Fast Recursive Low-rank Tensor Learning for Regression

Ming Hou and Brahim Chaib-draa
Department of Computer Science and Software Engineering, Laval University, Quebec, Canada
ming.hou.1@ulaval.ca, brahim.chaib-draa@ift.ulaval.ca

Abstract

In this work, we develop a fast sequential low-rank tensor regression framework, namely recursive higher-order partial least squares (RHOPLS). It addresses the great challenges posed by the limited storage space and fast processing time required by dynamic environments when dealing with large-scale high-speed general tensor sequences. Smartly integrating a low-rank modification strategy of Tucker into a PLS-based framework, we efficiently update the regression coefficients by effectively merging the new data into the previous low-rank approximation of the model at a small-scale factor (feature) level instead of the large raw data (observation) level. Unlike batch models, which require accessing the entire data, RHOPLS conducts a blockwise recursive calculation scheme and thus only a small set of factors is needed to be stored. Our approach is orders of magnitude faster than other sequential methods while maintaining a highly comparable predictability with the best batch methods, as verified on challenging real-life tasks.

1 Introduction

Tensor-variate regression approaches arise frequently in the machine learning community because of the increasing demands to deal with the problems that usually involve large-scale higher-order data with complex structures. Such problems abound in a broad range of applications, including computer vision [Zhao et al., 2014], neural signal processing [Zhao et al., 2011; 2013; Eliseyev and Aksenova, 2013], medical imaging data analysis [Zhou et al., 2013], climate data analysis [Bahadori et al., 2014], and are in general very challenging to solve. One of these approaches in widespread use is N-way partial least squares (NPLS) [Bro, 1996] which carries out a joint CP (CANDECOMP/PARAFAC) decomposition [Harshman, 1970] of tensorial input and output into sum of rank-one tensors. To overcome the low predictability and the slow convergence rate of NPLS, the state-of-the-art higher-order partial least squares (HOPLS) [Zhao et al., 2011; 2013] has been proposed based on the joint orthogonal block Tucker decomposition [De Lathauwer et al., 2000a] of input and output tensors. The most preferable property of HOPLS over NPLS lies in its power to provide superior predictability with optimal balance between fitness and model complexity [Zhao et al., 2013].

For many real-world applications, however, high-order big size tensorial data often take the form of extremely large or even infinite tensor sequences, especially in time-critical dynamic environments, where the new data keep coming fast over time. Although we can apply the batch method to all the data each time a new pair arrives, in this case HOPLS can quickly become computationally prohibitive or merely infeasible. Moreover, it is often impossible to store the whole data or require them to be available up front. Thus, the memory efficient sequential methods are absolutely essential.

Among the methods that can recursively handle the data, the recursive N-way partial least squares (RNPLS) [Eliseyev and Aksenova, 2013] processes the tensor sequences by unifying a recursive calculation scheme with the multivway data representation of NPLS. Inheriting the drawbacks of NPLS, RNPLS likewise suffers from the lack of adequate accuracy and the slow convergence rate because of its CP-based solution. Thus, the speed is rather slow especially when a relatively larger number of latent vectors are required for sufficient accuracy, significantly reducing the applicability of RNPLS in many time-critical applications [Zhao et al., 2013]. Another recent work named accelerated low-rank tensor online learning (ALTO) [Yu et al., 2015] has been proposed for the speed of processing spatio-temporal tensor sequence using a random low-rank projection technique. However, ALTO is only restricted to spatio-temporal data structure which demands the input and output tensors are at least in common on structures along the spatial and temporal modes.

In this paper, we introduce a super-fast tensor regression framework, called recursive higher-order partial least squares (RHOPLS), for sequential blockwise processing of general tensor sequences. Our contributions are: (i) designing a recursive framework that efficiently updates the regression coefficients (factors) at a small-scale factor (feature) level instead of the large raw data (observation) level by integrating a low-rank modification strategy of the Tucker [O’Hara, 2010] into PLS; (ii) developing an efficient algorithm based on a series of computationally advantageous calculations yet with only a small number of factors for storage; (iii) applying RHOPLS to several challenging tasks such as estimation of human pose from videos, showing the great potentiality
for fast real-time predictions of human pose positions. In
brief, our framework does not suffer from neither inferior pre-
dictability nor poor convergence rate of NPLS-based models.
It is also free from the computational issues related to the
HOPLS-based models when the data order, “sample” or “di-
imensionality” complexity is high. Finally, our RHOPLS ex-
hibits highly competitive accuracy with the best batch meth-
ods but is extremely faster than other sequential methods.

