Tensor Train Neighborhood Preserving Embedding

Wenqi Wang, Vaneet Aggarwal, and Shuchin Aeron

Abstract—In this paper, we propose a Tensor Train Neighborhood Preserving Embedding (TTNPE) to embed multi-dimensional tensor data into low dimensional tensor subspace. Novel approaches to solve the optimization problem in TTNPE are proposed. For this embedding, we evaluate novel trade-off gain among classification, computation, and dimensionality reduction (storage) for supervised learning. It is shown that compared to the state-of-the-arts tensor embedding methods, TTNPE achieves superior trade-off in classification, computation, and dimensionality reduction in MNIST handwritten digits and Weizmann face datasets.

I. INTRODUCTION

Robust feature extraction and dimensionality reduction are among the most fundamental problems in machine learning and computer vision. Assuming that the data is embedded in a low-dimensional subspace, popular and effective methods for feature extraction and dimensionality reduction are the Principal Component Analysis (PCA) [1], [2], and the Laplacian eigenmaps [3]. However, simply projecting data to a low dimensional subspace may not efficiently extract discriminative features. Motivated by a recent works [4]–[6] that demonstrate applying tensor factorization (after reshaping matrices to multidimensional arrays or tensors) improves data representation, we consider reshaping vision data into tensors and embedding the tensors into Kronecker structured subspaces, i.e. tensor subspaces, to further refine this subspace based approaches with significant gains. In this context, a very popular representation format namely Tucker format has shown to be useful for a variety of applications [7]–[11]. However, Tucker representation can still be exponential in storage requirements. In [12], it was shown that hierarchical Tucker representation, and in particular Tensor Train (TT) representation is a promising format for the approximation of solutions in high dimensional data and can alleviate the curse of dimensionality under fixed rank, which inspires us to investigate its application in efficient dimensionality reduction and embedding. Tensor train representation has also been shown to be useful for dimensionality reduction in [13]–[15].

In this paper, we begin by noting that TT decompositions are associated with a structured subspace model, namely the Tensor Train subspace [16]. Using this notion, we extend a popular approach, namely the Neighborhood Preserving Embedding (NPE) [17] for unsupervised classification of data. In the past, the NPE approach has been extended to exploit the Tucker subspace structure on the data [18], [19]. Here, we embed the data into a Tensor Train subspace and propose a computationally efficient Tensor Train Neighborhood Preserving Embedding (TTNPE) algorithm. We show that this approach achieves significant improvement in the storage of embedding and computation complexity for classification after embedding as compared to the embedding based on the Tucker representation in [18], [19]. We validate the approach on classification of MNIST handwritten digits data set [20] and Weizmann Facebase [21].

The rest of the paper is organized as follows. The technical notations and definitions are introduced in Section II. The Tensor Train subspace (TT-subspace) is described in Section III. In Section IV, the optimization problem for Tensor Train Neighborhood Preserving Embedding (TTNPE) is formulated. We then outline algorithms to solve the resulting problem highlighting the computational challenges and propose an approximate method to alleviate them. In Section V, we evaluate the proposed algorithm on MNIST handwritten digits and Weizmann databases. Section VI concludes the paper.

II. NOTATIONS AND PRELIMINARIES

Vectors and matrices are represented by boldface lower letters (e.g. \( \mathbf{x} \)) and boldface capital letters (e.g. \( \mathbf{X} \)), respectively. An \( n \)-order tensor is denoted by calligraphic letters \( \mathbf{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_n} \), where \( I_i, i = 1, 2, \ldots, n \) denotes the dimensionality along the \( i \)-th order. An element of a tensor \( \mathbf{X} \) is represented as \( \mathbf{X}(i_1, i_2, \ldots, i_n) \), where \( i_k, k = 1, 2, \ldots, n \) denotes the location index along the \( k \)-th order. A colon is applied to represent all the elements of an order in a tensor, e.g. \( \mathbf{X}(::, i_2, \ldots, i_n) \) represents the fiber along order 1 and \( \mathbf{X}(i_1, ::, i_3, i_4, \ldots, i_n) \) represents the slice along order 1 and order 2 and so forth. \( \mathbf{V}(\cdot) \) is a tensor vectorization operator such that \( \mathbf{X} \in \mathbb{R}^{I_1 \times \cdots \times I_n} \) is mapped to a vector \( \mathbf{V}(\mathbf{X}) \in \mathbb{R}^{I_1 \times \cdots \times I_n} \times \) and \( \otimes \) represent matrix product and kronecker product respectively. Let \( \mathbf{tr}_i \) be a tensor trace operation, which reduces 2 tensor orders by getting the trace along the slices formed by the \( i \)-th and \( j \)-th order (assuming \( I_i = I_j \)). As an example, let \( \mathbf{U} \in \mathbb{R}^{I_1 \times I_2 \times I_3} \) be a 3-mode tensor, then \( \mathbf{v} = \mathbf{tr}_1(\mathbf{U}) \in \mathbb{R}^{I_2} \) is given as \( \mathbf{v}(i_2) = \text{trace}(\mathbf{U}(i_2,:,:)) \).

