HHS Public Access

Author manuscript

Adv Neural Inf Process Syst. Author manuscript; available in PMC 2023 November 22.

Published in final edited form as:

Adv Neural Inf Process Syst. 2022 ; 35: 32039–32052.

ATD: Augmenting CP Tensor Decomposition by Self Supervision

Chaoqi Yang[1], **Cheng Qian**[2], **Navjot Singh**[1], **Cao Xiao**[3], **M Brandon Westover**[4,5], **Edgar Solomonik**[1], **Jimeng Sun**[1]

[1]University of Illinois Urbana-Champaign

[2]IQVIA

[3]Relativity

[4]Massachusetts General Hospital

[5]Harvard Medical School

Abstract

Tensor decompositions are powerful tools for dimensionality reduction and feature interpretation of multidimensional data such as signals. Existing tensor decomposition objectives (e.g., Frobenius norm) are designed for fitting raw data under statistical assumptions, which may not align with downstream classification tasks. In practice, raw input tensor can contain irrelevant information while data augmentation techniques may be used to smooth out class-irrelevant noise in samples. This paper addresses the above challenges by proposing augmented tensor decomposition (`ATD`), which effectively incorporates data augmentations and self-supervised learning (SSL) to boost downstream classification. To address the non-convexity of the new augmented objective, we develop an iterative method that enables the optimization to follow an alternating least squares (ALS) fashion. We evaluate our proposed `ATD` on multiple datasets. It can achieve 0.8% ~ 2.5% accuracy gain over tensor-based baselines. Also, our `ATD` model shows comparable or better performance (e.g., up to 15% in accuracy) over self-supervised and autoencoder baselines while using less than 5% of learnable parameters of these baseline models.

1 Introduction

Extracting unsupervised features from high-dimensional data is essential in various scenarios, such as physiological signals (Cong et al., 2015), hyperspectral images (Wang et al., 2017) and fMRI (Hamdi et al., 2018). Tensor decomposition models are often used for high-order feature extraction (Sidiropoulos et al., 2017), among these, CANDECOMP/PARAFAC (CP) decomposition is one of the most popular models. The low-rank CP tensor decompositions (Kolda and Bader, 2009) assume that the input data is composited by a small set of components, while the reduced features are the coefficients that quantify the importance of each basis, which provides a compact representation.

chaoqiy2@illinois.edu .

Under this low-rank assumption, existing tensor decomposition objectives aim to *fit individual data samples* with statistical error measures (Hong et al., 2020; Singh et al., 2021), e.g., Frobenius norm or KL-divergence. Though *fitness* is an essential principle for feature reduction, common objective functions do not account for downstream tasks, e.g., classification.

Contrastive self-supervised learning (SSL) (He et al., 2020) is recently popular for unsupervised feature learning, which utilizes the class-preserving data augmentations (Dao et al., 2019) and learns an encoder that can filter out class-irrelevant information. The optimization goal is to *enforce alignments* (Chen et al., 2020; Wang and Isola, 2020), ensuring that the anchor sample is closer to the positive sample (which has the same latent class as the anchor sample) than to the negative sample (which is in a different latent class) in the embedding space. In an unsupervised setting, positive samples are given by data augmentations, while negative samples are hard to acquire (Arora et al., 2019; He et al., 2020; Chen et al., 2020). Also, previous models are mostly built on deep neural networks, which are often black-box models with tens of thousands of learnable parameters.

This paper aims to incorporate both the *fitness* and *alignment* principles into CP tensor decomposition[1] by augmenting the common fitness objective with a new self-supervised loss. The new self-supervised loss is based on the unbiased estimation of negative samples Chuang et al. (2020), which effectively prevents the sampling bias issue (Arora et al., 2019). The purpose of our design (i.e., introducing SSL into CP tensor decomposition) is to learn class-aware tensor decomposition results for boosting the downstream tensor sample classifications. To address the non-convex subproblems from the new objective, we formulate an iterative method, which solves least squares optimization in each step with closed-form solution. Main contributions are summarized below.

- We propose **augmented tensor decomposition**, named `ATD`, which learns an unsupervised CP structure decomposition by extending the original fitness objective with a self-supervised loss on the contrastiveness of similar and dissimilar tensor samples.
- We develop an iterative method to address the non-convex subproblem from the new objective, which enables our algorithm to **follow an alternative least squares fashion**. Our algorithm has *asymptotically the same complexity* of each optimization sweep as the common CP-ALS.
- We provide **extensive evaluations on four real-world datasets** and compare to recent tensor decomposition models, autoencoder models, and self-supervised models. Our method shows better or comparable prediction performance in various downstream classifications while only requiring much fewer (e.g., less than 5% of) parameters than that of deep learning baselines.

[1]Our design may work for other tensor models, such as Tucker decomposition. We leave it to future work.