2 Recursive Higher-order PLS

2.1 Preliminaries and Problem Setting

Throughout the paper, higher-order tensors will be denot-
ed as \( X \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D} \) in calligraphy, where \( D \) is the order of \( X \). The \( d \)-mode unfolding of \( X \), defined as \( X_{(d)} \in \mathbb{R}^{I_1 \times I_d \cdots I_D} \), is the process of rearranging the \( d \)-mode vectors into the columns of the resulting matrix.

We refer to \( d \)-mode product of tensor \( X \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D} \) and matrix \( A \in \mathbb{R}^{I_d \times I_d} \) as \( Y = X \times_d A \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D \times I_d} \). We used contracted product of tensors \( X \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d \times J_1 \times \cdots \times J_P} \) and \( Z \in \mathbb{R}^{I_1 \times \cdots \times I_d \times K_1 \times \cdots \times K_Q} \) along the same \( C \) modes to define the cross-covariance tensor [Zhao et al., 2013] \( \mathcal{Y} \) as

\[
\mathcal{Y} = <X, Z> \mid \{I_1, \ldots, C, \ldots, C\} = \sum_{i_1=1}^{I_1} \cdots \sum_{i_C=1}^{I_C} x_{i_1, \ldots, i_C, j_1, \ldots, j_P} z_{i_1, \ldots, i_C, k_1, \ldots, k_Q} \in \mathbb{R}^{I_1 \times \cdots \times J_P \times K_1 \times \cdots \times K_Q}.
\]

The Tucker decomposition can be expressed as \( \mathcal{Y} \approx \mathcal{G} \times_1 A^{(1)} \times_2 \cdots \times_D A^{(D)} \), where \( \mathcal{G} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D} \) is the factor matrix. \( \mathcal{G} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D} \) represents the core tensor indicating the interaction among factor matrices in different modes. \( R_d \) serves as the \( d \)-rank of \( \mathcal{Y} \) in mode \( d \).

Without the loss of generality, we consider an \( N \)-th-order input tensor \( X \in \mathbb{R}^{I_1 \times \cdots \times I_N} \) and an \( M \)-th-order output tensor \( \mathcal{Y} \in \mathbb{R}^{I_1 \times \cdots \times I_M} \) with coupled observations, namely \( I_1 = 1 \) and we process the data in terms of mini-batch of size \( b \), say \( \{X'(t) : t \in \mathbb{N}^{b} \} \times \{Y'(t) : t \in \mathbb{N}^{b} \} \) for mini-
batch \( t \). \( I_0 \) will stand for the number of initial observations. We use \( I \) to denote the maximum index of all modes, while \( R \) is the maximum \( d \)-rank of all modes which are supposed to be small in practice, i.e., \( R \ll I \). Finally, \( F \) is the number of latent vectors, while \( L \) and \( K \) are used to denote the collection of \( d \)-ranks for the input and output loadings, respectively.

2.2 General Framework

As mentioned in Section 1, sequential approaches to regres-
sion are capable of handling big data as well as online data.
In this context, the key idea behind RNPLS [Eliseyev and
Aksenova, 2013] is to directly add the old tensor data, rep-
resented by refolding the previous set of factors into a fixed data size, to the new arriving tensor data to generate the joint tensor data, and then applies NPLS to joint tensor data to es-
timate the new set of factors. Although recursively handling
the new data, RNPLS has one major disadvantage in that it
has to perform a slow iterative NIPALS-style procedure on
the joint tensor data at the level of original dimensionality \( I \).
Furthermore, RNPLS represents the old tensor via a refolding
operation of the factor matrices into tensor, which may cause undesired errors of approximating the old tensor data.

Different from RNPLS, we propose to recursively merge
the new data into the old one in terms of the factors rather
than considering the raw tensor data. Our framework directly
updates the PLS-related parameters at much smaller scale
using a closed-form solution, thus inexpensive keeping track
of the subspace patterns and the summary of model status.

Generally speaking, up to the point \( t - 1 \), the set of old
factors representing the model summarizes the total variation
in all previous mini-batches, and hence approximates the cur-
rent modeling state. Then, a newly arriving mini-batch \( t \) is
decomposed into a set of factors which contain the variation
merely corresponding to the new data. Our framework
carries out a recursive update procedure that effectively
yet inexpensively “absorbs” the incremental factors into the
old ones to produce a set of new factors at \( t \). Continuing
in the same spirit, we are able to efficiently process the new data
in a mini-batch way over time. Figure 1 illustrates the whole
scheme that consists of the following principal steps as below.
We choose \( N = 3 \) order input tensor and \( M = 2 \) order output
tensor for simplicity of visualization.