We first introduce the tensor train decomposition.

Definition 1. (Tensor Train (TT) Decomposition) [12], [22] Each element of a \( n \)-mode tensor \( \mathbf{Y} \in \mathbb{R}^{I_1 \times \cdots \times I_n} \) in tensor train representation is generated by

\[
\mathbf{y}(i_1, \ldots, i_n) = \mathbf{U}_1(i_1,:)\mathbf{U}_2(:, i_2, :) \cdots \mathbf{U}_{n-1}(; i_{n-1,:}) \mathbf{U}_n(:, i_n),
\]

where \( \mathbf{U}_k \in \mathbb{R}^{I_k \times R_k} \) and \( \mathbf{U}_n \in \mathbb{R}^{R_{n-1} \times I_n} \) are the boundary matrices and \( \mathbf{U}_i \in \mathbb{R}^{R_{i-1} \times I_i \times R_i}, i = 2, \cdots, n - 1 \) are the decomposed tensors.

In this paper, we consider a tensor train decomposition for a \( n + 1 \) mode tensor \( \mathbf{X} \in \mathbb{R}^{I_1 \times \cdots \times I_n \times R_n} \), where each element...
Tensor merging product, $k$ be two

Let

The matrix product between $U$ and $R$ produces the matrix obtained by taking the first mode as row indices and the remaining $n$ mode as column indices such that $R(X) \in R^{t_2 \times t_n \times R_{n}}$. Similarly, the right unfolding operation produces the matrix obtained by taking the first mode as row indices and the remaining $n$ mode as column indices such that $R(X) \in R^{t_1 \times t_n \times R_{n}}$.

We further introduce a tensor operation and show the equivalence of tensor operations to matrix product.

**Definition 2. (Left and Right Unfolding)** Let $X \in R^{t_1 \times \cdots \times t_n \times R_{n}}$ be a $n + 1$ mode tensor. The left unfolding operation is the matrix obtained by taking the first $n$ modes as row indices and the last mode as column indices such that

$$L(X) \in R^{t_1 \times \cdots \times t_n \times R_{n}}.$$

Similarly, the right unfolding operation produces the matrix obtained by taking the 1st mode as row indices and the remaining $n$ modes as column indices such that $R(X) \in R^{t_1 \times (t_2 \cdots \times t_n \times R_{n})}$.

**Definition 3. (Tensor Merging Product)** Tensor merging product is an operation to merge the two tensors along the given sets of mode indices. Let $U_1 \in R^{t_1 \times \cdots \times t_{n-1}}$ and $U_2 \in R^{t_1 \times \cdots \times t_{n-2}}$ be two tensors. Let $g_1 \subseteq \{1, \cdots, n\}$ and $g_2 \subseteq \{1, \cdots, n\}$ be two vectors, with $|g_1| = |g_2| = k$, and $I_{g_1}(p) = I_{g_2}(p)$ for $p = 1, \cdots, k$. Then, the tensor merging product is $U_3 = U_1 \times_{g_1} U_2 \in R^{t_{g_1} \times t_{g_2}}$, which is a $n + n - 2k$ mode tensor given as

$$U_3(i_1 \forall t \not\in g_1, j_q \forall q \not\in g_2) = \sum_{d_1, \cdots, d_k} U_1(a_1, \cdots, a_m)U_2(b_1, \cdots, b_k)$$

for $w = 1, \cdots, k$, and $b_{g_1}(w) = d_w$ for $w = 1, \cdots, k$.

Based on tensor merging product, we note that recovering a tensor from tensor train decomposition is a process of applying tensor merging product on tensor train factorizations. Let $R_0 = 1$, $U_i \in R^{t_{i-1} \times t_i}$, $i = 1, \cdots, n$, be $n$ 3rd order tensors. The recovery of the $n + 1$ order tensor is defined as

$$(U = U_1 \times_{1} U_2 \times_{2} \cdots \times_{n} U_n) \in R^{t_1 \times \cdots \times t_n \times R_{n}}.$$

**Proof.** The $(m, n)^{th}$ entry of $A \times_{2}^k B$ is

$$(A \times_{2}^k B)_{m,n} = \sum_{r_2, \cdots, r_k} A(m, r_1, \cdots, r_k)B(r_1, \cdots, r_k, n),$$

which is the same as the $(m, n)^{th}$ entry of $A \times B$.

**III. Tensor Train Subspace (TTS)**

A tensor train subspace, $S_{TTS} \subseteq R^{t_1 \times t_2 \times \cdots \times t_n}$, is defined as the span of an $n$-order tensor that is generated by the tensor merging product of a sequence of 3-order tensors. Specifically,

$$S_{TTS} = \{U_1 \times_{1} U_2 \times_{2} \cdots \times_{n} U_n | a\forall a \in R^{t_n}\}.$$

**Lemma 2.** (Subspace Property) $S_{TTS}$ is a $R_n$ dimensional subspace of $R^{t_1 \times \cdots \times t_n}$ for a given set of MPSs, $\{U_1, U_2, \cdots, U_n\}$.

