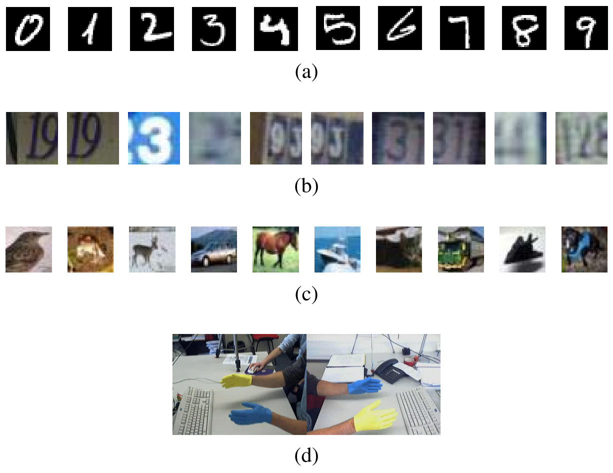


Fig. 7. Example samples from four data sets. (a) MINIST, (b) SVHN, (c) CIFAR-10, and (d) IDIAP.

The extended implementation from the TFNN to the CTFNN is straightforward. The CTFNN can be carried out by imposing the shared factor matrices on the subregions of input tensors and feature tensors. The same tensor-factorized error backpropagation can be applied to train the resulting factor matrices in different ways for different feature maps in different layers. Similar to the CNN, the max-pooling is performed over a spatial window for the CTFNN. The number of parameters and time complexity using CTFNN depends on the filter size for subregions, the dimension of the factorized features, the number of feature maps, the size of stride, and the size of max-pooling in different layers.

IV. Experiments

We propose a *general* framework of TFNN and CTFNN and conduct the experiments on image recognition and gesture recognition by using two-way, three-way, or four-way TFNN.

A. Experimental Data

We first introduce four data sets for evaluation of classification system. Fig. 7 shows some example samples.

1) MNIST: The Mixed National Institute of Standards and Technology (MNIST) database [19] consists of handwritten digits from 0 to 9 and is widely used for evaluation of classification model. This database contains 60 000 training images and 10 000 test images. A small set of training images is used for validation. Each digit is centered in a gray-scale image of size 28×28. A two-way TFNN is implemented by using two-way training images. This model is compared with an NN trained from one-way vectorized images of size 784×1.

2) SVHN: The street view house number (SVHN) data set [32], [33] is a real-world image data set, which was collected and cropped from the house number of Google street view. This data set is composed of color images of digits from 0 to 9. The labeled digit is in the center of image, but the remaining of the image may contain the other digits with natural backgrounds. The images are varied by shifting and rotation. There are totally 73 257 training images and 26 032 test images of size 32×32. A small set of training samples is held out for validation. We use the pixel values of red, green, and blue from three channels of an RGB image to form a three-way input tensor. A three-way TFNN is constructed from the training tensors of size $32 \times 32 \times 3$. We also implement an NN by using the vectorized training samples of size 3072×1.

3) CIFAR-10: The Canadian Institute for Advanced Research (CIFAR)-10 is a subset from an original database, which contains 80 million tiny images. This data set consists of 60 000 color images in ten classes with 6000 images per class with the label of airplane, automobile, bird, cat, deer, dog, frog, horse, ship, or truck. There are 50 000 training images and 10 000 test images in a size of 32×32. A set of validation data is held out for validation. We use the RGB images to construct three-way tensor inputs of size $32 \times 32 \times 3$. The images are highly varied. There is no canonical viewpoint or scale at which the objects appear.

4) IDIAP: The TwoHandManip Gesture database[1] from IDIAP consists of seven gesture classes: Front, Back, Push, Up, Down, Left, and Right collected from seven persons [34]. Each gesture had 70 video clips. We used 85% of the whole data set as the training and validation data and the remaining 15% as the test data. Each video was acquired with ten frames per second. Each frame had the resolution of 240×320. The region of interest in each video was cropped in a fixed duration of 1 s. Each image was resized to 8×12 by using the bilinear transformation. The RGB images in each frame were adopted. The four-way tensors containing spatial, temporal, and color information in a size of $8 \times 12 \times 10 \times 3$ were observed to construct four-way TFNN. Fivefold cross validation over different subsets of training data was performed.

B. Experimental Conditions

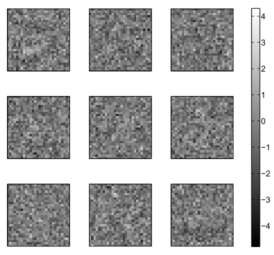
In the implementation, the pixel values of images in different data sets were scaled in a range between 0 and 1. Each image was subtracted by the per-pixel mean, which was computed over the data set. The weight parameters in different models were randomly initialized by the values from a normal distribution with zero mean and a variance, which was high enough to produce positive inputs to the neurons in each layer. Rectified linear unit was used as the activation function. To facilitate model training with good initialization and convergence, the layerwise pre-training was applied to build the initial stacked model. After stacking a layered architecture, we optimized the network model by fine tuning the weight parameters by using error backpropagation. The SGD algorithm with momentum of 0.9 and the mini-batch size of 100 examples in MNIST, SVHN, and CIFAR-10 and ten examples in IDIAP was used to minimize the cross-entropy error function. Learning rate was set equal for each layer. Initial learning rate was set to be 0.01. If the error function of validation data was not improved, we kept training the model in the next epoch. In case of no improvement again, the learning rate was halved to let network learning more stable and accurate. Training procedure was run for 100 learning epochs. To prevent overfitting problem in large-scale model training, a small weight decay was heuristically selected for individual tasks. Data augmentation using the elastic distortions [35] was applied. Random dropout [36] was performed. Local response normalization [20], [36] was run with the hyperparameters selected from validation data.

