Parallelized Tensor Train Learning of Polynomial Classifiers

Zhongming Chen*, Kim Batselier†, Johan A.K. Suykens‡ and Ngai Wong†

arXiv:1612.06505v3 [cs.LG] 23 Dec 2016

***Abstract*—In pattern classification, polynomial classifiers are well-studied methods as they are capable of generating complex decision surfaces. Unfortunately, the use of multivariate polynomials is limited to kernels as in support vector machines, because polynomials quickly become impractical for high-dimensional problems. In this paper, we effectively overcome the curse of dimensionality by employing the tensor train format to represent a polynomial classifier. Based on the structure of tensor trains, two learning algorithms are proposed which involve solving different optimization problems of low computational complexity. Furthermore, we show how both regularization to prevent overfitting and parallelization, which enables the use of large training sets, are incorporated into these methods. Both the efficiency and efficacy of our tensor-based polynomial classifier are then demonstrated on the two popular datasets USPS and MNIST.**

***Index Terms*—Supervised learning, tensor train, pattern classification, polynomial classifier.**

I. Introduction

Pattern classification is the machine learning task of identifying to which category a new observation belongs, on the basis of a training set of observations whose category membership is known. This machine learning task based on fully known label information is called supervised learning, which has been extensively studied and has wide applications in the fields of bioinformatics [1], computer-aided diagnosis (CAD) [2], machine vision [3], speech recognition [4], handwriting recognition [5], spam detection and many others [6]. Usually, different kinds of learning methods use different models to generalize from training examples to novel test examples.

As pointed out in [7], [8], one of the important invariants in these applications is the *local structure*: variables that are spatially or temporally nearby are highly correlated. Local correlations benefit extracting local features because configurations of neighboring variables can be classified into a small number of categories (e.g. edges, corners...). For instance, in handwritten character recognition, correlations between image pixels that are nearby tend to be more reliable than the ones of distant pixels. Learning methods incorporating this kind of prior knowledge often demonstrate state-of-the-art performance in practical applications. One popular method for handwritten character recognition is using convolutional neural networks (CNNs) [9], [10] which are variations of multilayer perceptrons designed to use minimal amounts of preprocessing. In this model, each unit in a layer receives inputs from a set of units located in a small neighborhood in the previous layer, and these mappings share the same weight vector and bias in a given convolutional layer. Another important component of a CNN are the pooling layers, which implement a nonlinear form of down-sampling. In this way, the amount of parameters and computational load are reduced in the network. Another popular method uses support vector machines (SVMs) [11], [12]. The original finite-dimensional feature space is mapped into a much higher-dimensional space, where the inner product is easily computed through the 'kernel trick'. By considering the Wolfe dual representation, one can find the maximum-margin hyperplane to separate the examples of different categories in that space. However, it is worth mentioning that these models require a large amount of memory and a long processing time to train the parameters. For instance, if there are thousands of nodes in the convolutional neural network, the weight matrices of fully-connected layers are of the order of millions. The major limitation of basic support vector machines is the high computational complexity which is at least quadratic with the dataset size. One way to deal with large datasets in the SVM-framework is by using fixed-size least squares support vector machines (fixed-size LS-SVM) [13]. The main idea used in fixed-size LS-SVM is to approximate the kernel mapping in such a way that the problem can be solved in the primal space.

To make storage and computation feasible, some researchers try to do the task of pattern classification by using tensors [14], [15], [16], [17], [18]. Different from these methods, we focus on extending the classic polynomial classifiers due to the close relationship between polynomials and tensors. That is, we exploit the efficient representation of a multivariate polynomial as a Tensor Train in order to avoid the curse of dimensionality, allowing us to work directly in the feature space. In this paper, we present the framework of tensor train learning. The main contributions are listed as follows.

- We derive a compact description of a polynomial classifier using the tensor train format, avoiding the curse of dimensionality.
- Two efficient learning algorithms are proposed by exploiting the tensor train structure.
- Both regularization and a parallel implementation are incorporated into our methods, thus avoiding overfitting and allowing the use of large training datasets.

This paper is organized as follows. In Section II, we give a brief introduction to tensor basics, including the tensor train decomposition, important tensor operations and properties.

*Department of Mathematics, School of Science, Hangzhou Dianzi University, Hangzhou 310018, China. Email: czm183015@126.com.

†Department of Electrical and Electronic Engineering, The University of Hong Kong. Email: {kimb, nwong}@eee.hku.hk.

‡KU Leuven, ESAT, STADIUS. B-3001 Leuven, Belgium. Email: johan.suykens@esat.kuleuven.be.

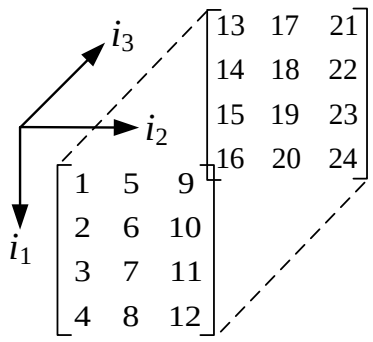


Fig. 1. An example tensor in $\mathbb{R}^{4\times3\times2}$.

The framework of tensor train learning for pattern classification is presented in Section III. Based on different loss functions, two efficient learning algorithms are proposed in Section IV, together with a discussion on regularization and parallelization. In Section V, we test our algorithms on two popular datasets: USPS and MNIST and compare their performance with polynomial classifiers trained with least squares support vector machines [13]. Finally, some conclusions and further work are summarized in Section VI.

Throughout this paper, we use small letters $x, y, \ldots,$ for scalars, small bold letters $\mathbf{x}, \mathbf{y}, \ldots,$ for vectors, capital letters $A, B, \ldots,$ for matrices, and calligraphic letters $\mathcal{A}, \mathcal{B}, \ldots,$ for tensors. The transpose of a matrix A or vector $\mathbf{x}$ is denoted by $A^\top$ and $\mathbf{x}^\top$, respectively. The identity matrix of dimension n is denoted by I_n.

II. Preliminaries

A. Tensors and pure-power-$\mathbf{n}$ polynomials

A real dth order tensor is a multidimensional array $\mathcal{A} \in \mathbb{R}^{n_1\times n_2\times\cdots\times n_d}$ that generalizes the notions of vectors and matrices to higher orders. Each of the entries $\mathcal{A}_{i_1 i_2\cdots i_d}$ is determined by d indices. The numbers $n_1, n_2, \ldots, n_d$ are called the dimensions of the tensor. An example tensor with dimensions 4, 3, 2 is shown in Fig. 1. We now give a brief introduction to some required tensor operations and properties, more information can be found in [19].

The k-mode product $\mathcal{B} = \mathcal{A} \times_k U$ of a tensor $\mathcal{A} \in \mathbb{R}^{n_1\times n_2\times\cdots\times n_d}$ and a matrix $U \in \mathbb{R}^{n'_k\times n_k}$ is defined by

$$\mathcal{B}_{i_1\cdots i_{k-1} j i_{k+1}\cdots i_d} = \sum_{i_k=1}^{n_k} \mathcal{A}_{i_1\cdots i_{k-1} i_k i_{k+1}\cdots i_d} U_{j i_k}, \quad (1)$$

and $\mathcal{B} \in \mathbb{R}^{n_1\times\cdots\times n_{k-1}\times n'_k\times n_{k+1}\times\cdots\times n_d}$. In particular, given a dth order tensor $\mathcal{A} \in \mathbb{R}^{n\times n\times\cdots\times n}$ and a vector $\mathbf{x} \in \mathbb{R}^n$, the multidimensional contraction, denoted by $\mathcal{A}\mathbf{x}^d$, is the scalar

$$\mathcal{A}\mathbf{x}^d = \mathcal{A} \times_1 \mathbf{x}^\top \times_2 \mathbf{x}^\top \times_3 \cdots \times_d \mathbf{x}^\top, \quad (2)$$

which is obtained as a homogeneous polynomial of $\mathbf{x} \in \mathbb{R}^n$ with degree d. The inner product of two same-sized tensors $\mathcal{A}, \mathcal{B} \in \mathbb{R}^{n_1\times n_2\times\cdots\times n_d}$ is the sum of the products of their entries, i.e.,

$$\langle \mathcal{A}, \mathcal{B} \rangle = \sum_{i_1=1}^{n_1} \sum_{i_2=1}^{n_2} \cdots \sum_{i_d=1}^{n_d} \mathcal{A}_{i_1 i_2\cdots i_d} \mathcal{B}_{i_1 i_2\cdots i_d}. \quad (3)$$