2 Background

Notation.

We use plain letters for scalars, such as x or X, boldface uppercase letters for matrices, e.g., $\mathbf{X}$, boldface lowercase letters for vectors, e.g., $\mathbf{x}$, and Euler script letters for tensors, random variables of tensors, and probability distributions, e.g., $\mathscr{X}$. Tensors are multidimensional arrays indexed by three or more indices (modes). For example, an N-mode tensor $\mathscr{X}$ is an N-dimensional array of size $I_1 \times \cdots \times I_N$, where $x_{i_1,\ldots,i_N}$ is the element at the $(i_1, \cdots, i_N)$-th position. For matrix $\mathbf{X}$, the r-th row and column are $\mathbf{x}^{(r)}$ and $\mathbf{x}_r$ respectively, while x_{ij} is for the (i, j)-th element. $\|\mathbf{X}\|_F$ is the Frobenius norm. For vector $\mathbf{x}$, the r-th element is x_r, and we use $\|\mathbf{x}\|_2$ to denote the vector 2-norm, $\langle \cdot, \cdot \rangle$ for the vector inner product, $\circ$ for the outer product, and $[\![\cdot]\!]$ for the Kruskal product. Indices in the paper start from 1, e.g., $\mathbf{x}_1$ is the first column of the matrix.

2.1. Tensor Modeling

This paper aims to learn tensor bases from unlabeled data and then use the bases to build a feature extractor for downstream classification. Without loss of generality (w.r.t. tensor order), we consider the fourth-order tensor, e.g., a collection of multi-channel Electroencephalography (EEG) signals,

$$\mathscr{T} = \left[\mathscr{T}^{(1)}, \mathscr{T}^{(2)}, \ldots, \mathscr{T}^{(N)}\right] \in \mathbb{R}^{N \times I \times J \times K},$$

where $\mathscr{T}^{(n)} \in \mathbb{R}^{I \times J \times K}$. The first dimension of $\mathscr{T}$ corresponds to data samples (e.g., one for each patient), while the other three are feature dimensions (e.g., *channel by frequency by timestamp*).

Data Model.

To model the above tensor, previous works (Kolda and Bader, 2009) assume that

- There are a set of rank-one tensor components $\mathscr{C} = \{\mathscr{C}_1, \ldots, \mathscr{C}_R\}$, which are learnable;
- The tensor data sample/slice $\mathscr{T}^{(n)}$ admits a low-rank structure and can be represented as a weighted sum of these tensor components $\mathscr{C}$, where $\mathbf{x}^{(n)}$ denotes its coefficient vector;
- On top of the low-rank structure, each data sample $\mathscr{T}^{(n)}$ also contains additional element-wise i.i.d. Gaussian noise due to real-world distortion (e.g., physical noise in signal measurements).

In the context of downstream classifications, we further assume that each sample $\mathscr{T}^{(n)}$ is semantically associated to one of the latent classes $p \in \{1, \ldots, P\}$, and we let $\mathscr{D}_p$ be the sample distribution of class-p. Thus, the data sample can be formulated as, $\forall n$,

$$\mathcal{T}^{(n)} = \sum_{r=1}^{R} x_r^{(n)} \cdot \mathcal{E}_r + \epsilon^{(n)} \sim \mathcal{D}_p, \quad p \in \{1, \ldots, P\}, \tag{1}$$

where

$$\mathcal{E}_r = \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r, \quad r \in \{1, \ldots, R\},$$
$$\epsilon^n \sim_{\text{i.i.d.}} \mathcal{N}(0, \sigma), \quad where\ \sigma\ is\ generally\ small\ .$$

Here $\mathbf{a}_r, \mathbf{b}_r, \mathbf{c}_r$ are the learnable parameters, which produces the rank-one component $\mathcal{E}_r$, and they are the column vectors of $\{\mathbf{A} \in \mathbb{R}^{I \times R}, \mathbf{B} \in \mathbb{R}^{J \times R}, \mathbf{C} \in \mathbb{R}^{K \times R}\}$, referred as "bases".

CANDECOMP/PARAFAC Decomposition (CPD).

Given the above tensor model, standard CPD only captures the i.i.d. Gaussian noise by minimizing the negative log-likelihood (NLL), which results in the following standard fitness/reconstruction loss,

$$\mathcal{L}_{cpd} = \sum_{n=1}^{N} \left\| \mathcal{T}^{(n)} - [\mathbf{x}^{(n)}, \mathbf{A}, \mathbf{B}, \mathbf{C}] \right\|_F^2 = \| \mathcal{T} - [\mathbf{X}, \mathbf{A}, \mathbf{B}, \mathbf{C}] \|_F^2 .$$

Here, the Kruskal product $[\cdot]$ outputs a fourth-order reconstructed tensor from four input factor matrices. For consistency, if the first input is a vector, the output is considered as a third-order tensor.