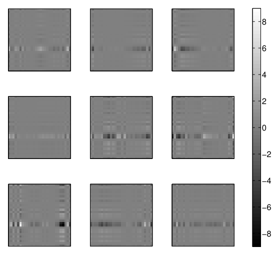
In the experiments on MNIST, SVHN, and CIFAR-10, two-layer and three-layer NNs and TFNNs were implemented and compared with CNNs [19], [20] and CTFNNs consisting of single layer and two layers of feature extraction. TNN [16] was carried out for comparison. Different topologies were evaluated in different models. In the implementation of CNN and CTFNN, the first layer produced eight feature maps using 5×5 filters with stride 1 followed by 2×2 max-pooling. The second layer produced eight feature maps using 5×5 filters with stride 1 followed by 2×2 max-pooling. A single fully connected softmax output layer was consistently implemented in different models. The CTFNN was implemented by using the factorized convolution based on 5×5 filter for two-way data or $5 \times 5 \times 3$ filter for three-way data. In addition to 5×5, this paper examines the filter sizes using 4×4, 6×6, and 7×7 for the CNN and the CTFNN. These conditions were generally employed in three tasks. MNIST used two-way input matrices while SVHN and CIFAR-10 used three-way input tensors. In the experiments on IDIAP, we implemented and compared the performance of gesture recognition by using a two-layer NN and TFNN in the presence of four-way tensor inputs. In our evaluation, the effects of the estimated procedure, the size of training data, the convolutional model, the number of parameters, the computation time and the training/test error rates were investigated by using different tasks. The averaged error rate with one standard deviation was calculated across test images.

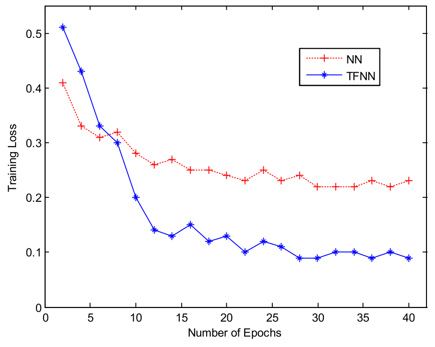
C. Evaluation for the Estimation Procedure

First of all, we evaluate the estimated weight parameters by using vectorized NN and TFNN where one-way transformation and two-way factorization are performed, respectively. MNIST task is investigated. A two-layer NN with 784 vectorized inputs, 300 hidden features, and ten class outputs (denoted by 784-300-10) and a two-layer TFNN with 28×28 input

[1]http://www.idiap.ch/resources/twohanded/


(a)


(b)



(c)

Fig. 8. Estimated parameters by using (a) NN and (b) TFNN. Parameter values are shown by gray levels in matrices. (c) Training loss in estimation procedure of an NN and an TFNN.

matrix, 40×40 hidden matrix, and ten outputs (denoted as 28×28-40×40-10) are included for a comparison. The topology is expressed in terms of the number of neurons in different layers. As shown in Fig. 8(a) and (b), the values of parameters are shown by gray levels in matrices. For the ease of comparison, the estimated weights of the NN are reshaped into matrices in the same size of the estimated weight matrices in the TFNN. We randomly show nine outer product matrices from the TFNN and nine reshaped matrices from an NN. Using the TFNN, these matrices are seen as rank-one matrices, which are formed by outer products of two rows of the estimated factor matrices in two ways similar to the interpretation for a three-way TFNN in Fig. 5. We can see that the weights in TFNN act more meaningfully as the dictionary for feature representation than those weights in an NN. In addition, the weight values in the TFNN are found to be much sparser than those in an NN. Such sparse weights are caused by the rank-one outer product in a weight matrix when using the TFNN. The TFNN has the capability of learning the relevance weights to represent geometrical or structural information in observed data. A sparse and compact model is obtained. An NN model contains redundant parameters, which may result in inefficient representation learning [37].

Fig. 8(c) displays the training loss in the estimation procedure of using NN (784-300-10) and TFNN (28×28-40×40-10). Training loss means the cross-entropy error function, which is averaged over all training samples. We find that the training procedure converges by using NN as well as TFNN. The convergence is achieved for TFNN since the TF and the layerwise propagation are jointly optimized under a unified cross-entropy error function. Here, TFNN obtains a much lower training loss than an NN.