The Frobenius norm of a tensor $\mathcal{A} \in \mathbb{R}^{n_1\times n_2\times\cdots\times n_d}$ is given by

$$\|\mathcal{A}\|_F = \sqrt{\langle \mathcal{A}, \mathcal{A} \rangle}. \quad (4)$$

The vectorization of a tensor $\mathcal{A} \in \mathbb{R}^{n_1\times n_2\times\cdots\times n_d}$ is denoted by $\mathrm{vec}(\mathcal{A})$ and maps the tensor element with indices $(i_1, i_2, \ldots, i_d)$ to the vector element with index i where

$$i = i_1 + (i_2 - 1)n_1 + \cdots + (i_d - 1)\prod_{k=1}^{d-1} n_k.$$

Given d vectors $\mathbf{x}^{(i)} \in \mathbb{R}^{n_i}$, $i = 1, 2, \ldots, d$, their outer product is denoted by $\mathbf{x}^{(1)} \circ \mathbf{x}^{(2)} \circ \cdots \circ \mathbf{x}^{(d)}$, which is a tensor in $\mathbb{R}^{n_1\times n_2\times\cdots\times n_d}$ such that its entry with indices $(i_1, i_2, \ldots, i_d)$ is equal to the product of the corresponding vector elements, namely, $x^{(1)}_{i_1} x^{(2)}_{i_2} \cdots x^{(d)}_{i_d}$. It follows immediately that

$$\mathrm{vec}(\mathbf{x}^{(1)} \circ \mathbf{x}^{(2)} \circ \cdots \circ \mathbf{x}^{(d)}) = \mathbf{x}^{(d)} \otimes \mathbf{x}^{(d-1)} \otimes \cdots \otimes \mathbf{x}^{(1)}, \quad (5)$$

where the symbol "$\otimes$" denotes the Kronecker product.

We now illustrate how to represent a polynomial by using tensors. Denote by $\mathbb{R}[\mathbf{x}]$ the polynomial ring in d variables $\mathbf{x} = (x_1, x_2, \ldots, x_d)^\top$ with coefficients in the field $\mathbb{R}$.

Definition 1. *Given a vector $\mathbf{n} = (n_1, n_2, \ldots, n_d) \in \mathbb{N}^d$, a polynomial $f \in \mathbb{R}[\mathbf{x}]$ with d variables is called pure-power-$\mathbf{n}$ if the degree of f is at most n_i with respect to each variable x_i, $i = 1, 2, \ldots, d$.*

The set of all pure-power-$\mathbf{n}$ polynomials with the degree vector $\mathbf{n} = (n_1, n_2, \ldots, n_d) \in \mathbb{N}^d$ is denoted by $\mathbb{R}[\mathbf{x}]_\mathbf{n}$. For any $f(\mathbf{x}) \in \mathbb{R}[\mathbf{x}]_\mathbf{n}$, there are a total of $\prod_{k=1}^d (n_k + 1)$ distinct monomials

$$\prod_{k=1}^{d} x_k^{i_k - 1}, \quad 1 \le i_k \le n_k + 1, \quad k = 1, 2, \ldots, d.$$

For $\mathbf{x} = (x_1, x_2, \ldots, x_d)^\top \in \mathbb{R}^d$, denote by $\{\mathbf{v}(x_k)\}_{k=1}^d$ the Vandermonde vectors

$$\mathbf{v}(x_k) := (1, x_k, \ldots, x_k^{n_k})^\top \in \mathbb{R}^{n_k+1}. \quad (6)$$

It follows that there is a one-to-one mapping between pure-power-$\mathbf{n}$ polynomials and tensors. To be specific, for any $f(\mathbf{x}) \in \mathbb{R}[\mathbf{x}]_\mathbf{n}$, there exists a unique tensor $\mathcal{A} \in \mathbb{R}^{(n_1+1)\times(n_2+1)\times\cdots\times(n_d+1)}$ such that

$$f(\mathbf{x}) = \mathcal{A} \times_1 \mathbf{v}(x_1)^\top \times_2 \mathbf{v}(x_2)^\top \times_3 \cdots \times_d \mathbf{v}(x_d)^\top. \quad (7)$$

B. Tensor trains

It is well known that the number of tensor elements grows exponentially with the order d. Even if the mode size (i.e. the number of possible values of each index) of a tensor is small, the storage cost for all elements is prohibitive for large d. The tensor train decomposition [20] gives an efficient way (in storage and computation) to overcome this so-called *curse of dimensionality*.

The main idea of the tensor train (TT) decomposition is to re-express a tensor $\mathcal{A} \in \mathbb{R}^{n_1\times n_2\times\cdots\times n_d}$ as

$$\mathcal{A}_{i_1 i_2\cdots i_d} = \mathcal{G}_1(i_1)\mathcal{G}_2(i_2)\cdots\mathcal{G}_d(i_d), \quad (8)$$

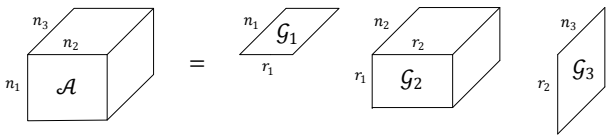


Fig. 2. The TT-decomposition for a tensor in $\mathbb{R}^{n_1 \times n_2 \times n_3}$.

where $\mathcal{G}_k(i_k)$ is a $r_{k-1} \times r_k$ matrix for each index i_k, also called the TT-core. To turn the matrix-by-matrix product (8) into a scalar, boundary conditions $r_0 = r_d = 1$ have to be introduced. The quantities $\{r_k\}_{k=0}^d$ are called the TT-ranks. Note that each core $\mathcal{G}_k$ is a third-order tensor with dimensions r_{k-1}, n_k and r_k. The TT-decomposition for a tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ is illustrated in Fig. 2. Let $n = \max\{n_1, n_2, \ldots, n_d\}$. It turns out that if all TT-ranks are bounded by r, the storage of the tensor train is $O(dnr^2)$, which only grows linearly with the order d. It has been shown that any tensor can be represented in tensor train format.

Proposition 1 (Theorem 2.1 of [21])**.** *For any tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$, there exists a TT-decomposition with TT-ranks*

$$r_k \leq \min(\prod_{i=1}^{k} n_i, \prod_{i=k+1}^{d} n_i), \quad k = 1, 2, \ldots, d-1.$$

We also mention that the TT representation of a tensor is not unique. For instance, let Q be an orthogonal matrix in $\mathbb{R}^{r_1 \times r_1}$, namely, $QQ^\top = Q^\top Q = I_{r_1}$. Then the tensor $\mathcal{A}$ in (8) also has the TT-decomposition

$$\mathcal{A}_{i_1 i_2 \cdots i_d} = \mathcal{G}_1'(i_1)\mathcal{G}_2'(i_2) \cdots \mathcal{G}_d(i_d), \tag{9}$$

where

$$\mathcal{G}_1'(i_1) = \mathcal{G}_1(i_1)Q, \quad \mathcal{G}_2'(i_2) = Q^\top \mathcal{G}_2(i_2).$$

Numerical stability of our learning algorithms is guaranteed by keeping all the TT-cores left-orthogonal or right-orthogonal [22], which is achieved through a sequence of QR decompositions as explained in Section IV.

Definition 2. *The $r_{k-1} \times n_k \times r_k$ core $\mathcal{G}_k$ is called left-orthogonal if*

$$\sum_{i_k=1}^{n_k} \mathcal{G}_k(i_k)^\top \mathcal{G}_k(i_k) = I_{r_k},$$

and the $r_{k-1} \times n_k \times r_k$ core $\mathcal{G}_k$ is called right-orthogonal if

$$\sum_{i_k=1}^{n_k} \mathcal{G}_k(i_k)\mathcal{G}_k(i_k)^\top = I_{r_{k-1}}.$$

As stated before, the structure of a tensor train also benefits the computation of the general multidimensional contraction:

$$f = \mathcal{A} \times_1 (\mathbf{v}^{(1)})^\top \times_2 (\mathbf{v}^{(2)})^\top \times_3 \cdots \times_d (\mathbf{v}^{(d)})^\top, \tag{10}$$

where $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$ and $\mathbf{v}^{(i)} = (v_1^{(i)}, v_2^{(i)}, \ldots, v_{n_i}^{(i)})^\top \in \mathbb{R}^{n_i}$, $i = 1, 2, \ldots, d$. If a tensor $\mathcal{A}$ is given in the tensor train format (8), then we have

$$f = \prod_{k=1}^{d} \sum_{i_k=1}^{n_k} v_{i_k}^{(k)} \mathcal{G}_k(i_k). \tag{11}$$

It follows that the computation of multidimensional contraction reduces to the computation of d matrices and evaluating matrix-by-vector products. The total computational complexity is $O(dnr^2)$, which is also linear in d. The described procedure for fast tensor train contraction is summarized in Algorithm 1. For more basic operations implemented in the tensor train format, such as tensor addition and computing the Frobenius norm, the reader is referred to [20].