2.2 Problem Formulation

CP decomposition seeks a low-rank reconstruction, without special consideration for the downstream task. In this paper, we are motivated to improve the CPD model by exploiting the latent classes (in an unsupervised way) and learn good bases to provide better class-aware features for classification.

What are Good Bases?

This paper considers two design principles for good bases. The first principle is *fitness*, which requires a low-rank tensor reconstruction with the bases. Second, data samples associated with the same latent class should be decomposed into similar coefficient vectors, with the bases, while the vectors should be dissimilar if the samples are from different latent classes. This principle is called *alignment*, which is important for classification but not considered in the standard tensor decomposition. In this paper, we assess the quality of the learned bases by the performance of downstream classification, where the coefficient vectors (obtained using the bases via decomposition) are the feature inputs (into the downstream classifier).

Working Pipelines.

To put it succinctly, the paper tackles an unsupervised learning problem while using downstream supervised classification for evaluation. The procedures are briefly outlined:

- First, we **learn** the bases $\{\mathbf{A}, \mathbf{B}, \mathbf{C}\}$ from a large set of unlabeled data. The loss function is developed in consideration of the *fitness* and *alignment* (defined in the next section) principles.
- Then, we **construct** the following feature extractor given $\{\mathbf{A}, \mathbf{B}, \mathbf{C}\}$. The feature vector of a new sample is obtained by the closed-form solution of the least squares problem ($\alpha > 0$ is a hyperparameter),

$$\mathbf{f}(\mathcal{T}^{(new)}; \mathbf{A}, \mathbf{B}, \mathbf{C}) = \underset{\mathbf{x} \in \mathbb{R}^{1 \times R}}{\arg\min} \left(\left\| \mathcal{T}^{(new)} - [\mathbf{x}, \mathbf{A}, \mathbf{B}, \mathbf{C}] \right\|_F^2 + \alpha \|\mathbf{x}\|_2^2 \right). \tag{2}$$

  Note that, when $\mathbf{f}(\cdot)$ is applied to a batch of samples, it outputs a coefficient matrix.
- Next, we **evaluate** the feature extractor with a set of labeled data. Given $\mathbf{f}(\cdot)$, we first apply it on the labeled data to extract their features and then train an additional logistic regression model (as the downstream classifier) on top of the extracted features, so that the result of classifications will implicitly reflect how good the bases are.

3 Augmented Tensor Decomposition

We show our model in Figure 1. The design is inspired by the recent popularity of SSL. To exploit the latent class information, we introduce class-preserving data augmentation into CPD model and design self-supervised loss to constrain the learned low-rank features (i.e., the coefficient vectors).

Data Augmentation[2].

In general, data augmentation methods are chosen to perturb the raw data while preserving class label information. Given a sample $\mathcal{T}^{(n)}$, we assume that after data augmentation, its perturbation $\widetilde{\mathcal{T}}^{(n)}$ preserves the label and admits the component-based representation as in Eqn. (1).

3.1 Self-supervised Loss

The design of our self supervised loss corresponds to the *alignment* principle, which is based on pairwise feature similarity and dissimilarity. We call a pair of data samples from the same latent class as *positive pair*, a pair of samples from different latent classes as *negative pair* and a pair of independent samples (two random samples from the dataset) as *random pair*. Intuitively, an anchor plus a positive sample composes a positive pair, similarly for negative pairs and random pairs. In this work, our self-supervised loss aims to maximize the feature similarity between positive pairs and minimizes that between negative pairs *in an unsupervised way* (no labels during optimization).

[2]We provide further discussions and ablation studies on data augmentation in appendix D.3 and D.5

Formally, let $\mathcal{X}_p$, $\mathcal{Y}_p$ be discrete random variables (of tensor samples) distributed as $\mathcal{D}_p$, $p \in \{1, \ldots, P\}$. We want to minimize the following objective when *no class labels are given*,

$$\mathcal{L}_{ss} = \mathcal{L}_{pos} + \lambda \mathcal{L}_{neg},$$

where $\lambda \geq 1$ is a hyperparameter and

$$\begin{aligned} \mathcal{L}_{pos} &= -\mathbb{E}\left[\mathrm{sim}\left(\mathbf{f}\left(\mathcal{X}_p\right), \mathbf{f}\left(\mathcal{Y}_q\right)\right) \mid p = q\right], \\ \mathcal{L}_{neg} &= \mathbb{E}\left[\mathrm{sim}\left(\mathbf{f}\left(\mathcal{X}_p\right), \mathbf{f}\left(\mathcal{Y}_q\right)\right) \mid p \neq q\right]. \end{aligned} \tag{3}$$