In what follows, Step 0 is a preprocessing step that ex-
cutes only once, while the other steps are conducted repeat-
edly for each new mini-batch. With the extracted loadings
and individual core tensors in hand, the PLS-based regression
coefficients can be formed in terms of these extracted factors
and the prediction can be made similar to the one in HOPLS
[Zhao et al., 2013].

Step 0: Initial Approximation (red arrow)

As for the initial tensor pair \( \{X'(0) \in \mathbb{R}^{I_1 \times \cdots \times I_N}, Y'(0) \in \mathbb{R}^{I_1 \times \cdots \times I_M} \} \), we aim to extract a set of initial factors. Instead of using a deflation process of block tensor terms in HOPLS [Zhao et al., 2011; 2013], we simply apply the standard Tucker to jointly decompose the initial data pair such that the latent components extracted from \( X'(0) \) and \( Y'(0) \) have the
maximum pairwise covariance, which is of the form

\[
\mathcal{X}'(0) \approx \mathcal{G}^{(0)} \times_1 T(0) \times_2 P^{(2)}(0) \cdots \times_N P^{(N)}(0),
\]

\[
\mathcal{Y}'(0) \approx \mathcal{G}^{(0)} \times_1 T(0) \times_2 Q^{(2)}(0) \cdots \times_M Q^{(M)}(0),
\]

where \( \mathcal{G}^{(0)} \in \mathbb{R}^{I_1 \times \cdots \times I_N} \) and \( \mathcal{G}^{(0)} \in \mathbb{R}^{I_1 \times \cdots \times I_M} \) serve as core tensors with \( L_1 = 1 \) and \( L_2 = 1 \), respectively.

To solve above factors, we first form the 1-mode
cross-covariance tensor \( \mathcal{C}(0) = <X'(0), Y'(0)>_{1:1:1} \) which follows by estimating the loadings using a higher-order orthogonal iteration (HOOI) [De Lathauwer et al., 2000b]

\[
\mathcal{C}(0) \approx \mathcal{G}^{xy}(0) \times_1 Q^{(2)}(0) \times_2 \cdots \times_{M-1} Q^{(M)}(0) \times_M P^{(2)}(0) \times_{M+1} \cdots \times_{M+N-2} P^{(N)}(0).
\]

Here, \( \mathcal{G}^{xy}(0) \in \mathbb{R}^{K_2 \times \cdots \times K_M \times L_2 \times \cdots \times L_N} \) is said to be 1-mode
cross-covariance core tensor. Having obtained all the load-
ings, the core tensors and common latent matrix can be calcu-
lated according to (1) and (2) using higher-order singular vec-
tor decomposition (HOSVD) [De Lathauwer et al., 2000a].
These factors (common latent matrix, loadings, core tensors and the 1-mode cross-covariance core tensor), indicating the initial modeling state, are to be updated on new mini-batch.

**Step 1: Incremental Approximation (yellow arrow)**

Turning to mini-batch $t$, a new tensor pair \( \mathcal{X}(t) \in \mathbb{R}^{b \times J_2 \times \cdots \times J_N} \), \( \mathcal{Y}(t) \in \mathbb{R}^{b \times J_2 \times \cdots \times J_M} \) with small size $b$ comes into the picture. To approximate the incremental data efficiently, we propose to collect the set of factors by first performing partial Tucker to \( \mathcal{X}(t) \) and \( \mathcal{Y}(t) \) separately, such that all the modes except the first mode are decomposed with $d$-ranks $\langle L_2, \ldots, L_N \rangle$ and $\langle K_2, \ldots, K_M \rangle$

\[
\mathcal{X}(t) \approx \hat{G}^x(t) \times_2 \hat{P}^{(2)}(t) \times_3 \cdots \times_N \hat{P}^{(N)}(t),
\]

\[
\mathcal{Y}(t) \approx \hat{G}^y(t) \times_2 \hat{Q}^{(2)}(t) \times_3 \cdots \times_M \hat{Q}^{(M)}(t).
\]