Algorithm 1 Fast Tensor Train Contraction [20]

Input: Vectors $\mathbf{v}^{(k)} \in \mathbb{R}^{n_k}$, $k = 1, 2, \ldots, d$ and a tensor $\mathcal{A}$ in the TT-format with cores $\mathcal{G}_k$

Output: The multidimensional contraction f in (10)

```
for k = 1 : d do
    V^(k) = Σ_{i_k=1}^{n_k} v_{i_k}^(k) G_k(i_k)      %Computed in parallel
end for
v := V^(1)
for k = 2 : d do
    v := v V^(k)
end for
return f = v
```

III. Tensor Train Learning

It is easy for us to recognize a face, understand spoken words, read handwritten characters and identify the gender of a person. Machines, however, make decisions based on the data measured by a lot of sensors. In this section, we present the framework of tensor train learning. Like most pattern recognition systems [23], our tensor train learning method consists in dividing the system into three main modules, shown in Fig. 3.

The first module is called feature extraction, which is of paramount importance in any pattern classification problem. The goal of this module is to build features via transformations of the input (measurements) samples. The basic reasoning behind transform-based features is that an appropriately chosen transformation can exploit and remove information redundancies, which usually exist in the set of samples obtained by the measuring devices. The set of features exhibit high *information packaging* properties compared with the original input samples. This means that most of the classification-related information is compressed into a relatively small number of features, leading to a reduction of the necessary feature space dimension. Feature extraction benefits to training the classifier in terms of memory and computation, and also alleviates the problem of over-fitting since we get rid of redundant information. To deal with the task of feature extraction, some linear or nonlinear transformation techniques are widely used in the literature. For example, the Karhunen-Loève transform, related to principal component analysis (PCA), is one popular

method for feature generation and dimensionality reduction. A nonlinear kernel version of the classical PCA is called kernel principal component analysis, which is an extension of PCA using the techniques of kernel methods. The discrete Fourier transform (DFT) can be another good choice due to the fact that for many practical applications, most of the energy lies in the low-frequency components. Compared with PCA, the basis vectors in the DFT are fixed and problem dependent, which leads to a low computational complexity.

The second module, the TT classifier, is the core of tensor train learning. The purpose of this module is to mark a new observation based on its features generated by the previous module. As we discuss later, the task of pattern classification can be divided into a sequence of binary classifications. For each particular binary classification, the TT classifier assigns to each new observation a score that indicates which class it belongs to. In order to construct a good classifier, we exploit the fact that we know the labels for each sample of a given dataset. The TT classifier is trained optimally with respect to an *optimality criterion*. In some ways, the TT classifier can be regarded as a kind of generalized linear classifier, it does a linear classification in a higher dimensional space generated by the items of a given pure-power polynomial. The local information is encoded by the products of features. In contrast to kernel-based SVM classifiers that work in the dual space, the TT classifier is able to work in the high dimensional space by exploiting the tensor train format. Similar with the backpropagation algorithm for multilayer perceptrons, the structure of tensor trains allows for updating the cores in an alternating way. In the next section, we will describe the training of two TT classifiers through the optimization of two different loss functions.

The last module in Fig. 3 is called the decision module and decides which category a new observation belongs to. For binary classification, decisions are made according to the sign of the score assigned by the TT classifier, namely, the decision depends on the value of corresponding discriminant function. In an m-class problem, there are several strategies to decompose it into a sequence of binary classification problems. A straightforward extension is the *one-against-all*, where m binary classification problems are involved. We seek to design discriminant functions $\{g_i(\mathbf{x})\}_{i=1}^m$ so that $g_i(\mathbf{x}) > g_j(\mathbf{x})$, $\forall j \neq i$ if $\mathbf{x}$ belongs to the ith class. Classification is then achieved according to the rule:

$$\text{assign } \mathbf{x} \text{ to the } i\text{th class if } i = \text{argmax}_k\, g_k(\mathbf{x}).$$

An alternative technique is the *one-against-one*, where we need to consider $m(m-1)/2$ pairs of classes. The decision is made on the basis of a majority vote. It means that each classifier casts one vote and the final class is the one with the most votes. When the number m is too large, one can also apply the technique of binary coding. It turns out that only $\lceil \log_2 m \rceil$ classifiers are used, where $\lceil \cdot \rceil$ is the ceiling operation. In this case, each class is represented by a unique binary code word of length $\lceil \log_2 m \rceil$. The decision is then made on the basis of minimal Hamming distance.

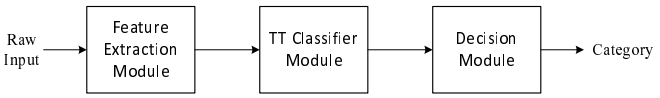


Fig. 3. Framework of tensor train learning.

IV. Learning Algorithms

As stated before, TT classifiers are designed for binary classification. Given a set of N training examples of the form $\{(\mathbf{x}^{(j)}, y^{(j)})\}_{j=1}^N$ such that $\mathbf{x}^{(j)} \in \mathbb{R}^d$ is the feature vector of the jth example and $y^{(j)} \in \{-1, 1\}$ is the corresponding label, depending on the class ownership of $\mathbf{x}^{(j)}$. Let $\mathbf{n} = (n_1, n_2, \ldots, n_d)^\top \in \mathbb{N}^d$ be the degree vector. Each feature is then mapped to a higher dimensional space generated by all corresponding pure-power-$\mathbf{n}$ monomials through the mapping $\mathcal{T} : \mathbb{R}^d \to \mathbb{R}^{(n_1+1)\times(n_2+1)\times\cdots\times(n_d+1)}$

$$\mathcal{T}(\mathbf{x})_{i_1 i_2 \cdots i_d} = \prod_{k=1}^{d} x_k^{i_k - 1}. \tag{12}$$

Here, we define $0^0 = 1$ for simplicity of notation. For $\mathbf{x} = (x_1, x_2, \ldots, x_d)^\top \in \mathbb{R}^d$, let $\{\mathbf{v}(x_k)\}_{k=1}^d$ be the Vandermonde vectors defined in (6). Clearly, we have

$$\mathcal{T}(\mathbf{x}) = \mathbf{v}(x_1) \circ \mathbf{v}(x_2) \circ \cdots \circ \mathbf{v}(x_d). \tag{13}$$

The introduction of this high-dimensional pure-power polynomial space benefits the learning task from the following aspects:

- all the interactions between features are well described by the monomials of pure-power polynomials;
- the dimension of the tensor space grows exponentially with d, namely, $\prod_{k=1}^d (n_k + 1)$, which increases the probability of separating all training examples linearly into two-class groupings;
- the one-to-one mapping between pure-power polynomials and tensors enables the use of tensor trains to lift the curse of dimensionality.

With these preparations, our goal is to find a decision hyperplane to separate these two-class examples in the tensor space, also called the *generic feature space*. In other words, like the inductive learning described in [14], we try to find a tensor $\mathcal{A} \in \mathbb{R}^{(n_1+1)\times(n_2+1)\times\cdots\times(n_d+1)}$ such that

$$y^{(j)} \langle \mathcal{T}(\mathbf{x}^{(j)}), \mathcal{A} \rangle > 0, \quad j = 1, 2, \ldots, N.$$

Note that the bias is absorbed in the first element of $\mathcal{A}$. It can also be interpreted to find a pure-power-$\mathbf{n}$ polynomial $g(\mathbf{x})$ such that

$$g(\mathbf{x}^{(j)}) > 0, \quad \forall y^{(j)} = 1,$$

and

$$g(\mathbf{x}^{(j)}) < 0, \quad \forall y^{(j)} = -1.$$

Here we consider that the tensor $\mathcal{A}$ is expressed as a tensor train with cores $\{\mathcal{G}_k\}_{k=1}^d$. The main idea of the TT learning algorithms is to update the cores in an alternating way by optimizing an appropriate loss function. Prior to updating the TT-cores, the TT-ranks are fixed and a particular initial guess of $\{\mathcal{G}_k\}_{k=1}^d$ is made. The TT-ranks can be interpreted as tuning

parameters, higher values will result in a better fit at the risk of overfitting. It is straightforward to extend our algorithms by means of the Density Matrix Renormalization Group (DMRG) method [24] such that the TT-ranks are updated adaptively. Each core is updated in the order

$$\mathcal{G}_1 \to \mathcal{G}_2 \to \cdots \to \mathcal{G}_d \to \mathcal{G}_{d-1} \to \cdots \to \mathcal{G}_1 \to \cdots$$

until convergence is reached. Convergence is guaranteed under certain conditions as described in [25], [26]. It turns out that updating one TT-core is equivalent with minimizing a loss function in a small number of variables, which can be done in a very efficient manner. The following theorem shows how the inner product $\langle \mathcal{T}(\mathbf{x}), \mathcal{A} \rangle$ in the generic feature space is a linear function in any of the TT-cores $\mathcal{G}_k$.