We next briefly outline some useful properties of the TT decomposition that will be used in this paper.

**Lemma 3.** (Left-Orthogonality Property) If $U_i$ is left-orthogonal for all $i = 1, \cdots, n$, then $L(U_1) \times \cdots \times L(U_n)$ is left-orthogonal for all $1 \leq j \leq n$.

**Proof.** Let $B_j = L(U_1) \times \cdots \times L(U_j)$, Then, $B_{j+1} = (L_{j+1} \otimes B_j)\times L(U_{j+1})$. Using this, and induction (since the result holds for $j = 1$), the result follows.

We can without loss of generality, assume that $L(U_i)$ are left-orthogonal for all $i$. Then, the projection of a data point $y \in R^{t_n}$ on the subspace $S_{TTS}$ is given by $L(U_1) \times \cdots \times L(U_n)\times T \times T \times \cdots \times T \times T \times Y$.

**IV. Tensor Train Neighborhood Preserving Embedding (TTNE)**

Given a set of tensor data $X_i \in R^{t_1 \times \cdots \times t_n}$, $i = 1, \cdots, N$, we wish to project the data $X_i$ to vector $t_i \in R^{t_n}$, satisfying $t_i = L(U_1) \times \cdots \times L(U_n)\times T \times Y$ and preserving neighborhood among the projected data. We first construct a neighbor graph to capture the neighborhood information in the given data and generate the affinity matrix $F$ as

$$F_{ij} = \begin{cases} \exp(-\|X_i - X_j\|^2/\epsilon), & \text{if } X_i, X_j \in O(K, X_i) \\ 0, & \text{otherwise} \end{cases}$$
where $O(K, X_i)$ denotes the subset of data excluding $X_i$ that are within the $K$-nearest neighbors of $X_i$, and $\epsilon$ is the scaling factor. By definition, $F_{ii} = 0$. We also note that this is an unsupervised tensor embedding method since the label information is not used in the embedding procedure. Without loss of generality, we set $S = F + F^T$ and $S$ is further normalized such that each row sum is one.

The goal is to find the decomposition $U_1, \ldots, U_n$ that minimizes the average distance between all the points and their weighted combination of remaining points, weighted by the symmetric affinity matrix in the projection, i.e.

$$
\min_{U_k, \forall k=1, \ldots, n} \sum_{i} \|L^T (U_1 x_1^T \cdots x_3^T U_n) V(X_i) - S_{ij} L^T (U_1 x_1^T \cdots x_3^T U_n) V(X_j)\|_2^2.
$$

Let $D \in \mathbb{R}^{I_1 \cdots I_n \times N}$ be the matrix that concatenates the $N$ vectorized tensor data such that the $i^{th}$ column of $D$ is $V(X_i)$, and let $E = L(U_1 x_1^T \cdots x_3^T U_n)$. Then, (10) is equivalent to

$$
\min_{U_k, \forall k=1, \ldots, n} \|E^T(D - DS^T)\|_F^2.
$$

Since $D - DS^T \in \mathbb{R}^{I_1 \cdots I_n \times N}$ is determined, we set $Y = D - DS^T$. Thus the Frobenius norm in (11) can be further expressed in the form of matrix trace to reduce the problem to

$$
\min_{U_k, \forall k=1, \ldots, n} \text{tr}(Y^T EE^T Y).
$$

Based on the cyclic permutation property of the trace operator, (12) is equivalent to

$$
\min_{U_k, \forall k=1, \ldots, n} \text{tr}(E^T Y Y^T E).
$$

Let $Z = YY^T \in \mathbb{R}^{(I_1 \cdots I_n) \times (I_1 \cdots I_n)}$ be the constant matrix. Then, the problem (13) becomes

$$
\min_{U_k, \forall k=1, \ldots, n} \text{tr}(E^T Z E).
$$

We will use the alternating minimization method [23] to solve (14) such that each $U_k$ is updated by solving

$$
\min_{U_k \in U_k} \text{tr}(E^T Z E).
$$

In order to solve (15), we use an iterative algorithm. Each $U_{k; k=1, \ldots, n}$ is initialized by tensor train decomposition [22] with a thresholding parameter $\tau$, such that tensor train ranks $(R_1, \cdots, R_n)$ are determined. Note that when $\tau = 0$, TTNEP is reduced to PCA.