Thereafter, the 1-mode cross-covariance core tensor $\hat{G}^{xy}(t)$ is immediately formed exploiting only the core tensors $\hat{G}^x(t)$ and $\hat{G}^y(t)$ as $\hat{G}^{xy}(t) = \langle \hat{G}^x(t), \hat{G}^y(t) \rangle_{\{2, 1\}}$, where $\hat{G}^{xy}(t) \in \mathbb{R}^{K_2 \times \cdots \times K_M \times L_2 \times \cdots \times L_N}$ can be calculated using only $\mathcal{O}(R^{M+N-2})$. In total, the cost for approximating the incremental data requires merely $\mathcal{O}(\max\{R^{M+N-2}, R^{M-1}, R^{M+N-1}\})$. Note that the HOPLS has to compute and decompose the 1-mode cross-covariance tensor of size $J_2 \times \cdots \times J_M \times J_2 \times \cdots \times J_N$ using entire data when estimating each one of $R$ latent vectors [Zhao et al., 2013], leading to total cost as large as $\mathcal{O}(R^{M+N-1})$. In HOPLS, calculating and decomposing such huge tensor $R$ times yields substantial costs especially when data dimensionality, data order and $d$-ranks are large. In contrast to HOPLS, our Step 1 overcomes this drawback by first partially decomposing only the new mini-batch input and output individually via Tucker and then calculating the desired cross-covariance core tensor to gather the incremental factors. This step contributes to the speed-ups mainly from the perspective of significantly reducing the “dimensionality” complexity, because we extract eigenvectors on two much smaller-scale individual mini-batch tensors instead of on a massive cross-covariance tensor, and we also form the cross-covariance core tensor at the core tensor scale of $R$ instead of at the original dimensionality $I$.

**Step 2: Expansion (blue arrow)**

Inspired by the work [O’Hara, 2010], we first append the set of incremental factors associated with the new mini-batch to the set of old factors that corresponds to the current model status, resulting in an augmented set of factors. Unlike [O’Hara, 2010], we apply this strategy to the cross-covariance core tensor in a PLS framework involving both input-output tensors rather than just single tensor as in [O’Hara, 2010]. Specifically, concatenating the variation captured by loading $\hat{P}^{(n)}(t) \in \mathbb{R}^{l_n \times L_n}$ to $\hat{P}^{(n)}(t-1) \in \mathbb{R}^{l_n \times L_n}$, we can get the augmented variation via loading $\hat{P}^{(n)}(t) = [\hat{P}^{(n)}(t-1) \hat{P}^{(n)}(t)] \in \mathbb{R}^{l_n \times (L_n + L_n)}$, where $n = 2, ..., N$. Likewise, we obtain the loading $\hat{Q}^{(m)}(t) = [\hat{Q}^{(m)}(t-1) \hat{Q}^{(m)}(t)] \in \mathbb{R}^{J_m \times (K_m + K_m)}$ for $m = 2, ..., M$. We can also get the augmented 1-mode cross-covariance core tensor $\hat{G}^{xy}(t) \in \mathbb{R}^{(K_2 + K_2) \times \cdots \times (K_M + K_M) \times (L_2 + L_2) \times \cdots \times (L_N + L_N)}$ by appending $\hat{G}^{xy}(t) \in \mathbb{R}^{K_2 \times \cdots \times K_M \times L_2 \times \cdots \times L_N}$ to $\hat{G}^{xy}(t-1) \in \mathbb{R}^{K_2 \times \cdots \times K_M \times L_2 \times \cdots \times L_N}$ in a super block-diagonal manner, leaving other new entries to be zeros.

**Step 3: Compression (purple arrow)**

We next truncate the set of augmented factors back into the set of factors with the size of original $d$-ranks, which means finding the most dominant principal components in the subspaces of loadings out of the augmented set of loadings. To this end, the augmented factors $\{\hat{P}^{(n)}(t)\}_m^{N_m=2}$, $\{\hat{Q}^{(m)}(t)\}_m^{M_m=2}$ and $\hat{G}^{xy}(t)$ described in last step are truncated to produce...
new loadings \( \{ \mathbf{P}(n)(t) \}_{n=2}^{N} \) as well as 1-mode cross-covariance core tensor \( G^{xy}(t) \in \mathbb{R}^{K_2 \times \cdots \times K_M \times L_2 \times \cdots \times L_N} \), hence keeping track of the change of patterns in each loading subspace, leading to

1. compute QR factorization on the augmented loadings \( \mathbf{F}^{(n)}(t) = \mathbf{U}^{(n)}(t) \mathbf{V}^{(n)}(t), \) where \( \mathbf{U}^{(n)}(t) \in \mathbb{R}^{I_n \times (L_n + L_t)} \) and \( \mathbf{V}^{(n)}(t) \in \mathbb{R}^{(L_n + L_t) \times (L_n + L_t)} \) for \( n = 2, \ldots, N; \)