Theorem 1. *Given a vector $\mathbf{n} = (n_1, n_2, \ldots, n_d)^\top \in \mathbb{N}^d$, let $\mathcal{T}$ be the mapping defined by (12), and let $\mathcal{A}$ be a TT with cores $\mathcal{G}_k \in \mathbb{R}^{r_{k-1} \times (n_k+1) \times r_k}$, $k = 1, 2, \ldots, d$. For any $\mathbf{x} \in \mathbb{R}^d$ and $k = 1, \ldots, d$, we have that*

$$\langle \mathcal{T}(\mathbf{x}), \mathcal{A} \rangle = \left(\mathbf{q}_k(\mathbf{x})^\top \otimes \mathbf{v}(x_k)^\top \otimes \mathbf{p}_k(\mathbf{x}) \right) \mathrm{vec}(\mathcal{G}_k), \quad (14)$$

where

$$\mathbf{p}_1(\mathbf{x}) = 1, \quad \mathbf{p}_k(\mathbf{x}) = \prod_{i=1}^{k-1} \left(\mathcal{G}_i \times_2 \mathbf{v}(x_i)^\top \right) \in \mathbb{R}^{1 \times r_{k-1}},$$

and

$$\mathbf{q}_k(\mathbf{x}) = \prod_{i=k+1}^{d} \left(\mathcal{G}_i \times_2 \mathbf{v}(x_i)^\top \right) \in \mathbb{R}^{r_k \times 1}, \quad \mathbf{q}_d(\mathbf{x}) = 1.$$

Proof. By definition, we have

$$\begin{aligned} \langle \mathcal{T}(\mathbf{x}), \mathcal{A} \rangle &= \mathcal{A} \times_1 \mathbf{v}(x_1)^\top \times_2 \cdots \times_d \mathbf{v}(x_d)^\top \\ &= \left(\mathcal{G}_1 \times_2 \mathbf{v}(x_1)^\top \right) \cdots \left(\mathcal{G}_d \times_2 \mathbf{v}(x_d)^\top \right) \\ &= \mathcal{G}_k \times_1 \mathbf{p}_k(\mathbf{x}) \times_2 \mathbf{v}(x_k)^\top \times_3 \mathbf{q}_k(\mathbf{x})^\top \\ &= \left(\mathbf{q}_k(\mathbf{x})^\top \otimes \mathbf{v}(x_k)^\top \otimes \mathbf{p}_k(\mathbf{x}) \right) \mathrm{vec}(\mathcal{G}_k) \end{aligned}$$

for any $k = 1, 2, \ldots, d$. This completes the proof. □

Example 1. *In this example we illustrate the advantageous representation of a pure-power polynomial f as a TT. Suppose we have a polynomial f with $d = 10$ and all degrees $n_i = 9$ $(i = 1, \ldots, 10)$. All coefficients of $f(\mathbf{x})$ can then be stored into a 10-way tensor $10 \times 10 \times \cdots \times 10$ tensor $\mathcal{A}$ such that the evaluation of f in a particular $\mathbf{x}$ is given by* (7)*. The TT-representation of f consists of 10 TT-cores $\mathcal{G}_1, \ldots, \mathcal{G}_{10}$, with a storage complexity of $O(100r^2)$, where r is the maximal TT-rank. This demonstrates the potential of the TT-representation in avoiding the curse of dimensionality when the TT-ranks are small.*

Example 2. *Next, we illustrate the expressions for $\mathcal{T}(\mathbf{x}), \mathcal{A}, \mathbf{v}(x_k), \mathbf{q}_k(\mathbf{x}), \mathbf{p}_k(\mathbf{x})$ for the following quadratic polynomial in two variables $f(\mathbf{x}) = 1 + 3x_1 - x_2 - x_1^2 + 7x_1x_2 + 9x_2^2$. Since $d = 2$ and $n_1 = n_2 = 2$, both $\mathcal{T}$ and $\mathcal{A}$ are the following 3×3 matrices*

$$\mathcal{T}(\mathbf{x}) = \begin{pmatrix} 1 & x_2 & x_2^2 \\ x_1 & x_1x_2 & x_1x_2^2 \\ x_1^2 & x_1^2x_2 & x_1^2x_2^2 \end{pmatrix}, \mathcal{A} = \begin{pmatrix} 1 & -1 & 9 \\ 3 & 7 & 0 \\ -1 & 0 & 0 \end{pmatrix}.$$

The TT-representation of $\mathcal{A}$ consists of a $1 \times 3 \times 3$ tensor $\mathcal{G}_1$ and a $3 \times 3 \times 1$ tensor $\mathcal{G}_2$. Suppose now that $k = 2$ and we want to compute the evaluation of the polynomial f in a particular $\mathbf{x}$, which is $\langle \mathcal{T}(\mathbf{x}), \mathcal{A} \rangle$. From Theorem 1 we then have that

$$\langle \mathcal{T}(\mathbf{x}), \mathcal{A} \rangle = \left(\mathbf{q}_2(\mathbf{x})^\top \otimes \mathbf{v}(x_2)^\top \otimes \mathbf{p}_2(\mathbf{x}) \right) \mathrm{vec}(\mathcal{G}_2),$$

with

$$\begin{aligned} \mathbf{q}_2(\mathbf{x}) &= 1 \in \mathbb{R}, \\ \mathbf{v}(x_2) &= \begin{pmatrix} 1 & x_2 & x_2^2 \end{pmatrix}^\top \in \mathbb{R}^3, \\ \mathbf{p}_2(\mathbf{x}) &= \mathcal{G}_1 \times_2 \mathbf{v}(x_1)^\top \in \mathbb{R}^{1 \times 3}, \\ \mathbf{v}(x_1) &= \begin{pmatrix} 1 & x_1 & x_1^2 \end{pmatrix}^\top \in \mathbb{R}^3. \end{aligned}$$

In what follows, we first present two learning algorithms based on different loss functions. These algorithms will learn the tensor $\mathcal{A}$ directly in the TT-representation from a given dataset. Some techniques, like regularization and parallel implementation, will be described in the last two subsections.

A. TT Learning by Least Squares

Least squares estimation is the simplest and thus most common estimation method. In the generic feature space, we attempt to design a linear classier so that its desired output is exactly the label 1 or -1. However, we have to live with errors, that is, the true output will not always be equal to the desired one. The least squares estimator of the linear classifier is then found from minimizing the following mean square error function

$$J(\mathcal{A}) = \frac{1}{N} \sum_{j=1}^{N} \left(\langle T(\mathbf{x}^{(j)}), \mathcal{A} \rangle - y^{(j)} \right)^2. \quad (15)$$

We now show how updating a TT-core $\mathcal{G}_k$ is equivalent with solving a relatively small linear system. First, we define the $N \times r_{k-1}(n_k + 1)r_k$ matrix

$$C_k = \begin{bmatrix} \mathbf{q}_k(\mathbf{x}^{(1)})^\top \otimes \mathbf{v}(x_k^{(1)})^\top \otimes \mathbf{p}_k(\mathbf{x}^{(1)}) \\ \mathbf{q}_k(\mathbf{x}^{(2)})^\top \otimes \mathbf{v}(x_k^{(2)})^\top \otimes \mathbf{p}_k(\mathbf{x}^{(2)}) \\ \vdots \\ \mathbf{q}_k(\mathbf{x}^{(N)})^\top \otimes \mathbf{v}(x_k^{(N)})^\top \otimes \mathbf{p}_k(\mathbf{x}^{(N)}) \end{bmatrix} \quad (16)$$

for any $k = 1, 2, \ldots, d$. The matrix C_k is hence obtained from the concatenation of the row vectors $\mathbf{q}_k(\mathbf{x})^\top \otimes \mathbf{v}(x_k)^\top \otimes \mathbf{p}_k(\mathbf{x})$ from (14) for N samples $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}$. It follows from Theorem 1 that

$$J(\mathcal{A}) = \frac{1}{N} \| C_k \, \mathrm{vec}(\mathcal{G}_k) - \mathbf{y} \|^2 \quad (17)$$

where

$$\mathbf{y} = (y^{(1)}, y^{(2)}, \ldots, y^{(N)})^\top \in \mathbb{R}^N. \quad (18)$$

We have thus shown that updating the core $\mathcal{G}_k$ is equivalent with solving a least square optimization problem in $r_{k-1}(n_k + 1)r_k$ variables. Minimizing (17) with respect to $\mathcal{G}_k$ for any $k = 1, \ldots, d$ results in solving the linear system

$$(C_k^\top C_k) \, \mathrm{vec}(\mathcal{G}_k) = C_k^\top \mathbf{y}, \quad (19)$$

with a complexity of at most $O((r_{k-1}(n_k + 1)r_k)^3)$.