**A. Tensor Train Neighbor Preserving Embedding using Tensor Network (TTNEP-TN)**

Let $Z \in \mathbb{R}^{I_1 \cdots I_n \times I_1 \cdots I_n}$ be the reshaped tensor of $Z$, and

$$
\begin{align*}
\mathcal{T}_1 &= U_1 \times_1 \cdots \times_3 U_{k-1} \in \mathbb{R}^{I_1 \times \cdots \times I_{k-1} \times R_{k-1}}, \\
\mathcal{T}_n &= U_{k+1} \times_1 \cdots \times_3 U_n \in \mathbb{R}^{R_k \times I_k \times \cdots \times I_n \times R_n},
\end{align*}
$$

For updating $U_{k; k=1, \cdots, n-1}$, based on Lemma [11] we note that (15) can be written as

$$
\begin{align*}
&\min_{U_k \in U_k} U_k \overset{6,1,5}{\times} 1,2,3 \text{tr}^2 (Z) \times_{n+k+1, \ldots, 2n} \mathcal{T}_n \times_{1, \cdots, k-1} \mathcal{T}_1 \\
&\quad \overset{1,2,3}{\times} 3,1,2 U_k.
\end{align*}
$$

Let $\mathcal{A} \in \mathbb{R}^{R_{k-1} \times I_k \times R_k \times R_{k-1} \times I_{k-1}}$ be the 6-order tensor, given as $\text{tr}_4^2 (Z) \times_{n+k+1, \ldots, 2n} \mathcal{T}_n \times_{1, \cdots, k-1} \mathcal{T}_1$, then (17) becomes

$$
\min_{U_k \in U_k} U_k \overset{6,1,5}{\times} 1,2,3 \mathcal{A} \times_{3,1,2} U_k.
$$

Based on Lemma [11] the tensor merging product (18) can be transformed into matrix product. Thus, (18) becomes

$$
\min_{U_k \in U_k} \text{tr}(V(U_k)^T A V(U_k)),
$$

where $A \in \mathbb{R}^{(R_{k-1} \times I_k \times R_k \times R_{k-1})}$ is the reshaped form of $A$. A differentiable function under unitary constraint can be solved by the algorithm proposed in [24]. In problem (19), the gradient of objective function to $V(U_k)$ is $2A V(U_k)$.

**Updating $U_n$** is different from solving $U_{k; k=1, \cdots, n-1}$ since the trace operation, which is represented by the red edge highlighted in Fig.1 merges the tensor $U_n$ with itself, thus (19) does not apply for solving $U_n$. Instead, updating $U_n$ in (15) is equivalent to solving

$$
\min_{U_n \in U_n} \text{tr}_4^2 (U_n \times_{3,1,2} \mathcal{B} \times_{3,1,2} U_n),
$$

which by Lemma [11] can be transformed into the matrix form

$$
\min_{U_n \in U_n} \text{tr}(L(U_n)^T B L(U_n)),
$$

where $B \in \mathbb{R}^{(R_{n-1} \times I_n) \times (R_{n-1} \times I_n)}$ is reshaped from $B$. The gradient of the objective function to $L(U_n)$ is $2B L(U_n)$.

We now analyze the **computation and memory complexity** of TTNEP-TN algorithm. For $U_{k; k=1, \cdots, n-1}$, the generation of $A$ requires merging the tensor networks, which has a computation complexity of $O \left( \left( I_1 \cdots I_n \right)^2 R_k R_k + I_1 \cdots I_n \right)^2 (R_k R_k + I_1 \cdots I_n)^2 R_k R_k$, and solving (19) takes $O \left( R_k R_k R_k \right)$ time. Thus, the computation of $A$ dominates the complexity. The memory requirement for generating $A$ is $O \left( R_k R_k R_k \right)^2$, which is large when the tensor train ranks are high. Similarly, the generation of $B$ to solve $U_n$ takes $O \left( \left( I_1 \cdots I_n \right)^2 R_n R_n \right)$ time and solving (22) takes $O \left( R_n R_n R_n \right)$, and the memory for generating $B$ is $O \left( (I_1 \cdots I_n)^2 \right)$, indicating solving for $U_n$ is
less expensive than that for solving for $\mathbf{U}_k$ in terms of both memory and computation complexity.

Although TTNPE-TN algorithm gives an exact solution for updating $\mathbf{U}_i$ in each alternating minimization step, the memory and computation cost prohibits its application when the tensor train ranks are large. In order to address this, we propose a Tensor Train Neighbor Preserving Embedding using Approximate Tensor Network (TTNPE-ATN) algorithm in the next section, to approximate (15), aiming to reduce computation and memory cost.