2. transform cross-covariance core tensor \( \tilde{G}^{xy}(t) \) to get

\[
\tilde{G}^{xy}(t) = \tilde{G}^{xy}(t) \times_1 \mathbf{V}^{(2)}(t) \times_2 \cdots \times_{M-1} \mathbf{V}^{(M)}(t) \times_M \mathbf{V}^{(2)}(t) \times_{M+1} \cdots \times_{M+N-2} \mathbf{V}^{(N)}(t).
\]

3. calculate the rank-\((L_2, \ldots, L_N, K_2, \ldots, K_M)\) orthogonal Tucker on the core \( \tilde{G}^{xy}(t) \) to get the resulting 1-mode cross-covariance core tensor \( \hat{G}^{xy}(t) \) for mini-batch \( t \)

\[
\hat{G}^{xy}(t) \approx G^{xy}(t) \times_1 \mathbf{Z}^{(2)}(t) \times_2 \cdots \times_{M-1} \mathbf{Z}^{(M)}(t) \times_M \mathbf{Z}^{(2)}(t) \times_{M+1} \cdots \times_{M+N-2} \mathbf{Z}^{(N)}(t),
\]

where \( G^{xy}(t) \in \mathbb{R}^{L_2 \times \cdots \times L_N \times K_2 \times \cdots \times K_M}, \mathbf{Z}^{(2)}(t) \in \mathbb{R}^{(L_n + L_t) \times L_n} \), and \( \mathbf{Z}^{(m)}(t) \in \mathbb{R}^{(K_m + K_m) \times K_m} \) for \( m = 2, \ldots, M. \)

4. compute the loadings \( \mathbf{P}(n)(t) = \mathbf{U}^{(n)}(t) \mathbf{Z}^{(x)(t)} \in \mathbb{R}^{I_n \times L_n}; \)

\( \mathbf{Q}^{(m)}(t) = \mathbf{U}^{(m)}(t) \mathbf{Z}^{(y)(t)} \in \mathbb{R}^{I_m \times L_m} \).

The purpose of above operations is to maintain computations at small cross-covariance core tensor level of \( R \) instead of at huge cross-covariance tensor level of \( I \), the dominating cost is thus reduced to \( O(R^{N+M-1}) \) from \( O(I^{N+M-1}). \) In a word, Steps 2 and 3 together are applied to 1-mode cross-covariance core tensor for purpose of updating the loading factors, which contributes in part to the speed-ups from perspective of reducing the "sample" complexity to a low constant level.

**Step 4: Projection (green arrow)**

Finally, we propose to update the individual core tensors of input and output from the internal representation of model under the projection of loadings obtained in the last step. These individual core tensors in conjunction with loadings are exploited to produce the final prediction. Specifically, we begin with reconstruction of the old internal tensor representation \( X_{int}(t-1) \in \mathbb{R}^{I_0 \times L_2 \times \cdots \times L_N} \) from the latent matrix \( T(t-1) \), the loadings \( \{ \mathbf{P}(n)(t-1) \}_{n=2}^{N} \) and the core tensor \( \hat{G}^{xy}(t-1) \) under the projection of current loadings \( \{ \mathbf{P}(n)(t) \}_{n=2}^{N} \) as

\[
X_{int}(t-1) = \hat{G}^{xy}(t-1) \times_1 \mathbf{T}(t-1) \times_2 \mathbf{P}^{(2)}(t)^{T} \mathbf{P}^{(2)}(t-1) \times_3 \cdots \times_N \mathbf{P}^{(N)}(t)^{T} \mathbf{P}^{(N)}(t-1).
\]

Meanwhile, we also reconstruct the incremental internal tensor representation \( \tilde{X}_{int}(t) \) in \( \mathbb{R}^{I_0 \times L_2 \times \cdots \times L_N} \) from the core tensor \( \tilde{G}^{xy}(t) \) and loadings \( \{ \mathbf{P}(n)(t) \}_{n=2}^{N} \) obtained in Step 1 by projecting onto the subspaces of the current loadings \( \{ \mathbf{P}(n)(t) \}_{n=2}^{N} \), which becomes

\[
\tilde{X}_{int}(t) = \hat{G}^{xy}(t) \times_2 \mathbf{P}^{(2)}(t)^{T} \tilde{\mathbf{P}}^{(2)}(t) \times_3 \cdots \times_N \mathbf{P}^{(N)}(t)^{T} \tilde{\mathbf{P}}^{(N)}(t).
\]