B. TT Learning by Logistic Regression

Since our goal is to find a hyperplane to separate two-class training examples in the generic feature space, we may not care about the particular value of the output. Indeed, only the sign of the output makes sense. This gives us the idea to decrease the number of sign differences as much as possible when updating the TT-cores, that is to minimize the number of misclassified examples. However, this model is discrete so that a difficult combinatorial optimization problem is involved. Instead, we try to find a suboptimal solution in the sense of minimizing a continuous cost function that penalizes misclassified examples. Here, we consider the logistic regression cost function. First let us consider the standard sigmoid function

$$\sigma(z) = \frac{1}{1+e^{-z}}, \quad z \in \mathbb{R},$$

where the output always takes values between 0 and 1. An important property is that its derivative can be expressed by the function itself, i.e.,

$$\sigma'(z) = \sigma(z)(1-\sigma(z)). \tag{20}$$

The logistic function for the jth example $\mathbf{x}^{(j)}$ is given by

$$h_{\mathcal{A}}(\mathbf{x}^{(j)}) := \sigma\left(\langle T(\mathbf{x}^{(j)}), \mathcal{A}\rangle\right). \tag{21}$$

We can also interpret the logistic function as the probability that the example $\mathbf{x}^{(j)}$ belongs to the class denoted by the label 1. The predicted label $\tilde{y}^{(j)}$ for $\mathbf{x}^{(j)}$ is then obtained according to the rule

$$\begin{cases} h_{\mathcal{A}}(\mathbf{x}^{(j)}) \geq 0.5 \Leftrightarrow \langle T(\mathbf{x}^{(j)}), \mathcal{A}\rangle \geq 0 \rightarrow \tilde{y}^{(j)} = 1, \\ h_{\mathcal{A}}(\mathbf{x}^{(j)}) < 0.5 \Leftrightarrow \langle T(\mathbf{x}^{(j)}), \mathcal{A}\rangle < 0 \rightarrow \tilde{y}^{(j)} = -1. \end{cases}$$

For a particular example $\mathbf{x}^{(j)}$, we define the cost function as

$$\text{Cost}(\mathbf{x}^{(j)}, \mathcal{A}) = \begin{cases} -\log\left(h_{\mathcal{A}}(\mathbf{x}^{(j)})\right) & \text{if } y^{(j)} = 1, \\ -\log\left(1 - h_{\mathcal{A}}(\mathbf{x}^{(j)})\right) & \text{if } y^{(j)} = -1. \end{cases}$$

The goal now is to find a tensor $\mathcal{A}$ such that $h_{\mathcal{A}}(\mathbf{x}^{(j)})$ is near 1 if $y^{(j)} = 1$ or near 0 if $y^{(j)} = -1$. As a result, the logistic regression cost function for the whole training dataset is given by

$$\begin{aligned} J(\mathcal{A}) &= \frac{1}{N}\sum_{j=1}^{N} \text{Cost}(\mathbf{x}^{(j)}, \mathcal{A}) \\ &= \frac{-1}{N}\sum_{j=1}^{N}\left[\frac{1+y^{(j)}}{2}\log\left(h_{\mathcal{A}}(\mathbf{x}^{(j)})\right) + \right. \\ &\qquad \left. \frac{1-y^{(j)}}{2}\log\left(1-h_{\mathcal{A}}(\mathbf{x}^{(j)})\right)\right]. \end{aligned} \tag{22}$$

It is important to note that the logistic regression cost function (22) is convex though the sigmoid function is not. This guarantees that we can find the globally optimal solution instead of getting stuck in a local optimum.

From equation (21) and Theorem 1 one can see that the function $J(\mathcal{A})$ can also be regarded as a function of the core $\mathcal{G}_k$ since

$$\langle T(\mathbf{x}^{(j)}), \mathcal{A}\rangle = C_k(j,\ :)\,\text{vec}(\mathcal{G}_k)$$

where $C_k(j,\ :)$ denote the jth row vector of C_k defined in (16). It follows that updating the core $\mathcal{G}_k$ is equivalent with solving a convex optimization problem in $r_{k-1}(n_k+1)r_k$ variables. Let

$$\mathbf{h}_{\mathcal{A}} = \left(h_{\mathcal{A}}(\mathbf{x}^{(1)}), h_{\mathcal{A}}(\mathbf{x}^{(2)}), \ldots, h_{\mathcal{A}}(\mathbf{x}^{(N)})\right)^{\top} \in \mathbb{R}^N \tag{23}$$

and $D_{\mathcal{A}}$ be the diagonal matrix in $\mathbb{R}^{N\times N}$ with the jth diagonal element given by $h_{\mathcal{A}}(\mathbf{x}^{(j)})\left(1-h_{\mathcal{A}}(\mathbf{x}^{(j)})\right)$. By using the property (20) one can derive the gradient and Hessian with respect to $\mathcal{G}_k$ as

$$\nabla_{\mathcal{G}_k} J(\mathcal{A}) = \frac{1}{N} C_k^{\top}\left(\mathbf{h}_{\mathcal{A}} - \frac{\mathbf{y}+\mathbf{1}}{2}\right) \tag{24}$$

and

$$\nabla^2_{\mathcal{G}_k} J(\mathcal{A}) = \frac{1}{N} C_k^{\top} D_{\mathcal{A}} C_k, \tag{25}$$

respectively, where $\mathbf{y}$ is defined in (18) and $\mathbf{1}$ denotes the all-ones vector in $\mathbb{R}^N$. Though we do not have a closed-form solution to update the core $\mathcal{G}_k$, the gradient and Hessian allows us to find the solution by efficient iterative methods, e.g. Newton's method whose convergence is at least quadratic in a neighbourhood of the solution. The quasi-Newton method, like the BFGS algorithm, is another good choice if the inverse of the Hessian is difficult to compute.

C. Regularization

The cost functions (15) and (22) of the two TT learning algorithms do not have any regularization term, which may result in overfitting and hence bad generalization properties of the obtained TT classifier. Next, we discuss how the addition of a regularization term to (15) and (22) results in a small modification of the small optimization problem that needs to be solved when updating the TT-cores $\mathcal{G}_k$.

Consider the regularized optimization problem

$$\tilde{J}(\mathcal{A}) = J(\mathcal{A}) + \gamma R(\mathcal{A}), \tag{26}$$

where $J(\mathcal{A})$ is given by (15) or (22), γ is a parameter that balances the loss function and the regularization term. Here we use the Tikhonov regularization, namely,

$$R(\mathcal{A}) = \frac{1}{2}\langle \mathcal{A}, \mathcal{A}\rangle. \tag{27}$$

Thanks to the TT structure, the gradient of $R(\mathcal{A})$ with respect to the TT-core $\mathcal{G}_k$ can be equivalently rewritten as a linear transformation of $\text{vec}(\mathcal{G}_k)$. In other words, there is a matrix $D_k \in \mathbb{R}^{r_{k-1}(n_k+1)r_k \times r_{k-1}(n_k+1)r_k}$ determined by the cores $\{\mathcal{G}_j\}_{j\neq k}$ such that $\nabla_{\mathcal{G}_k} R(\mathcal{A}) = D_k \text{vec}(\mathcal{G}_k)$. See Appendix A for more details. It follows that

$$\nabla_{\mathcal{G}_k}\tilde{J}(\mathcal{A}) = \nabla_{\mathcal{G}_k} J(\mathcal{A}) + \gamma D_k \text{vec}(\mathcal{G}_k)$$

and

$$\nabla^2_{\mathcal{G}_k}\tilde{J}(\mathcal{A}) = \nabla^2_{\mathcal{G}_k} J(\mathcal{A}) + \gamma D_k.$$

These small modifications lead to small changes when updating the core $\mathcal{G}_k$. For instance, the first-order condition of (26)

for the least squares model results in solving the modified linear system

$$\left(C_k^\top C_k + \frac{N}{2}\gamma D_k\right) \text{vec}(\mathcal{G}_k) = C_k^\top \mathbf{y}, \tag{28}$$

when compared with the original linear system (19).