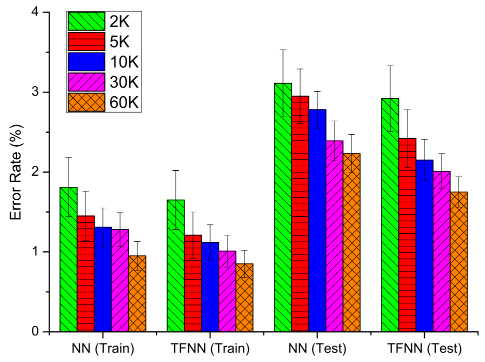


Fig. 9. Training and test error rates versus number of training samples (2000, 5000, 10000, 30000, and 60000) by using an NN and an TFNN. Standard deviation over test images and threefold validation is shown.

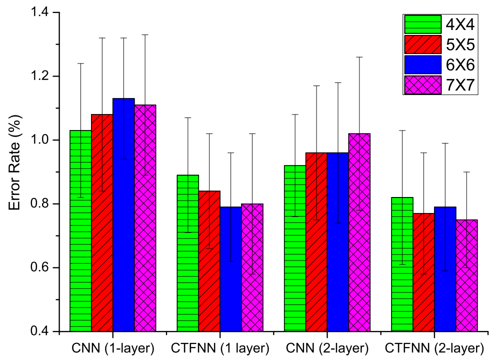


Fig. 10. Test error rate for CNN and CTFNN with different filter sizes (4×4, 5×5, 6×6, and 7×7) under single layer and two layers of feature extraction.

D. Effect on the Size of Training Data

Next, the training and test error rates are evaluated by using different sizes of training samples. A random selection of 2000, 5000, 10000, half or the full training set from MNIST data set is used. Error rates are averaged through threefold cross validation over subsets of training data. The NN with topology 784-300-10 and the TFNN with topology 28×28-40×40-10 are fixed. As shown in Fig. 9, the increasing number of training data does reduce the training errors as well as the test errors. Training errors are much lower than test errors. The more the training samples are used, the lower the error rates are obtained. The standard deviations of error rates are increased for smaller number of training data. This phenomenon is caused by the effect of small sample size. The TFNN performs better than an NN among these cases. For the case of using all training samples, TFNN achieves test error of 1.79%, which is lower than that of 2.23% by using an NN.

E. Evaluation for Convolutional Model

NN and TFNN are further extended to the convolutional models CNN and CTFNN by applying a set of shared filters over image subregions for convolution and a max-pooling for downsizing. The convolution and max-pooling are performed in one stage or two stages [38] similar to one or two hidden layers in an NN and a TFNN. To see the effect of convolution, we compare the test errors of CNN and CTFNN by using different sizes of filters including 4×4, 5×5, 6×6 and 7×7 as displayed in Fig. 10. The standard deviation of error rate

TABLE II
COMPARISON OF TEST ERROR RATE (%) AND NUMBER OF PARAMETERS BY USING DIFFERENT MODELS UNDER DIFFERENT TOPOLOGIES ON MNIST DATA SET

Model	Topology	Test Error	Par. Size
NN	784-70-10	3.03 ±0.25	55,580
NN	784-300-10	2.23 ±0.24	238,200
NN	784-300-300-10	2.11 ±0.21	328,200
TNN	784-{50;50}-10	1.88 ± 0.21	103,400
TNN	784-{80;80}-10	1.79 ±0.23	189,400
TFNN	28×28-20×20-10	2.05 ±0.23	5,120
TFNN	28×28-40×40-10	1.75 ±0.19	18,240
TFNN	28×28-40×40-20×20-10	1.67 ±0.24	7,840
CNN	5×5, 8 maps, stride 1, 1 layer	1.08 ±0.21	11,720
CNN	5×5, 8 maps, stride 1, 2 layers	0.96 ±0.21	3,080
CTFNN	5×5, 8 maps, stride 1, 1 layer	0.84 ±0.18	12,880
CTFNN	5×5, 8 maps, stride 1, 2 layers	0.77 ±0.19	3,520

TABLE III
COMPARISON OF COMPUTATION TIME (SECOND PER EPOCH) AND TEST ERROR RATE (%) BY USING DIFFERENT MODELS UNDER DIFFERENT TOPOLOGIES ON SVHN DATA SET

Model	Topology	Time	Test Error
NN	3072-500-10	200	11.32±0.63
NN	3072-1000-10	262	10.21±0.65
NN	3072-300-300-10	185	9.33± 0.54
TNN	3072-{200;200}-10	195	9.54±0.69
TNN	3072-{500;500}-10	250	9.11±0.61
TFNN	32×32×3-10×10×3-10	135	10.12±0.68
TFNN	32×32×3-20×20×3-10	148	8.35± 0.59
TFNN	32×32×3-20×20×3-10×10×2-10	109	8.19±0.61
CNN	5×5×3, 8 maps, stride 1, 1 layer	142	5.85±0.45
CNN	5×5×3, 8 maps, stride 1, 2 layers	111	5.15±0.35
CTFNN	5×5×3, 8 maps, stride 1, 1 layer	146	5.03±0.55
CTFNN	5×5×3, 8 maps, stride 1, 2 layers	119	4.91±0.49

is shown. MNIST data set is applied. For comparative study, the same pooling size (2×2), number of feature maps (8) and stride (1) are applied for one layer and two layers of feature extraction in CNN and CTFNN. Different from CNN, CTFNN conducts the factorized convolution by using two-way factor matrices estimated from training samples through the tensor-factorized error backpropagation. The feature map is formed by spatially connecting the convolved subimages. Here, two factor matrices are assumed to be 4×5. We find that the test errors are reduced by introducing more hidden layers in convolution models. CNN is generally improved by smaller filter size while CTFNN performs better for larger filter size which comes up with higher dimensional factor matrices. No matter what filter size is used, CTFNN has lower test error rate than CNN.