After concatenating \( \tilde{X}_{int}(t) \) to \( X_{int}(t-1) \), we get \( X_{int}(t) \in \mathbb{R}^{(I_0 + b) \times L_2 \times \cdots \times L_N} \) as the augmented internal representation. Then, \( X_{int}(t) \) is decomposed using Tucker-1 model [Cichocki et al., 2009] to get the common internal latent matrix \( T_{int}(t) \in \mathbb{R}^{(I_0 + b) \times L_1} \) and core \( \hat{G}^{xy}(t) \in \mathbb{R}^{L_1 \times L_2 \times \cdots \times L_N} \) as

\[
X_{int}(t) \equiv \hat{G}^{xy}(t) \times_1 T_{int}(t),
\]

where \( T_{int}(t) \) is thereafter truncated to keep the very last \( I_0 \) rows, leading to the common \( T(t) \in \mathbb{R}^{I_0 \times L_1} \) for the purpose of internal representation reconstruction of the model in the subsequent mini-batch. Similarly, we finally have \( Y_{int}(t) \in \mathbb{R}^{(I_0 + b) \times K_2 \times \cdots \times K_M} \) and \( \hat{G}^{xy}(t) \in \mathbb{R}^{K_1 \times K_2 \times \cdots \times K_M} \) for the output side. By employing our proposed projection strategy, the dominating cost of this step is substantially cut down from \( O(\max(I_0, I_0^{N+1})) \) to \( O(\max(I_0 R^N, R^{N+1})) \), because we equivalently represent the model and calculate the new factors in terms of projected internal tensors that lie in the tensor space with small scale \( R \) but large original \( I \). Step 4 is responsible for efficiently updating the individual core tensors, which also contributes in part to the total acceleration by reducing the “sample” complexity.

Note that RHOPLS demands a minimum space complexity in the sense that only a small number of factors, dominating by \( O(\max(R^{N+M-2})) \) of cross-covariance core tensor, are needed to be stored to represent the running model.

### 2.3 Related Work

In contrast to the standard (batch) Tucker, incremental tensor analysis (ITA) [Sun et al., 2008] has been proposed as an incremental tensor approximation tool that dynamically updates a Tucker approximation using new arriving data. However, ITA is a component model that is restricted to unsupervised learning settings only, focusing only on a single tensor rather than input-output tensor pairs in supervised regression tasks. As another incremental tensor approximation tool, incremental Tucker (IT) [O‘Hara, 2010] additively updates the Tucker model using a low-rank subspace truncation strategy. Like ITA, IT is a component model that considers single tensor in the context of unsupervised learning only and cannot be directly applied to the tensor regression tasks. Our Step 2 and 3 are adapted from the basic idea of this tool, however, we here propose to integrate IT as just one part of our whole PLS-based framework to extract loadings. Notice that the majority contribution of our RHOPLS is from the system level, all the steps are essential to the whole framework and should not be treated separately. Moreover, we like to stress that the concepts of tensor decomposition tool, i.e., Tucker, ITA or IT, are fundamentally different from tensor regression models w.r.t. the targeted goal, algorithm and applications.

### 3 Experimental Results

In our experiments, the root mean squares of prediction (RMSEP) [Kim et al., 2005] as well as the \( Q \) index [Luo et al., 2015] are used to quantitatively gauge the predictive performance of our approach. We recorded the CPU learning time per new mini-batch for all recursive methods, and we also gave CPU learning time for batch methods using the entire training set. We compared RHOPLS with NPLS [Bro, 1996],
Table 1: Performance comparison on UMPM with frame size 24 × 32 for \( I_0 = 200 \) and \( b = 2 \).