D. Orthogonalization and Parallelization

The matrix C_k from (16) needs to be reconstructed for each TT-core $\mathcal{G}_k$ during the execution of the two TT learning algorithms. Fortunately, this can be done efficiently by exploiting the tensor train structure. In particular, after updating the core $\mathcal{G}_k$ in the left-to-right sweep, the new row vectors $\{\mathbf{p}_{k+1}(\mathbf{x}^{(j)})\}_{j=1}^N$ to construct the next matrix C_{k+1} can be easily computed from

$$\mathbf{p}_{k+1}(\mathbf{x}^{(j)}) = \mathcal{G}_k \times_1 \mathbf{p}_k(\mathbf{x}^{(j)}) \times_2 \mathbf{v}(x_k^{(j)})^\top.$$

Similarly, in the right-to-left sweep, the new column vectors $\{\mathbf{q}_{k-1}(\mathbf{x}^{(j)})\}_{j=1}^N$ to construct the next matrix C_{k-1} can be easily computed from

$$\mathbf{q}_{k-1}(\mathbf{x}^{(j)}) = \mathcal{G}_k \times_2 \mathbf{v}(x_k^{(j)})^\top \times_3 \mathbf{q}_k(\mathbf{x}^{(j)})^\top.$$

To make the learning algorithms numerically stable, the techniques of orthogonalization are also applied. The main idea is to make sure that before updating the core $\mathcal{G}_k$, the cores $\mathcal{G}_1, \ldots, \mathcal{G}_{k-1}$ are left-orthogonal and the cores $\mathcal{G}_{k+1}, \ldots, \mathcal{G}_d$ are right-orthogonal by a sequence of QR decompositions. In this way, the condition number of the constructed matrix C_k is upper bounded so that the subproblem is well-posed. After updating the core $\mathcal{G}_k$, we do an extra QR decomposition to orthogonalize it, and absorb the upper triangular matrix into the next core (depending on the direction of updating). More details on the orthogonalization step can be found in [25].

Another computational challenge is the potentially large size N of the training dataset. Luckily, the dimension of the optimization problem when updating $\mathcal{G}_k$ in the TT learning algorithms is $r_{k-1}(n_k+1)r_k$, which is much smaller and independent from N. We only need to compute the products $C_k^\top C_k$, $C_k^\top \mathbf{y}$, $C_k^\top \mathbf{h}_\mathcal{A}$ and $C_k^\top D_\mathcal{A} C_k$ in (19), (24) and (25). These computations are easily done in parallel. To be specific, given a proper partition $\{N_l\}_{l=1}^L$ satisfying $\sum_{l=1}^L N_l = N$, we divide the large matrix C_k into several blocks, namely,

$$C_k = \begin{bmatrix} C_k^{(1)} \\ C_k^{(2)} \\ \vdots \\ C_k^{(L)} \end{bmatrix} \in \mathbb{R}^{N \times r_{k-1}(n_k+1)r_k},$$

where $C_k^{(l)} \in \mathbb{R}^{N_l \times r_{k-1}(n_k+1)r_k}$, $l = 1, 2, \ldots, L$. Then, for example, the product $C_k^\top D_\mathcal{A} C_k$ can be computed by

$$C_k^\top D_\mathcal{A} C_k = \sum_{l=1}^L (C_k^{(l)})^\top D_\mathcal{A}^{(l)} C_k^{(l)},$$

where $D_\mathcal{A}^{(l)}$ denotes the corresponding diagonal block. Each term in the summation on the right-hand side of the above equation can be computed in parallel. The other matrix products can also be computed in a similar way.

We summarize our learning algorithms in Algorithm 2. Note that based on the decision strategy, an m-class problem is decomposed into a sequence of two-class problems whose TT classifiers can be trained in parallel.

Algorithm 2 Tensor Train Learning Algorithm

Input: Training dataset of pairs $\{(\mathbf{x}^{(j)}, y^{(j)})\}_{j=1}^N$, TT-ranks $\{r_k\}_{k=1}^{d-1}$, degree vector $\mathbf{n} = (n_1, n_2, \ldots, n_d)^\top \in \mathbb{N}^d$ and regularization parameter γ

Output: Tensor $\mathcal{A}$ in TT format with cores $\{\mathcal{G}_k\}_{k=1}^d$

```
Initialize right orthogonal cores {G_k}_{k=1}^d of prescribed ranks
while termination condition is not satisfied do
    for k = 1, 2, ..., d − 1 do
        G_k^* ← find the minimal solution of the regularized optimization problem (26) with respect to G_k
        U_k ← reshape(G_k^*, r_{k−1}(n_k + 1), r_k)
        [Q, R] ← compute QR decomposition of U_k
        G_k ← reshape(Q, r_{k−1}, n_k + 1, r_k)
        V_{k+1} ← R ∗ reshape(G_{k+1}, r_k, n_{k+1} + 1, r_{k+1})
        G_{k+1} ← reshape(V_{k+1}, r_k, n_{k+1} + 1, r_{k+1})
    end for
    Perform the right-to-left sweep
end while
```

We end this section with the following remarks:

- Other loss functions can also be used in the framework of tensor train learning provided that there exists an efficient way to solve the corresponding subproblems.
- The Density Matrix Renormalization Group (DMRG) method [24] can also be used to update the cores. This involves updating two cores at a time so that the TT-ranks are adaptively determined by means of a singular value decomposition (SVD). This may give better performance at the cost of a higher computational complexity. It also removes the need to fix the TT-ranks a priori.
- The local linear convergence of Algorithm 2 has been established in [25], [26] under certain conditions. In particular, if the TT-ranks are correctly estimated for convex optimization problems, then the obtained solution is guaranteed to be the global optimum. When choosing the TT-ranks, one should keep the upper bounds of the TT-ranks from Proposition 1 in mind.

V. Experiments

In this section, we test our TT learning algorithms on two popular digit recognition datasets: USPS and MNIST. All the algorithms were implemented in Matlab Version R2016a. The numerical experiments were done on a desktop PC with an Intel i5 quad-core processor running at 3.3GHz and 16GB of RAM.

Features of the handwritten digits are extracted through PCA by choosing a varying number of d principal components, resulting in a smaller feature space when compared to the dimension of the input samples. The TT classifiers are trained

with these principal components. We adopt the one-against-all decision strategy, where ten TT classifiers are trained to separate each digit from all the others. In the implementation of Algorithm 2, we normalize each initial core such that its Frobenius norm is equal to one. The degree vector is given by $n_1 = \cdots = n_d = n$. The TT-ranks are upper bounded by $r_{\max}$. The values of $d, n, r_{\max}$ were chosen to minimize the test error rate and to ensure that each of the subproblems to update the TT-cores $\mathcal{G}_k$ could be solved in a reasonable time. The dimension of each subproblem is at most $(n+1)\, r_{\max}^2$. For example, in the USPS case, we first fixed the values of n and $r_{\max}$. Test error rates were then found to be the smallest when d is located in the interval $[20, 30]$. We then fixed the value of d and incremented n and $r_{\max}$ to see whether this resulted in a better test error rate. We use the optimality criterion

$$\frac{|\tilde{J}(\mathcal{A}^+) - \tilde{J}(\mathcal{A})|}{|\tilde{J}(\mathcal{A})|} \leq 10^{-2},$$

where $\mathcal{A}^+$ is the updated tensor from tensor $\mathcal{A}$ after one sweep. And the maximum number of sweeps is 4, namely, $4(d-1)$ iterations through the entire training data are performed for each session. To simplify notations, we use "TTLS" and "TTLR" to denote the TT learning algorithms based on minimizing loss functions (15) and (22), respectively. For these two models, the regularization parameter γ is determined by the technique of 10-fold cross-validation. In other words, we randomly assign the training data to ten sets of equal size. The parameter γ is chosen so that the mean over all test errors is minimal.

The US Postal Service (USPS) database[1] contains 9298 handwritten digits, including 7291 for training and 2007 for testing. Each digit is a 16×16 image, represented as a 256-dimensional vector with entries between -1 and 1. It is known that the USPS test set is rather difficult and the human error rate is 2.5%. The numerical results are reported in Table I.