In addition, we compare the test error rates of convolutional models CNN and CTFNN with those of non-convolutional models NN, TNN and TFNN. Table II reports this comparison under different model topologies. The results of using single hidden layer and two hidden layers are compared. MNIST data set is used. Using TNN [16], the numbers in brackets in TNN topology 784-{50;50}-10 denote the size of an DP layer with 50 units in each of two parts. From Table II, the error rates are decreased with more hidden layers and more hidden units. The tensor methods TNN and TFNN perform better than non-tensor method NN. The error rates of using TFNN are comparable with those of using TNN. The error rates are substantially reduced by running convolutional models CNN and CTFNN when compared with non-convolutional models NN, TNN and TFNN. The lowest error rate 0.77% is achieved by using CTFNN with two hidden layers for feature extraction. These results illustrate the merit of conducting convolution and max-pooling in TFNN model.

F. Effects on Number of Parameters and Computation Time

Table II shows the number of parameters by using different models including NN, TNN, TFNN, CNN and CTFNN. Using NN and TFNN, this number is determined by (23) and (24), respectively. The number of parameters of CNN and CTFNN depends on the filter size, the pooling size, the number of feature maps and the stride. Although TNN obtains comparable error rates with TFNN, the number of parameters of TFNN is much smaller than that of TNN. Interestingly, TFNN with two hidden layer 28×28-40×40-20×20-10 obtains lower error rate with even more compact model when compared with the TFNN with one hidden layer 28×28-40×40-10. Number of parameters is highly affected by the number of units connecting to the class outputs. Similarly, the CNN and the CTFNN using two hidden layers are more compact than those using single hidden layer because two max-pooling layers have reduced a lot of parameters for connecting to class outputs. Here, the convolutional models attain desirable performance in error rates with moderate number of parameters. The number of parameters is comparable for CNN and CTFNN. CTFNN outperforms CNN in terms of test errors.

On the other hand, the computation time of running Python codes for different models is evaluated by using the personal computer equipped with an CPU of Intel$^{®}$Core^{TM} i7-4770@3.4GHz with 6 cores, a graphics processing unit (GPU) of NVIDIA GeForce GTX TITAN and a memory of 64G RAM. GPU was run for different neural models. CUDA library was applied to implement the parallelization for matrix multiplication in these models. Similar to [39], without running GPU, the computation cost of TFNN or CTFNN using CPU implementation was increased by 12 times. The computation time is measured by second per epoch in training procedure. SVHN data set is adopted. In general, this measure is affected by the number of training samples and the number of parameters and time complexity as addressed in Section III-D. Table III lists the comparison of computation times by using NN, TNN, TFNN, CNN, and CTFNN under different topologies. It is obvious that TFNN involves much smaller computation time than NN and TNN. In this evaluation, CTFNN is comparable with TFNN and CNN in computation time. This result is affected by the specification of convolutional models.

G. Evaluation for Different Tasks

To investigate the performance of different models, we further compare the test error rates of using NN, TNN, TFNN, CNN and CTFNN under different topologies. Tables III and IV displays the results of using SVHN and CIFAR-10 data

TABLE IV
COMPARISON OF COMPUTATION TIME AND TEST ERROR RATE (%) BY USING DIFFERENT MODELS UNDER DIFFERENT TOPOLOGIES ON CIFAR-10 DATA SET

Model	Topology	Time	Test Error
NN	3072-500-10	187	25.35±0.41
NN	3072-1000-10	248	21.58±0.33
NN	3072-300-300-10	166	19.97±0.35
TNN	3072-{200;200}-10	172	20.17±0.35
TNN	3072-{500;500}-10	228	19.67±0.28
TFNN	32×32×3-10×10×3-10	123	22.12±0.29
TFNN	32×32×3-20×20×3-10	129	19.11±0.34
TFNN	32×32×3-20×20×3-10×10×2-10	89	17.73±0.38
CNN	5×5×3, 8 maps, stride 1, 1 layer	127	13.63±0.25
CNN	5×5×3, 8 maps, stride 1, 2 layers	99	12.98±0.22
CTFNN	5×5×3, 8 maps, stride 1, 1 layer	128	12.75±0.25
CTFNN	5×5×3, 8 maps, stride 1, 2 layers	101	11.58±0.33

TABLE V
COMPARISON OF COMPUTATION TIME, TEST ERROR RATE (%), AND NUMBER OF PARAMETERS BY USING DIFFERENT METHODS UNDER DIFFERENT TOPOLOGIES ON IDIAP DATA SET