<table>
<thead>
<tr>
<th>Methods</th>
<th>( Q )</th>
<th>CPU−Time (s)</th>
<th>( Q )</th>
<th>CPU−Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPLS ( F = 2 ) (triangle)</td>
<td>32.0</td>
<td>8.431 (0.070)</td>
<td>32.0</td>
<td>8.431 (0.070)</td>
</tr>
<tr>
<td>RNPLS ( F = 2 ) (triangle)</td>
<td>32.0</td>
<td>9.799 (0.003)</td>
<td>32.0</td>
<td>9.799 (0.003)</td>
</tr>
<tr>
<td>HOPLS ( F = 40, L = [8, 12], K = [3, 9] )</td>
<td>32.0</td>
<td>127.8 (2.2)</td>
<td>32.0</td>
<td>127.8 (2.2)</td>
</tr>
<tr>
<td>IHOPLS ( F = 40, L = [12, 16], K = [3, 9] )</td>
<td>32.0</td>
<td>129.0 (2.2)</td>
<td>32.0</td>
<td>129.0 (2.2)</td>
</tr>
<tr>
<td>RHOPLS ( F = 40, L = [12, 16], K = [3, 9] )</td>
<td>32.0</td>
<td>132.3 (3.0)</td>
<td>32.0</td>
<td>132.3 (3.0)</td>
</tr>
</tbody>
</table>

Table 2: Performance on UMPM with same hyper-parameters as Table 1 for all methods for frame size 240 × 320.

<table>
<thead>
<tr>
<th>Methods</th>
<th>( Q )</th>
<th>Time (s)</th>
<th>( Q )</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPLS</td>
<td>8.056</td>
<td>340.8</td>
<td>8.056</td>
<td>340.8</td>
</tr>
<tr>
<td>RNPLS</td>
<td>8.105</td>
<td>39.9</td>
<td>8.105</td>
<td>39.9</td>
</tr>
<tr>
<td>HOPLS</td>
<td>0.7774</td>
<td>53.0</td>
<td>0.7774</td>
<td>53.0</td>
</tr>
<tr>
<td>IHOPLS</td>
<td>0.7125</td>
<td>50.1</td>
<td>0.7125</td>
<td>50.1</td>
</tr>
<tr>
<td>RHOPLS</td>
<td>0.8224</td>
<td>0.26</td>
<td>0.8224</td>
<td>0.26</td>
</tr>
</tbody>
</table>

Table 3: Performance versus the number of latent vectors and initial loadings for the ‘triangle’ scenario.

<table>
<thead>
<tr>
<th>Methods</th>
<th>( T )</th>
<th>( \theta )</th>
<th>( T )</th>
<th>( \theta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPLS</td>
<td>0.78</td>
<td>120</td>
<td>0.78</td>
<td>120</td>
</tr>
<tr>
<td>RNPLS</td>
<td>0.78</td>
<td>120</td>
<td>0.78</td>
<td>120</td>
</tr>
<tr>
<td>HOPLS</td>
<td>0.78</td>
<td>120</td>
<td>0.78</td>
<td>120</td>
</tr>
<tr>
<td>RHOPLS</td>
<td>0.78</td>
<td>120</td>
<td>0.78</td>
<td>120</td>
</tr>
</tbody>
</table>

Table 4: Performance versus the number of initial samples for the ‘triangle’ scenario.

<table>
<thead>
<tr>
<th>Methods</th>
<th>( T )</th>
<th>( \theta )</th>
<th>( T )</th>
<th>( \theta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPLS</td>
<td>0.78</td>
<td>120</td>
<td>0.78</td>
<td>120</td>
</tr>
<tr>
<td>RNPLS</td>
<td>0.78</td>
<td>120</td>
<td>0.78</td>
<td>120</td>
</tr>
<tr>
<td>HOPLS</td>
<td>0.78</td>
<td>120</td>
<td>0.78</td>
<td>120</td>
</tr>
<tr>
<td>RHOPLS</td>
<td>0.78</td>
<td>120</td>
<td>0.78</td>
<td>120</td>
</tr>
</tbody>
</table>

Figure 2: For ‘triangle’ and ‘table’ scenarios, learning error and learning time versus the iteration size 24 × 32.

Figure 3: Performance versus the number of latent vectors and initial samples sizes with frame size 24 × 32.

100 times, the acceleration rates are even higher with the larger numbers of latent vectors. Figure 2 shows that, for both scenarios, the predictive error of RHOPLS keeps decreasing at a faster rate as the iteration goes on, while the CPU cost remains as a low constant trend over time. We can also observe that, as illustrated in Figure 3, the prediction accuracy of RHOPLS stays very close to that of HOPLS as the number of initial samples decreases.