The Modified NIST (MNIST) database[2] of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a 28×28 image. Here, we use a reduced version by removing the 4 pixel padding around the digits, which results in a 20×20 image represented as a 400-dimensional vector with entries between 0 and 1. The numerical results are reported in Table II.

The monotonic decrease is always seen when training the ten TT classifiers. Fig. 4 shows the convergence of both TT learning algorithms on the USPS data for the case $d = 20, n = 1, r_{\max} = 8$ when training the classifier for the character "6". In addition, we also trained a polynomial classifier using least squares support vector machines (LSSVM [13]) on these two databases. The classifiers, using a polynomial kernel, were trained in Matlab with the LS-SVMlab toolbox. Using the basic SVM scheme, a training error rate of 0 and a test error rate of 8.37% were obtained for the USPS dataset after more than three and a half hours of computation. The MNIST dataset resulted in consistent out-of-memory errors, which is to be expected as the basic SVM scheme is not intended for large data sets.

[1]The USPS database is downloaded from http://statweb.stanford.edu/∼tibs/ElemStatLearn/data.html

[2]The MNIST database is downloaded from http://yann.lecun.com/exdb/mnist/

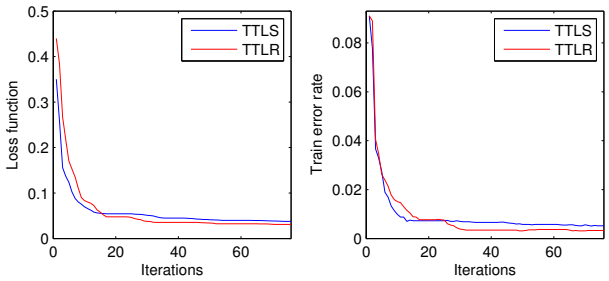


Fig. 4. The convergence of TT learning algorithms.

VI. Conclusion

This paper presents the framework of tensor train learning for pattern classification. Two efficient learning algorithms are proposed based on different loss functions. The numerical experiments show that each TT classifier is trained in up to several minutes with competitive test errors. We also mention that these results can be improved by adding virtual examples [8]. Future improvements are the implementation of on-line learning algorithms, together with the extension of the binary TT classifier to the multi-class case.

Appendix A

Given the degree vector $\mathbf{n} = (n_1, n_2, \ldots, n_d) \in \mathbb{N}^d$, let $\mathcal{A} \in \mathbb{R}^{(n_1+1)\times(n_2+1)\times\cdots\times(n_d+1)}$ be the tensor in TT format with cores $\mathcal{G}_k \in \mathbb{R}^{r_{k-1}\times(n_k+1)\times r_k}$, $k = 1, 2, \ldots, d$. To investigate the gradient of $R(\mathcal{A})$ in (27) with respect to the TT-core $\mathcal{G}_k$, we give a small variation ϵ to the ith element of $\mathrm{vec}(\mathcal{G}_k)$, resulting in a new tensor $\mathcal{A}_\epsilon$ given by

$$\mathcal{A}_\epsilon = \mathcal{A} + \epsilon \mathcal{I}_i^{(k)},$$

where $1 \leq i \leq r_{k-1}(n_k+1)r_k$ and $\mathcal{I}_i^{(k)}$ is the tensor which has the same TT-cores with $\mathcal{A}$ except that the vectorization of the core $\mathcal{G}_k$ is replaced by the unit vector in $\mathbb{R}^{r_{k-1}(n_k+1)r_k}$ with the ith element equal to 1 and 0 otherwise. Then we have

$$[\nabla_{\mathcal{G}_k} R(\mathcal{A})]_i = \lim_{\epsilon\to 0} \frac{R(\mathcal{A}_\epsilon) - R(A)}{\epsilon} = \langle \mathcal{A}, \mathcal{I}_i^{(k)} \rangle. \qquad (29)$$

On the other hand, by the definition of vectorization, the ith element of $\mathrm{vec}(\mathcal{G}_k) \in \mathbb{R}^{r_{k-1}(n_k+1)r_k}$ is mapped from the tensor element of $\mathcal{G}_k \in \mathbb{R}^{r_{k-1}\times(n_k+1)\times r_k}$ with indices $(\alpha_{k-1}, j_k, \alpha_k)$ satisfying

$$i = \alpha_{k-1} + (j_k - 1)r_{k-1} + (\alpha_k - 1)r_{k-1}(n_k + 1),$$

where $1 \leq \alpha_{k-1} \leq r_{k-1}$, $1 \leq j_k \leq n_k + 1$ and $1 \leq \alpha_k \leq r_k$. Denote by $E^{(\alpha_{k-1},\alpha_k)}$ the matrix in $\mathbb{R}^{r_{k-1}\times r_k}$ such that the element with index (α_{k-1}, α_k) equal to 1 and 0 otherwise. By simple computation, one can obtain that

$$\begin{aligned}\langle \mathcal{A}, \mathcal{I}_i^{(k)} \rangle &= \sum_{i_1, i_2, \ldots i_d} \mathcal{A}_{i_1 i_2 \cdots i_d} (\mathcal{I}_i^k)_{i_1 i_2 \cdots i_d} \\ &= \mathbf{a}_k \left(E^{(\alpha_{k-1},\alpha_k)} \otimes \mathcal{G}_k(j_k) \right) \mathbf{b}_k, \qquad (30)\end{aligned}$$

TABLE I
NUMERICAL RESULTS FOR DATASET USPS

			TTLS			TTLR		
d	n	$r_{\max}$	Train error	Test error	Time(s)	Train error	Test error	Time(s)
10	1	8	7.37%	12.6%	0.30×10	4.79%	10.9%	1.30×10
15	1	8	2.02%	7.72%	0.60×10	0.73%	7.37%	2.69×10
20	1	8	0.77%	6.63%	0.96×10	0.01%	7.57%	4.15×10
25	1	8	0.34%	6.58%	1.19×10	0.01%	6.43%	5.52×10
30	1	8	0.19%	7.08%	1.55×10	0	6.93%	6.83×10
35	1	8	0.09%	7.47%	1.82×10	0	7.22%	8.34×10
40	1	8	0.16%	7.62%	2.13×10	0	8.12%	9.80×10
20	2	8	0.34%	6.58%	1.75×10	0	6.78%	7.08×10
25	2	8	0.08%	6.58%	2.22×10	0.01%	6.43%	9.33×10
30	2	8	0.14%	7.37%	2.71×10	0.01%	6.78%	11.3×10
20	3	8	0.43%	5.83%	2.63×10	0.01%	6.68%	9.84×10
25	3	8	0.25%	7.03%	3.37×10	0.01%	6.63%	12.8×10
30	3	8	0.08%	7.32%	4.12×10	0.01%	7.22%	15.9×10
20	1	10	0.38%	6.48%	1.55×10	0	7.62%	14.7×10
25	1	10	0.12%	6.58%	2.04×10	0	6.13%	20.3×10
30	1	10	0.05%	6.98%	2.59×10	0	6.58%	25.7×10
20	4	8	0.48%	6.23%	3.32×10	0.08%	7.08%	12.6×10
20	3	9	0.40%	6.43%	3.25×10	0	6.33%	19.6×10