Model	Topology	Time	Test Error	Par. Size
CRF	–	6	14.92±0.19	1535
NN	960-10-7	21	18.35±0.23	9670
NN	960-20-7	28	17.83±0.21	19340
TNN	960-{8;8}-7	26	16.93±0.24	15808
TFNN	8×12×10×3-4×6×2×3-7	17	14.22±0.21	1141
TFNN	8×12×10×3-4×6×5×1-7	15	13.73±0.25	997
TFNN	8×12×10×3-4×6×5×3-7	19	13.21±0.18	2683

sets, respectively. The standard deviation across test samples is shown. The filter size $5 \times 5 \times 3$ is applied for convolution. Again, TFNN obtains better performance with fewer parameters than NN due to the use of multiway data structure. For example, using CIFAR-10, the error rate of TFNN (17.73%) with topology $32 \times 32 \times 3$-$20 \times 20 \times 3$-$10 \times 10 \times 2$-10 is lower than those of NN (19.97%) with topology 3072-300-300-10 and TNN (19.67%) with topology 3072-{600;600}-10. In addition, CNN and CTFNN performs better than NN, TNN and TFNN. The model performance of single-stage CNN and CTFNN is improved by running two stages of convolution and max-pooling. Error rates are reduced by introducing more hidden units and more layers. These findings are consistent for both tasks using SVHN and CIFAR-10 data sets. In these experiments, the lowest error rates 4.91% and 11.58% are obtained by applying CTFNN in SVHN and CIFAR-10 data sets, respectively. The factorization over different ways in TFNN and CTFNN is beneficial for system performance.

The evaluation of three-way TFNN is further extended to that of four-way TFNN by using IDIAP task. Number of training observations in this task is relatively smaller than that in the other three tasks. The schemes of data augmentation [35] and random dropout [36] considerably compensate the small sample size problem. For comparison, we carry out the result of conditional random field (CRF) which conducts the large-span data modeling using the procedure given in [34]. Table V shows the computation time, the test error rate and the number of parameters by using CRF, NN, TNN, and TFNN in the presence of $8 \times 12 \times 10 \times 3$ four-way tensors. The vectorized NN is implemented by using the motion frames in gray levels to avoid the overlarge parameters. In this comparison, CRF performs better than NN because the channel of temporal information in NN is not individually expressed. However, TFNN uses the temporal channel in four-way tensors and achieves the lowest error rate compared to CRF, NN and TNN. Using TFNN, reducing the temporal dimension of four-way hidden tensors as $4 \times 6 \times 2 \times 3$ or reducing the color dimension as $4 \times 6 \times 5 \times 1$ does improve the classification performance. Number of parameters of TFNN is even smaller than that of CRF. The lowest error rate is obtained by using TFNN with hidden tensors in a size of $4 \times 6 \times 5 \times\ 3$.

V. CONCLUSION

This paper has presented a novel TFNN as a multiway MLP, which discovers the multiway feature tensors from multiway structural observations. Through the factorization and feedforward computation, TFNN learns the factor matrices, which capture the mutual interaction between different horizons. This model can be efficiently trained by using tensor-factorized error backpropagation through SGD optimization. This model is compact with much fewer parameters than an NN model. The reconstructed tensors from the estimated factor matrices establish the meaningful dictionary with sparsity property. TFNN provides a new view to deal with multiway pattern classification. The factor matrices estimated by tensor-factorized error backpropagation are capable of extracting the factorized patterns in different ways. Similar to the extension of NN to CNN, TFNN is upgraded to its convolutional variant through multiple subregion filtering and max-pooling. Experimental results on classification tasks under different ways of input tensors show that TFNN obtains lower test error rates with fewer parameters and lower time complexity or computation time than NN. CNN and TFNN do improve system performance. CTFNN performs better than CNN for image recognition. Future works will be extended to explore the deep model and deal with the classification or regression for other types of multiway data.

APPENDIX: THREE-WAY TFNN

Considering a three-way TFNN with hidden layer containing three factor matrices $\mathbf{U} = \{U_{li}\} \in \mathbb{R}^{L \times I}$, $\mathbf{V} = \{V_{mj}\} \in \mathbb{R}^{M \times J}$ and $\mathbf{W} = \{W_{nk}\} \in \mathbb{R}^{N \times K}$ along three ways, the three-way input tensor $\mathcal{X} = \{\mathcal{X}_{ijk}\} \in \mathbb{R}^{I \times J \times K}$ can be factorized to obtain a core tensor $\mathcal{A}^{\{1\}} = \{\mathcal{A}^{\{1\}}_{lmn}\} \in \mathbb{R}^{L \times M \times N}$ by $\mathcal{A}^{\{1\}} = \mathcal{X} \times_1 \mathbf{U} \times_2 \mathbf{V} \times_3 \mathbf{W}$. Alternatively, each entry of core tensor is expressed by $\mathcal{A}^{\{1\}}_{lmn} = \sum_i \sum_j \sum_k \mathcal{X}_{ijk} U_{li} V_{mj} W_{nk}$. After this TF, the outputs of hidden layer are obtained via a nonlinear activation function $\mathcal{Z}^{\{1\}} = h(\mathcal{A}^{\{1\}}) \in \mathbb{R}^{L \times M \times N}$. These outputs are then treated as the inputs to calculate the activation tensor $\mathcal{A}^{\{2\}}$ and the feature tensor $\mathcal{Z}^{\{2\}}$ in the second hidden layer by using the corresponding factor matrices. Such factorization and activation are calculated toward the softmax layer to find $\mathbf{a}^{\{l\}} = \{a^{\{l\}}_c\} \in \mathbb{R}^C$ by using four-way parameter tensor $\mathcal{W} = \{\mathcal{W}_{\mathrm{lmnc}}\} \in \mathbb{R}^{L \times M \times N \times C}$ via a three-way tensor inner product $a^{\{l\}}_c = \langle \mathcal{W}_{:::c}, \mathcal{Z}^{\{l-1\}} \rangle$ and also the classification outputs $\mathbf{y} = \{y_c\} \in \mathbb{R}^C$ via (8). The feedforward computation is completed by repeating these calculations for T training tensors $\{\mathcal{X}_t\}$.