B. Tensor Train Neighbor Preserving Embedding using Approximated Tensor Network (TTNPE-ATN)

Our main intuition is as follows. Without the TT decomposition constraint, the solution to minimize the quadratic form $\text{tr} (\mathbf{E}^\top \mathbf{Z} \mathbf{E})$ where $\mathbf{E}$ is unitary is given by $\mathbf{E}$ being the matrix formed by eigenvectors corresponding to the lowest eigenvalues of $\mathbf{Z}$ and the value of the objective is the sum of the lowest eigenvalues of $\mathbf{Z}$ (25). Let the matrix corresponding to the eigenvectors corresponding to $r_n$ least eigenvalues of $\mathbf{Z}$ be $\mathbf{V}_{r_n}$. With the additional constraint that $\mathbf{E}$ has TT decomposition, the above choice of $\mathbf{E}$ may not be optimal. Thus, we relax the original problem to minimize the distance between $\mathbf{E}$ and $\mathbf{V}_{r_n}$. Thus, the relaxed problem of (15) is

$$\min_{\mathbf{L}(\mathbf{U}_k) \text{ is unitary}} \| \mathbf{L}(\mathbf{U}_1 \times_1 \cdots \times_3 \mathbf{U}_n) - \mathbf{V}_{r_n} \|^2_F,$$  

(23)

where $\mathbf{L}(\mathbf{U}_1 \times_1 \cdots \times_3 \mathbf{U}_n), \mathbf{V}_{r_n} \in \mathbb{R}^{(I_1 \cdots I_n) \times r_n}$.

Let $\mathbf{T}_k$ be a reshaping operator that changes the dimension from $\mathbb{R}^{(I_1 \cdots I_n) \times r_n} \rightarrow \mathbb{R}^{(I_1 \cdots I_k) \times (I_{k+1} \cdots I_n) r_n}$, thus (25) is equivalent to

$$\min_{\mathbf{u}_k: \mathbf{L}(\mathbf{U}_k) \text{ is unitary}} \| \mathbf{T}_k (\mathbf{L}(\mathbf{T}_1 \times_1 \mathbf{U}_k \times_3 \mathbf{T}_n)) - \mathbf{T}_k (\mathbf{V}_{r_n}) \|^2_F,$$

which is equivalent to

$$\min_{\mathbf{u}_k: \mathbf{L}(\mathbf{U}_k) \text{ is unitary}} \| (\mathbf{I}_k \otimes \mathbf{L}(\mathbf{T}_1)) \mathbf{L}(\mathbf{U}_k) \mathbf{R}(\mathbf{T}_n) - \mathbf{T}_k (\mathbf{V}_{r_n}) \|^2_F,$$

(24)

which is to minimize $\| \mathbf{AXB} - \mathbf{C} \|^2_F$ under unitary constraint. Since the gradient is $\mathbf{A}^\top (\mathbf{AXB} - \mathbf{C}) \mathbf{B}^1$, (25) can be solved by the algorithm proposed in (24).

After the relaxation, the computation complexity is $O(R_{k-1} I_1 \cdots I_n R_n)$ for calculating the gradient, $O((I_1 \cdots I_n) r_n^2)$ for generating $\mathbf{V}_{r_n}$, and $O(R_{k-1} I_k r_k^2)$ for solving (25). Thus the eigenvalue decomposition for generating $\mathbf{V}_{r_n}$ dominates the computational complexity. The memory for generating $\mathbf{A}$ and $\mathbf{B}$ is $\max(I_1 \cdots I_{k-1} R_{k-1}, R_k I_{k+1} \cdots I_n R_n)$.

C. Classification Using TTNPE-TN and TTNPE-ATN

The classification is conducted by first solving a set of MPS $\mathbf{U}_1, \ldots, \mathbf{U}_n$. Then, the training data and testing data is projected onto the tensor train subspace bases as follows:

$$\mathbf{t}_i = \mathbf{L}(\mathbf{u}_1 \times_1 \cdots \times_3 \mathbf{u}_n)^\top \mathbf{V} (\mathbf{x}_i) \in \mathbb{R}^{R_n}.$$  

(26)

Any data point in the testing set is labeled by applying $k$-nearest neighbors (KNN) classification with $K$ neighbors in the embedded space $\mathbb{R}^{R_n}$.

D. Storage and Computation Complexity

In this section, we will analyze the amount of storage to store the high dimensional data, complexity for finding the embedding using TTNPE-ATN and the cost of projection onto the TT subspace for classification. KNN and TNPE algorithms are considered for comparison. Note that finding a subspace by PCA is a special case of TTNPE-ATN, thus PCA is not selected for comparison. For the computational complexity analysis, let $d$ be the data dimension, $n$ and $r$ be the reshaped tensor order and rank in TTNPE-ATN model, $K$ be the number of neighbors, and $N_\text{tr}$ ($N_\text{te}$) be the total training (testing) data. We assume the dimension along each tensor mode is the same, thus each tensor mode is $d^{\frac{r}{s}}$ in dimension.