TABLE II
NUMERICAL RESULTS FOR DATASET MNIST

			TTLS			TTLR		
d	n	$r_{\max}$	Train error	Test error	Time(s)	Train error	Test error	Time(s)
10	1	8	16.8%	16.1%	2.45×10	8.98%	9.57%	5.75×10
15	1	8	6.99%	6.81%	5.24×10	3.77%	4.60%	11.0×10
20	1	8	4.49%	4.79%	6.98×10	2.27%	3.45%	16.6×10
25	1	8	3.44%	3.71%	9.60×10	1.83%	2.89%	21.8×10
30	1	8	2.90%	3.34%	11.5×10	1.47%	2.57%	27.6×10
35	1	8	2.54%	3.53%	15.5×10	1.21%	2.79%	32.6×10
40	1	8	2.32%	3.40%	16.2×10	1.14%	2.73%	37.5×10
20	2	8	4.42%	4.16%	11.9×10	1.66%	2.84%	27.6×10
25	2	8	3.59%	3.66%	15.5×10	1.29%	2.69%	36.7×10
30	2	8	2.96%	3.38%	19.1×10	0.97%	2.67%	44.1×10
35	2	8	2.72%	3.20%	22.3×10	0.84%	2.61%	52.3×10
40	2	8	2.60%	3.26%	25.9×10	0.67%	2.80%	60.7×10
20	3	8	4.72%	4.66%	17.2×10	1.71%	2.84%	39.4×10
25	3	8	3.72%	3.63%	22.4×10	1.28%	2.69%	51.3×10
30	3	8	3.26%	3.49%	27.9×10	0.97%	2.54%	62.7×10
35	3	8	2.79%	3.29%	32.1×10	0.77%	2.69%	74.8×10
40	3	8	2.59%	3.05%	37.9×10	0.65%	2.56%	85.9×10
25	1	10	3.09%	3.45%	13.8×10	0.96%	2.53%	46.0×10
30	1	10	2.37%	3.08%	17.2×10	0.66%	2.72%	57.9×10
35	1	10	2.00%	3.11%	20.8×10	0.54%	2.98%	70.5×10
40	1	10	1.82%	3.16%	24.4×10	0.39%	2.88%	82.6×10
40	2	10	2.30%	3.09%	40.5×10	0.15%	2.63%	135.2×10
40	3	10	2.34%	2.99%	57.6×10	0.17%	2.73%	189.5×10
40	4	10	2.32%	3.19%	78.0×10	0.23%	2.87%	250.3×10

where

$$\mathbf{a}_k = \prod_{l=1}^{k-1} \sum_{i_l=1}^{n_l+1} \left[\mathcal{G}_l(i_l) \otimes \mathcal{G}_l(i_l)\right] \in \mathbb{R}^{1\times r_{k-1}^2} \tag{31}$$

and

$$\mathbf{b}_k = \prod_{l=k+1}^{d} \sum_{i_l=1}^{n_l+1} \left[\mathcal{G}_l(i_l) \otimes \mathcal{G}_l(i_l)\right] \in \mathbb{R}^{r_k^2\times 1}. \tag{32}$$

Let $\mathbf{a}_k^{(1)}, \mathbf{a}_k^{(2)}, \ldots, \mathbf{a}_k^{(r_{k-1})} \in \mathbb{R}^{1\times r_{k-1}}$ be the row vectors such

that

$$\mathbf{a}_k = (\mathbf{a}_k^{(1)}, \mathbf{a}_k^{(2)}, \ldots, \mathbf{a}_k^{(r_{k-1})}) \in \mathbb{R}^{1 \times r_{k-1}^2},$$

and let $\mathbf{b}_k^{(1)}, \mathbf{b}_k^{(2)}, \ldots, \mathbf{b}_k^{(r_k)} \in \mathbb{R}^{r_k \times 1}$ be the column vectors such that

$$\mathbf{b}_k = \begin{bmatrix} \mathbf{b}_k^{(1)} \\ \mathbf{b}_k^{(2)} \\ \vdots \\ \mathbf{b}_k^{(r_k)} \end{bmatrix} \in \mathbb{R}^{r_k^2 \times 1},$$

Combining (29) and (30) together, we have

$$\begin{aligned}[\nabla_{\mathcal{G}_k} R(\mathcal{A})]_i &= \mathbf{a}_k^{(\alpha_{k-1})} \mathcal{G}_k(j_k) \mathbf{b}_k^{(\alpha_k)} \\ &= \left((\mathbf{b}_k^{(\alpha_k)})^\top \otimes (\mathbf{e}^{(j_k)})^\top \otimes \mathbf{a}_k^{(\alpha_{k-1})} \right) \text{vec}(\mathcal{G}_k),\end{aligned}$$

where $\mathbf{e}^{(j)} \in \mathbb{R}^{n_k+1}$ denotes the unit vector with the jth element equal to 1 and 0 otherwise. If we define the $r_{k-1}(n_k+1)r_k \times r_{k-1}(n_k+1)r_k$ matrix

$$D_k = \begin{bmatrix} (\mathbf{b}_k^{(1)})^\top \otimes (\mathbf{e}^{(1)})^\top \otimes \mathbf{a}_k^{(1)} \\ (\mathbf{b}_k^{(1)})^\top \otimes (\mathbf{e}^{(1)})^\top \otimes \mathbf{a}_k^{(2)} \\ \vdots \\ (\mathbf{b}_k^{(r_k)})^\top \otimes (\mathbf{e}^{(n_k+1)})^\top \otimes \mathbf{a}_k^{(r_{k-1})} \end{bmatrix}, \qquad (33)$$

it follows immediately that $\nabla_{\mathcal{G}_k} R(\mathcal{A}) = D_k \text{vec}(\mathcal{G}_k)$.

ACKNOWLEDGMENT

Johan Suykens acknowledges support of ERC AdG A-DATADRIVE-B (290923), KUL: CoE PFV/10/002 (OPTEC); FWO: G.0377.12, G.088114N, G0A4917N; IUAP P7/19 DYSCO.

REFERENCES

[1] H.-B. Shen and K.-C. Chou, "Ensemble classifier for protein fold pattern recognition," *Bioinformatics*, vol. 22, no. 14, pp. 1717–1722, 2006.

[2] H.-D. Cheng, X. Cai, X. Chen, L. Hu, and X. Lou, "Computer-aided detection and classification of microcalcifications in mammograms: a survey," *Pattern recognition*, vol. 36, no. 12, pp. 2967–2991, 2003.

[3] B. Roberto, *Template matching techniques in computer vision: theory and practice*. Wiley, Hoboken, 2009.

[4] B.-H. Juang, W. Hou, and C.-H. Lee, "Minimum classification error rate methods for speech recognition," *IEEE Transactions on Speech and Audio processing*, vol. 5, no. 3, pp. 257–265, 1997.

[5] L. Xu, A. Krzyzak, and C. Y. Suen, "Methods of combining multiple classifiers and their applications to handwriting recognition," *IEEE transactions on systems, man, and cybernetics*, vol. 22, no. 3, pp. 418–435, 1992.

[6] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*. John Wiley & Sons, 2012.

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

[8] D. Decoste and B. Schölkopf, "Training invariant support vector machines," *Machine learning*, vol. 46, no. 1-3, pp. 161–190, 2002.

[9] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[11] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[12] Y.-W. Chang, C.-J. Hsieh, K.-W. Chang, M. Ringgaard, and C.-J. Lin, "Training and testing low-degree polynomial data mappings via linear SVM," *Journal of Machine Learning Research*, vol. 11, no. Apr, pp. 1471–1490, 2010.

[13] J. A. K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle, *Least Squares Support Vector Machines*. World Scientific, Singapore, 2002.

[14] M. Signoretto, Q. T. Dinh, L. De Lathauwer, and J. A. K. Suykens, "Learning with tensors: a framework based on convex optimization and spectral regularization," *Machine Learning*, vol. 94, no. 3, pp. 303–351, 2014.

[15] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned cp-decomposition," *arXiv preprint arXiv:1412.6553*, 2014.

[16] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov, "Tensorizing neural networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 442–450.

[17] A. Novikov, M. Trofimov, and I. Oseledets, "Tensor train polynomial models via Riemannian optimization," *arXiv preprint arXiv:1605.03795*, 2016.

[18] E. M. Stoudenmire and D. J. Schwab, "Supervised learning with quantum-inspired tensor networks," *arXiv preprint arXiv:1605.05775*, 2016.

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

[20] I. Oseledets, "Tensor-train decomposition," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2295–2317, 2011.

[21] I. Oseledets and E. Tyrtyshnikov, "TT-cross approximation for multidimensional arrays," *Linear Algebra and its Applications*, vol. 432, no. 1, pp. 70–88, 2010.

[22] D. Savostyanov and I. Oseledets, "Fast adaptive interpolation of multi-dimensional arrays in tensor train format," in *2011 7th International Workshop on Multidimensional (nD) Systems (nDs)*. IEEE, 2011, pp. 1–8.

[23] S. Theodoridis and K. Koutroumbas, *Pattern Recognition, Fourth Edition*, 4th ed. Academic Press, 2008.

[24] S. R. White, "Density matrix formulation for quantum renormalization groups," *Physical Review Letters*, vol. 69, no. 19, p. 2863, 1992.

[25] S. Holtz, T. Rohwedder, and R. Schneider, "The alternating linear scheme for tensor optimization in the tensor train format," *SIAM Journal on Scientific Computing*, vol. 34, no. 2, pp. A683–A713, 2012.

[26] T. Rohwedder and A. Uschmajew, "On local convergence of alternating schemes for optimization of convex problems in the tensor train format," *SIAM Journal on Numerical Analysis*, vol. 51, no. 2, pp. 1134–1162, 2013.