To estimate the model parameters $\mathbf{\Theta} = \{\mathbf{U}, \mathbf{V}, \mathbf{W}, \mathcal{W}\}$, we minimize the cross-entropy error function between training tensors and their class targets $\{\mathcal{X}_t, \mathbf{r}_t\}$ as shown in (9). The SGD algorithm is implemented through calculating the gradients of E_t with respect to individual parameters in $\mathbf{\Theta}$. Starting from the softmax layer l, we calculate the local gradient $d_c^{\{l\}} \triangleq ((\partial E_t)/(\partial a_c^{\{l\}}))$ of an output neuron c by (11) and then find the gradients $((\partial E_t)/(\partial \mathcal{W}_{\text{lmnc}}))$ for updating the three-way parameter tensor $\mathcal{W}_{:::c}$ which is a scalar product of the local gradient $d_c^{\{l\}}$ of class c in layer l and the hidden tensor from the previous layer $\mathcal{Z}^{\{l-1\}} \in \mathbb{R}^{L\times M\times N}$. Next, we propagate the local gradient from $\mathbf{d}^{\{l\}} = \{d_c^{\{l\}}\}$ in layer l back to $\mathcal{D}^{\{l-1\}} = \{\mathcal{D}_{lmn}^{\{l-1\}}\}$, where $\mathcal{D}_{lmn}^{\{l-1\}} \triangleq ((\partial E_t)/(\partial \mathcal{A}_{lmn}^{\{l-1\}}))$, in layer $l-1$ by $\mathcal{D}^{\{l-1\}} = h'(\mathcal{A}^{\{l-1\}}) * (\mathcal{W} \times_4 \mathbf{d}^{(l)})$ using an elementwise product and a mode-4 product. Having the local gradient tensor $\mathcal{D}^{\{l-1\}}$ in layer $l-1$, we can calculate the derivatives of E_t with respect to individual entries of three factor matrices U_{li}, V_{mj} and W_{nk} in layer $l-1$ by

$$\begin{aligned}\frac{\partial E_t}{\partial U_{li}} &= \sum_m \sum_n \frac{\partial E_t}{\partial \mathcal{A}_{lmn}^{\{l-1\}}} \frac{\partial \mathcal{A}_{lmn}^{\{l-1\}}}{\partial U_{li}} \\ &= \sum_m \sum_n \mathcal{D}_{lmn}^{\{l-1\}} \left(\sum_j \sum_k \mathcal{Z}_{ijk}^{\{l-2\}} V_{mj} W_{nk} \right) \\ &= \langle \mathcal{D}_{l::}^{\{l-1\}}, \mathcal{Z}_{i::}^{\{l-2\}} \times_2 \mathbf{V} \times_3 \mathbf{W} \rangle \qquad (26)\\ \frac{\partial E_t}{\partial V_{mj}} &= \langle \mathcal{D}_{:m:}^{\{l-1\}}, \mathcal{Z}_{:j:}^{\{l-2\}} \times_1 \mathbf{U} \times_3 \mathbf{W} \rangle \qquad (27)\\ \frac{\partial E_t}{\partial W_{nk}} &= \langle \mathcal{D}_{::n}^{\{l-1\}}, \mathcal{Z}_{::k}^{\{l-2\}} \times_1 \mathbf{U} \times_2 \mathbf{V} \rangle \qquad (28)\end{aligned}$$

which are shown as a two-way inner product of two matrices with each matrix from a three-way tensor with one way fixed. One matrix is from the subtensor of local gradient $\mathcal{D}^{\{l-1\}}$ in current layer $l-1$ and the other matrix is the two-way TF of the output tensor $\mathcal{Z}^{\{l-2\}}$ in previous layer $l-2$. The gradients in (27)–(29) are used to find factor matrices $\{\mathbf{U}, \mathbf{V}, \mathbf{W}\}$ in hidden layer. After that, the local gradient tensor $\mathcal{D}^{\{l-1\}}$ in layer $l-1$ is further propagated back to $\mathcal{D}^{\{l-2\}} = \{\mathcal{D}_{ijk}^{\{l-2\}}\}$ in layer $l-2$, where $D_{ijk}^{\{l-2\}} \triangleq ((\partial E_t)/(\partial \mathcal{A}_{ijk}^{\{l-2\}}))$, by