Storage of data: Under KNN model, the storage required for $N_\text{tr}$ training data is $\text{Storage(KNN)} = d N_\text{tr}$. Under TNPE model, the storage for the $N_\text{tr}$ training data needs the space

---

Algorithm 1 TTNPE-TN and TTNPE-ATN Algorithms

**Input:** A set of $N$ tensors $\mathbf{x}_i = 1, \ldots, N \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_n}$, denoted as $\mathbf{x}$, threshold parameter $\tau$, kernel parameter $\sigma$, number of neighbors $K$, thresholding parameter $\tau$, and max iterations $\text{maxIter}$

**Output:** Tensor train subspace factors $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n$

1: Compute affinity matrix $\mathbf{F}$ by

$$F_{ij} = \begin{cases} \exp(-||\mathbf{x}_i - \mathbf{x}_j||^2_F/\epsilon), & \text{if } \mathbf{x}_i, j \in \mathcal{O}(K, \mathbf{x}_i), \\ 0, & \text{otherwise,} \end{cases}$$

2: Form $\mathbf{D} \in \mathbb{R}^{(I_1 \cdots I_n) \times N}$ as a reshape of the input data, compute $\mathbf{Y} = \mathbf{D} - \mathbf{DS}^\top$, and compute $\mathbf{Z} = \mathbf{YY}^\top$.

3: Apply tensor train decomposition (22) on $\mathbf{X}$ to initialize $\mathbf{U}_i = \mathbf{x}_i, \ldots, \mathbf{U}_n$ with thresholding parameter $\tau$, and the tensor train ranks are determined based on selection of $\tau$.

4: Solve $\mathbf{V}_{R_0}$, by applying eigenvalue decomposition on $\mathbf{Z}$.

5: Set $\text{iter} = 1$

6: while $\text{iter} \leq \text{maxIter}$ or convergence of $\mathbf{U}_i, \ldots, \mathbf{U}_n$

7: for $i = 1$ to $n$

8: (TTNPE-TN) Update $\mathbf{U}_i$ in (19) for $i < n$ and in (22) for $i = n$, using the algorithm proposed in (24).

9: (TTNPE-ATN) Update $\mathbf{U}_i$ in (25) by algorithm proposed in (24).

10: end for

11: $\text{iter} = \text{iter} + 1$

12: end while
for $n$ linear transformation which is $n(d^2 r - r^2)$, and the space for $N_r$ embedded training data of size $N_r r^n$, requiring the total storage $\text{Storage(TNPE)} = r^n N_r + n d^2 r$. Under TTNPE-ATN model, we need space $(n-1)(d^2 r^2 - r^2) + (d^2 r - r^2)$ \[12\] to store the projection bases $U_1, \ldots, U_n$, and $N_r r$ to store the embedded training data. Thus the total storage is $\text{Storage(TTNPE-ATN)} = (n-1)(d^2 r^2 - r^2) + (d^2 r - r^2) + r N_r$. We consider a metric of normalized storage, compression ratio, which is the ratio of storage required under the embedding method and storage for the entire data, calculated by $\rho_{ST} = \frac{\text{Storage(TN)}}{N_r d}$, where ST can be any of KNN, TNPE, TTNPE-ATN.

**Computation Complexity for estimating the embedding subspace** Under KNN model, data is directly used for classification and there is no embedding process. Under TNPE model, the embedding needs 3 steps, where solving $n$ linear transformations takes $O(N_r r d)$ for embedding raw data, matrix generation for an eigenvalue problem takes $O(N_r r^{2n})$, and eigenvalue decomposition for updating each linear transformation takes $r^3 n$, giving a total computational complexity $O(n(N_r r d + N_r r^{2n} + r^3 n))$. Under TTNPE-ATN model, the embedding takes 3 steps, where the initialization by tensor train decomposition algorithm takes $O(n d^2 r^2)$, the generation of $Z$ takes $O(d N_r^2 + d^2 N_r)$, and updating $U_r$, which includes a gradient calculation by merging a tensor network, takes $O(n d^2 r^3)$, thus giving a total computational complexity $O(n d^2 r^3 + d N_r^2 + d^2 N_r)$.

**Classification Complexity** Under KNN model, classification is conducted by pair-wise computations of the distance between a testing point with all training points, which has a computational complexity of $O(N_n N_r d)$. Under TNPE model, an extra time is required for embedding the testing data, which is $O(r^2 d N_r)$. However, less time is needed in classification by applying KNN in a reduced dimension, which is $O(N_r N_r r^n)$. Thus the total complexity is $O(r^2 d N_n + N_r N_r r^n)$. Similarly, under TTNPE-ATN algorithm, embedding takes an extra computation time of $O(N_r r^2 d)$, but a significantly less time used in classification, which is $O(N_r N_r r^n)$. Thus the total complexity is $O(N_r r^2 d + N_r N_r r^n)$.

The comparison of the three algorithms is shown in Table I where TTNPE-ATN exhibits a great advantage in storage and computation for classification after embedding.