3.1 Utrecht Multi-Person Motion Database

We first tested RHOPLS on the Utrecht Multi-Person Motion (UMPM) benchmark [Van Der Aa et al., 2011], which provides the simultaneous recordings of video sequences and 3D ground truth positions of human natural motions in daily life activities. For our test, the input data was an intensity image sequence from the front camera at 25fps, taking the form of a 3rd-order predictor tensor (i.e., frames × width × height). On the other hand, the corresponding output containing 3D positions of 37 reflective markers can be represented as a 3rd-order tensor (i.e., samples × 3D positions × marker). The averaged predictive performance as well as learning time are compared in Table 1. As we can see, RHOPLS achieves highly comparable accuracy with the batch HOPLS but is extremely faster. Spectacularly, the speed-ups of RHOPLS over RNPLS and NPLS for “table” scenario are more than 30 and 100 times, the acceleration rates are even higher with the larger numbers of latent vectors. Figure 2 shows that, for both scenarios, the predictive error of RHOPLS keeps decreasing at a faster rate as the iteration goes on, while the CPU cost remains as a low constant trend over time. We can also observe that, as illustrated in Figure 3, the prediction accuracy of RHOPLS stays very close to that of HOPLS as the number of initial samples decreases.
of latent vectors ranges from 10 to 90 for different loadings. However, the CPU time just slightly increases from 0.19s to 0.38s for RHOPLS with input loading [8, 12], which is in contrast to that of HOPLS from 1.79s to 19.22s, indicating the superior scalability of the RHOPLS with increasing number of latent vectors (d-ranks). The Q is given in Figure 3 with varying numbers of initial samples and mini-batch sizes. We may notice that only 50 initial samples, which is 8% of the whole training data, will suffice to guarantee a reasonably good result, i.e., nearly 0.78 for mini-batch size of 2.

![Figure 4: Time versus input loading L with frame size 240 × 320. Right plot zooms in on the lower region of the left plot.](image)

To see how all the methods behave in large “dimensionality” context, we fix the same hyper-parameters as in Table 1 but use frame size 240 × 320 instead of 24 × 32. Again, in Table 2, RHOPLS is quite close to HOPLS in accuracy but enormously enlarges the speed gaps against RNPLS and NPLS by more than 200 and 3000 times in “table” scenario, respectively. Figure 4 demonstrates the scalability of RHOPLS with increasing input loading size L. The right plot zooming in on the bottom of left plot shows that RHOPLS is much less sensitive to the growing loading size compared to HOPLS. This reflects the fact that the impact of large d-ranks on computational load when using HOPLS has been significantly diminished by using our feature-based updating strategy.

### 3.2 Neurotycho Electrocorticography Dataset

In this section, the tests were carried out on a benchmark tensor regression application, that is, decoding limb movements from monkey’s brain signals using Neurotycho food tracking Electrocorticography (ECoG) dataset [Chao et al., 2010]. ECoG data contains a 15 minute-long recording and we downsampled motion data to different frequencies, producing various lengths of observations with different levels of overlapped features. As for the input, the wavelet transformed ECoG signal was represented as a 4th-order tensor (i.e., samples × time × frequency × channel). A 3rd-order tensor of 3D movement distances of the monkey’s limb on 4 markers was used as the output. In Table 3, RHOPLS again performs consistently better than NPLS, RNPLS, IHOPLS for the 4th-order’s input situation. In Figure 5, RHOPLS maintains nearly the same predictability with HOPLS in the settings of both low frequency (difficult case) and high frequency (easy case). In the meantime, RHOPLS consumes low CPU time regardless of the length of sequence, and exhibits high speed-up rates over RNPLS, i.e., 24 times faster at 4500, and overall nearly 10 times faster than IHOPLS, while NPLS and HOPLS are almost useless in fast time-critical applications. Note that the recursiveness property of RHOPLS implies that any long (even infinite) low-rank sequence can be handled by RHOPLS with a short constant processing time, thus we will get similar result for sequence with even larger “sample” complexity. Figure 6 visualizes an example of the observed and the predicted trajectories of monkey’s hand.

![Figure 5: Performance versus sequence length or frequency.](image)

![Figure 6: An example of ground truth (150s time window) and the trajectories predicted by RHOPLS, HOPLS and RNPLS for Z-coordinate of the monkey’s hand.](image)

### 4 Discussion

The drastic accelerations of our RHOPLS are realized in two stages, the first, due to Step 1, focuses on the reduction of “dimensionality” complexity. On the basis of the first stage, the second stage (i.e., Steps 2, 3 and 4) concentrates on making the low constant “sample” complexity possible. The overall speed-ups stem from directly updating the set of factors (regression coefficients) in lightweight manner at a small-scale factor (feature) level instead of the raw data level, such that the relatively expensive eigenvector-style calculations are able to execute a lot faster on the factor scale.
References