$$\mathcal{D}^{\{l-2\}} = h'(\mathcal{A}^{\{l-2\}}) * \left(\mathcal{D}^{\{l-1\}} \times_1 \mathbf{U}^T \times_2 \mathbf{V}^T \times_3 \mathbf{W}^T\right) \qquad (29)$$

which is obtained by

$$\frac{\partial E_t}{\partial \mathcal{A}_{ijk}^{\{l-2\}}} = h'(A_{ijk}^{\{l-2\}}) \sum_l \sum_m \sum_n \mathcal{D}_{lmn}^{\{l-1\}} U_{li} V_{mj} W_{nk}. \qquad (30)$$

This local gradient tensor in layer $l-2$ will be used to find the derivatives for updating the factor matrices in layer $l-2$.

References

[1] L. De Lathauwer, "Decompositions of a higher-order tensor in block terms—Part II: Definitions and uniqueness," *SIAM J. Matrix Anal. Appl.*, vol. 30, no. 3, pp. 1033–1066, Sep. 2008.

[2] H. Lu, K. N. Plataniotis, and A. N. Venetsanopoulos, "A survey of multilinear subspace learning for tensor data," *Pattern Recognit.*, vol. 44, no. 7, pp. 1540–1551, Jul. 2011.

[3] A. Cichocki *et al.*, "Tensor decompositions for signal processing applications: From two-way to multiway component analysis," *IEEE Signal Process. Mag.*, vol. 32, no. 2, pp. 145–163, Mar. 2015.

[4] T. Barker and T. Virtanen, "Non-negative tensor factorisation of modulation spectrograms for monaural sound source separation," in *Proc. Annu. Conf. Int. Speech Commun. Assoc. (INTERSPEECH)*, Lyon, France, Aug. 2013, pp. 827–831.

[5] J.-T. Chien and H.-L. Hsieh, "Nonstationary source separation using sequential and variational Bayesian learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 24, no. 5, pp. 681–694, May 2013.

[6] J.-T. Chien and P.-K. Yang, "Bayesian factorization and learning for monaural source separation," *IEEE/ACM Trans. Audio, Speech, Language Process.*, vol. 24, no. 1, pp. 185–195, Jan. 2016.

[7] A. Cichocki, R. Zdunek, A. H. Phan, and S.-I. Amari, *Nonnegative Matrix and Tensor Factorizations: Applications to Exploratory Multi-Way Data Analysis and Blind Source Separation*. Hoboken, NJ, USA: Wiley, Jan. 2009.

[8] C. Xu, D. Tao, and C. Xu, "Multi-view intact space learning," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 12, pp. 2531–2544, Dec. 2015.

[9] Y. Luo, D. Tao, K. Ramamohanarao, C. Xu, and Y. Wen, "Tensor canonical correlation analysis for multi-view dimension reduction," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 11, pp. 3111–3124, Nov. 2015.

[10] D. Tao, X. Li, X. Wu, and S. J. Maybank, "General tensor discriminant analysis and Gabor features for gait recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 10, pp. 1700–1715, Oct. 2007.

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

[12] D. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Providence, RI, USA, Jun. 2012, pp. 3642–3649.

[13] I. Sutskever, J. Martens, and G. Hinton, "Generating text with recurrent neural networks," in *Proc. Int. Conf. Mach. Learn. (ICML)*, Bellevue, WA, USA, Jun. 2011, pp. 1017–1024.

[14] J.-T. Chien and C.-H. Lee, "Deep unfolding for topic models," *IEEE Trans. Pattern Anal. Mach. Intell.*, to be published. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/7869412/, doi: 10.1109/TPAMI.2017.2677439.

[15] M. Ranzato, A. Krizhevsky, and G. E. Hinton, "Factored 3-way restricted Boltzmann machines for modeling natural images," in *Proc. Int. Conf. Artif. Intell. Statist. (AISTATS)*, Sardinia, Italy, May 2010, pp. 621–628.

[16] D. Yu, L. Deng, and F. Seide, "The deep tensor neural network with applications to large vocabulary speech recognition," *IEEE Trans. Audio, Speech, Language Process.*, vol. 21, no. 2, pp. 388–396, Feb. 2013.

[17] R. Socher, D. Chen, C. D. Manning, and A. Ng, "Reasoning with neural tensor networks for knowledge base completion," in *Advances in Neural Information Processing Systems*, vol. 26, C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, Eds. Lake Tahoe, NV, USA: Curran Associates, Inc., Dec. 2013, pp. 926–934.

[18] R. Memisevic, C. Zach, M. Pollefeys, and G. E. Hinton, "Gated softmax classification," in *Advances in Neural Information Processing Systems*, vol. 23, J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, Eds. Vancouver, BC, Canada: Curran Associates, Inc., Dec. 2010, pp. 1603–1611.

[19] 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.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, vol. 25, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds. Lake Tahoe, NV, USA: Curran Associates, Inc., Dec. 2012, pp. 1097–1105.