### V. Experiment Results

In this section, we test our proposed tensor embedding on image datasets, where the 2D images are reshaped into multi-mode tensors. Reshaping images to tensors is a common practice to compare tensor algebraic approaches \[27\] since it exhibits improved data representation. The embedding is evaluated based on KNN classification, where an effective embedding that preserves neighbor information would give classification results close to that of KNN classification at lower compression ratios. We compare the proposed TTNPE-ATN algorithm with Tucker decomposition based neighbor preserving embedding (TNPE) algorithm as proposed in \[19\]. We further note that the authors of \[19\] compared their approach with different approaches based on vectorization of data, including Neighborhood Preserving Embedding (NPE), Locality Preserving Projection (LPP), and Local Discriminant Embedding (LDE). Since the approach in \[19\] was shown to outperform these approaches, we do not consider these vectorized data approaches in our comparison. Note that the tensor train rank, which determines the compression ratio, is learnt from the Algorithm \[1\] based upon the selection of $r \in (0, 1]$.

#### A. Weizmann Face Database

Weizmann Face Database \[21\] is a dataset that includes 26 human faces with different expressions and lighting conditions. 66 images from each of the 10 randomly selected people are used for multi-class classification, where 20 images from each person are selected for training and the remaining images are used for testing. The experiment is repeated 10 times (for the same 10 people, but random choices of the 20 training images per person) and the averaged classification errors are shown in Fig. 2. Each image is down sampled to $64 \times 44$ for ease of computation and is further reshaped to a 5-mode tensor of dimension $4 \times 4 \times 4 \times 4 \times 11$ to apply the TNPE and TTNPE-ATN algorithms. 10, 50, and 100 neighbors are considered to build the graph (from left to right) and the KNN from the same number of neighbors in the embedded space are used for classification. Since KNN does not compress the data, it results in a single point at a compression ratio of 1.

We show that TTNPE-ATN performs better than TNPE when the compression ratio is lower than 0.9, indicating TTNPE-ATN better captures the localized features in the dataset thus yielding better embedding under low compression ratios. With the increase of compression ratio, the classification error for TTNPE-ATN algorithm first decreases, which is because the data structure can be better captured with increasing compression ratio (lower compression). The classification error then increases with compression ratio since the embedding overfits the background noise in the images. Similar trend happens for TNPE algorithm. We note that for a compression ratio of 1, the result for TTNPE-ATN do not match that of KNN since we are learning at-most 200-rank space (due to 20 training images for each of 10 people) while the overall data dimension is $64 \times 44$, thus giving an approximation at the compression ratio of 1. Increasing $K$ helps preserve more neighbors for embedding, and the neighbor structure is

---

**TABLE I:** Storage and Computation Complexity Analysis for Embedding Methods. The bold entry in each column depicts the lowest order.

| Method      | Storage          | Subspace Computation | Classification |
|-------------|------------------|----------------------|----------------|
| KNN         | $d N_n$          | $O(n (N_r r + N_r r^{2n} + r^n))$ | $O(N_r N_r d)$ |
| TNPE        | $r^n N_n + n d^2 r$ | $O(n (N_r r + N_r r^{2n} + r^n))$ | $O(N_r r^2 d + N_r N_r r^n)$ |
| TTNPE-ATN   | $(n-1)(d^2 r^2 - r^2) + (d^2 r - r^2)$ | $O(n d^2 r^3 + d N_r^2 + d^2 N_r)$ | $O(N_r r^2 d + N_r N_r r^n)$ |
training samples is large. Each image is reshaped to investigate the embedding performance when the number of tensor. We perform binary classification for digits handwritten digits of size \(28\times 28\).

**B. MNIST Dataset**

We use the MNIST dataset [20], which consists 60000 handwritten digits of size \(28 \times 28\) from 0 to 9, to further investigate the embedding performance when the number of training samples is large. Each image is reshaped to \(4 \times 7 \times 4 \times 7\) tensor. We perform binary classification for digits 1 and 2 by using 600 training samples from each digit. Figure 3 shows the classification performance of the three algorithms (KNN on data directly, TNPE, and TTNPE-ATN) when different values of \(K \in \{3, 5, 7\}\) neighbors are used to construct the graph (from left to right). The same value of \(K\) is used for classification in the embedded space. 1000 out of sample images from each digit are selected for testing. The results in Fig. 3 are averaged over 10 independent experiments (over the choice of 600 training and 1000 test samples).

We first note that the proposed TTNPE-ATN is the same as the standard KNN for that point when the training sample size is sufficient large (since the number of training samples do not limit the performance). Further, as the compression ratio increases, the classification error of the proposed TTNPE-ATN decreases first, since TTNPE-ATN model can effectively capture the embedded data structure. The classification error then increases since it fits the inherent noise as compared to the low TT-rank approximation of the data. Overall, TTNPE-ATN algorithm shows comparable embedding performance as TNPE algorithm in the compression ratio region around 0.1, outperforms TNPE for higher compression ratios (lesser compression), and converges to KNN results at compression ratio of 1.
VI. CONCLUSION

This paper proposes a novel algorithm for non-linear Tensor Train Neighborhood Preserving Embedding (TTNPE-ATN) for tensor data classification. We investigate the tradeoffs between error, storage, and computation and evaluate the method on several vision datasets. We further show that TTNPE-ATN algorithm exhibits improved classification performance and better dimensionality reduction among the baseline approaches, and has lower computational complexity as compared to Tucker neighborhood preserving embedding method. In the future, we will investigate the convergence of tensor network optimization and provide the theoretical gap between TTNPE-ATN and TTNPE-TN.