[21] S. Ji, W. Xu, M. Yang, and K. Yu, "3D convolutional neural networks for human action recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 1, pp. 221–231, Jan. 2013.

[22] T. D. Nguyen, T. Tran, D. Phung, and S. Venkatesh, "Tensor-variate restricted Boltzmann machines," in *Proc. AAAI Conf. Artif. Intell.*, Austin, TX, USA, Jan. 2015, pp. 2887–2893.

[23] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Advances in Neural Information Processing Systems*, vol. 27, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Montreal, QC, Canada: Curran Associates, Inc., Dec. 2014, pp. 1269–1277.

[24] J. T. Chien and C. P. Liao, "Maximum confidence hidden Markov modeling for face recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 4, pp. 606–616, Apr. 2008.

[25] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, Oct. 1986.

[26] L. De Lathauwer, B. De Moor, and J. Vandewalle, "A multilinear singular value decomposition," *SIAM J. Matrix Anal. Appl.*, vol. 21, no. 4, pp. 1253–1278, 2000.

[27] J. Liu, P. Musialski, P. Wonka, and J. Ye, "Tensor completion for estimating missing values in visual data," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 1, pp. 208–220, Jan. 2013.

[28] J.-T. Chien and Y.-C. Ku, "Bayesian recurrent neural network for language modeling," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 2, pp. 361–374, Feb. 2016.

[29] O. Abdel-Hamid, A.-R. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *IEEE/ACM Trans. Audio, Speech Language Process.*, vol. 22, no. 10, pp. 1533–1545, Oct. 2015.

[30] Y.-T. Bao and J.-T. Chien, "Tensor classification network," in *Proc. IEEE Int. Workshop Mach. Learn. Signal Process. (MLSP)*, Boston, MA, USA, Sep. 2015, pp. 1–6.

[31] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov, "Tensorizing neural networks," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, R. Garnett, and R. Garnett, Eds. Montreal, QC, Canada: Curran Associates, Inc., Dec. 2015, pp. 442–450.

[32] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *Proc. NIPS Workshop Deep Learn. Unsupervised Feature Learn.*, Granada, Spain, Dec. 2011, p. 5.

[33] P. Sermanet, S. Chintala, and Y. LeCun, "Convolutional neural networks applied to house numbers digit classification," in *Proc. Int. Conf. Pattern Recognit. (ICPR)*, Tsukuba, Japan, Nov. 2012, pp. 3288–3291.

[34] C.-P. Liao and J.-T. Chien, "Graphical modeling of conditional random fields for human motion recognition," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Las Vegas, NV, USA, Mar./Apr. 2008, pp. 1969–1972.

[35] P. Y. Simard, D. Steinkraus, and J. C. Platt, "Best practices for convolutional neural networks applied to visual document analysis," in *Proc. 7th IEEE Int. Conf. Document Anal. Recognit. (ICDAR)*, Aug. 2003, pp. 958–963.

[36] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. (2012). "Improving neural networks by preventing co-adaptation of feature detectors." [Online]. Available: https://arxiv.org/abs/1207.0580

[37] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. de Freitas, "Predicting parameters in deep learning," in *Advances in Neural Information Processing Systems*, vol. 26, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Lake Tahoe, NV, USA: Curran Associates, Inc., Dec. 2013, pp. 2148–2156.

[38] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?" in *Proc. Int. Conf. Comput. Vis. (ICCV)*, Kyoto, Japan, Sep./Oct. 2009, pp. 2146–2153.

[39] W. Liu, H. Zhang, D. Tao, Y. Wang, and K. Lu, "Large-scale paralleled sparse principal component analysis," *Multimedia Tools Appl.*, vol. 75, no. 3, pp. 1481–1493, Feb. 2016.

Jen-Tzung Chien (M'97–SM'04) received the Ph.D. degree in electrical engineering from National Tsing Hua University, Hsinchu, Taiwan, in 1997.

He is currently with the Department of Electrical and Computer Engineering, National Chiao Tung University, Hsinchu, as a University Chair Professor. He has authored extensively, including the book *Bayesian Speech and Language Processing* (Cambridge University Press, 2015). His current research interests include machine learning, deep learning, natural language processing, and image recognition.

Dr. Chien received the Best Paper Award of the IEEE Automatic Speech Recognition and Understanding Workshop in 2011. He served as an Associate Editor of the IEEE SIGNAL PROCESSING LETTERS from 2008 to 2011, a Guest Editor of the IEEE TRANSACTIONS ON AUDIO, SPEECH, AND LANGUAGE PROCESSING in 2012, and the Tutorial Speaker of the Interspeech in 2013 and 2016, and ICASSP in 2012, 2015, and 2017. He currently serves as an elected member of the IEEE Machine Learning for Signal Processing Technical Committee. He is the General Co-Chair of the IEEE International Workshop on Machine Learning for Signal Processing in 2017.

Yi-Ting Bao received the B.S. and M.S. degrees in electrical and computer engineering from National Chiao Tung University, Hsinchu, Taiwan, in 2013 and 2015, respectively.

His current research interests include machine learning, deep learning, tensor factorization, and image recognition.