REFERENCES

[1] I. Jolliffe, *Principal component analysis*. Wiley Online Library, 2002.
[2] C. M. Bishop, “Pattern recognition,” *Machine Learning*, vol. 128, 2006.
[3] M. Belkin and P. Niyogi, “Laplacian eigenmaps for dimensionality reduction and data representation,” *Neural computation*, vol. 15, no. 6, pp. 1373–1396, 2003.
[4] H. N. Phien, H. D. Tuan, J. A. Bengua, and M. N. Do, “Efficient tensor completion: Low-rank tensor train,” *arXiv preprint arXiv:1601.01083*, 2016.
[5] W. Wang, V. Aggarwal, and S. Aeron, “Tensor completion by alternating minimization under the tensor train (tt) model,” *arXiv preprint arXiv:1609.05587*, 2016.
[6] ——, “Efficient low rank tensor ring completion,” in *ICCV*, 2017.
[7] L. De Lathauwer, B. De Moor, and J. Vandewalle, “A multilinear singular value decomposition,” *SIAM Journal on Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1253–1278, 2000.
[8] D. L. Donoho, “De-noising by soft-thresholding,” *IEEE transactions on information theory*, vol. 41, no. 3, pp. 613–627, 1995.
[9] H. Lu, K. N. Plataniotis, and A. N. Venetsanopoulos, “Multilinear principal component analysis of tensor objects for recognition,” in *18th International Conference on Pattern Recognition (ICPR’06)*, vol. 2. IEEE, 2006, pp. 776–779.
[10] M. A. O. Vasilescu and D. Terzopoulos, “Multilinear subspace analysis of image ensembles,” in *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, vol. 2. IEEE, 2003, pp. II–93.
[11] R. Zeng, J. Wu, Z. Shao, L. Senhadj, and H. Shu, “Multilinear principal component analysis network for tensor object classification,” *arXiv preprint arXiv:1411.1171*, 2014.
[12] S. Holtz, T. Rohwedder, and R. Schneider, “On manifolds of tensors of fixed tr-rank,” *Numerische Mathematik*, vol. 120, no. 4, pp. 701–731, 2012.
[13] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov, “Tensorizing neural networks,” in *Advances in Neural Information Processing Systems*, 2015, pp. 442–450.
[14] T. Garipov, D. Podoprikhin, A. Novikov, and D. Vetrov, “Ultimate tensorization: compressing convolutional and fc layers alike,” *arXiv preprint arXiv:1611.03214*, 2016.
[15] A. Tjandra, S. Sakti, and S. Nakamura, “Compressing recurrent neural network with tensor train,” *arXiv preprint arXiv:1705.08052*, 2017.
[16] W. Hackbusch, *Tensor spaces and numerical tensor calculus*. Springer Science & Business Media, 2012, vol. 42.
[17] X. He, D. Cai, S. Yan, and H.-J. Zhang, “Neighborhood preserving embedding,” in *Tenth IEEE International Conference on Computer Vision (ICCV’05) Volume 1*, vol. 2. IEEE, 2005, pp. 1208–1213.
[18] X. He, D. Cai, and P. Niyogi, “Tensor subspace analysis,” in *Advances in neural information processing systems*, 2005, pp. 499–506.
[19] G. Dai and D.-Y. Yeung, “Tensor embedding methods,” in *Proceedings of the National Conference on Artificial Intelligence*, vol. 21, no. 1. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006, p. 330.
[20] 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.
[21] “Weizmann facebase.” [http://ftp.idc.ac.il/pub/users/cs/yael/facebase](http://ftp.idc.ac.il/pub/users/cs/yael/facebase).
[22] I. V. Oseledets, “Tensor-train decomposition,” *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2295–2317, 2011.
[23] A. Beck, “On the convergence of alternating minimization for convex programming with applications to iteratively reweighted least squares and decomposition schemes,” *SIAM Journal on Optimization*, vol. 25, no. 1, pp. 185–209, 2015.
[24] Z. Wen and W. Yin, “A feasible method for optimization with orthogonality constraints,” *Mathematical Programming*, vol. 142, no. 1-2, pp. 397–434, 2013.
[25] M. S. Moslehian, “Ky fan inequalities,” *Linear and Multilinear Algebra*, vol. 60, no. 11-12, pp. 1313–1325, 2012.
[26] N. S. Altman, “An introduction to kernel and nearest-neighbor non-parametric regression,” *The American Statistician*, vol. 46, no. 3, pp. 175–185, 1992.
[27] Q. Zhao, G. Zhou, S. Xie, L. Zhang, and A. Cichocki, “Tensor ring decomposition,” *arXiv preprint arXiv:1606.05535*, 2016.