Here $\mathcal{L}_{pos}$ maximizes the feature similarity of positive pairs while $\mathcal{L}_{neg}$ minimizes the feature similarity of negative pairs. $\mathbf{f}(\cdot)$ is the feature extractor, defined in Eqn. (2), and the similarity measure is given by cosine distance, parameterized by two random variables,

$$\mathrm{sim}\left(\mathbf{f}\left(\mathcal{X}_p\right), \mathbf{f}\left(\mathcal{Y}_q\right)\right) = \left\langle \frac{\mathbf{f}(\mathcal{X}_p)}{\|\mathbf{f}\left(\mathcal{X}_p\right)\|_2}, \frac{\mathbf{f}(\mathcal{Y}_q)}{\|\mathbf{f}\left(\mathcal{Y}_q\right)\|_2} \right\rangle.$$

To this end, the key is to implement the self-supervised loss, i.e., Eqn. (3), in an unsupervised setting. Specifically, we want to construct the sampler of positive pairs and the sampler of negative pairs with unlabeled data only. The sampler of positive pairs (in short, positive sampler) can be easily approximated by data augmentation techniques, which provides "surrogate" positive pairs (given any sample as the anchor, we apply data augmentation methods to generate a perturbed data as positive sample, and then the anchor plus the perturbed data is a positive pair). However, the negative sampler is infeasible, without labels. As a compromise, previous works He et al. (2020); Chen et al. (2020) consider using random sampler to replace the negative sampler given that the random sampler can be easily achieved by picking two independent samples from the dataset. However, this practice is shown to induce sampling bias and hurts the learned representation Arora et al. (2019); Chuang et al. (2020) since a random pair may be from the same latent class.

Construction of Negative Sampler.

In this paper, we consider using the *law of total probability* to construct the negative sampler in a statistical way. Formally, assume r_p is the label rate of latent class-p (thus, we have $\sum_p r_p = 1$), we apply the law of total probability and the following holds,

$$\begin{aligned}
&\mathbb{E}\left[\operatorname{sim}\left(\mathbf{f}\left(\mathcal{X}_p\right), \mathbf{f}\left(\mathcal{Y}_q\right)\right) \mid p \neq q\right] = \sum_{p=1}^{P} r_p \sum_{q \neq p} \frac{r_q}{1-r_p} \mathbb{E}_{p,q}\left[\operatorname{sim}\left(\mathbf{f}\left(\mathcal{X}_p\right), \mathbf{f}\left(\mathcal{Y}_q\right)\right)\right] \\
&= -\sum_{p=1}^{P} \frac{r_p r_p}{1-r_p} \mathbb{E}_{p,q}\left[\operatorname{sim}\left(\mathbf{f}\left(\mathcal{X}_p\right), \mathbf{f}\left(\mathcal{Y}_q\right)\right) \mid p = q\right] + \sum_{p=1}^{P} \sum_{q=1}^{P} \frac{r_p r_q}{1-r_p} \mathbb{E}_{p,q}\left[\operatorname{sim}\left(\mathbf{f}\left(\mathcal{X}_p\right)\right.\right. \\
&\left.\left., \mathbf{f}\left(\mathcal{Y}_q\right)\right)\right] \\
&= -\mathbb{E}\left[\frac{r_p}{1-r_p} \operatorname{sim}\left(\mathbf{f}\left(\mathcal{X}_p\right), \mathbf{f}\left(\mathcal{Y}_q\right)\right) \mid p = q\right] + \mathbb{E}\left[\frac{1}{1-r_p} \operatorname{sim}\left(\mathbf{f}\left(\mathcal{X}_q\right), \mathbf{f}\left(\mathcal{Y}_q\right)\right)\right]
\end{aligned} \tag{4}$$

Here, the usage of $\mathbb{E}[\cdot]$ means that the expectation is taken over four interdependent random variables, i.e., $p, q, \mathcal{X}_p, \mathcal{Y}_q$, while $\mathbb{E}_{p,q}[\cdot]$ means that p, q is fixed and thus it is only taken over two random variables, i.e., $\mathcal{X}_p, \mathcal{Y}_q$. The result shows that the negative sampler can be equivalently replaced by a weighted combination of the random sampler and positive sampler. Here we do not have access to $r_p, \forall p$ with unlabeled data, this issue is dealt with later.

Self-supervised Loss.

Consequently, we can reformulate our self-supervised loss as,

$$\begin{aligned}
\mathcal{L}_{ss} &= \mathcal{L}_{pos} + \lambda \mathcal{L}_{neg} \\
&= -\mathbb{E}\left[\operatorname{sim}\left(\mathbf{f}\left(\mathcal{X}_p\right), \mathbf{f}\left(\mathcal{Y}_q\right)\right) \mid p = q\right] + \lambda \mathbb{E}\left[\operatorname{sim}\left(\mathbf{f}\left(\mathcal{X}_p\right), \mathbf{f}\left(\mathcal{Y}_q\right)\right) \mid p \neq q\right]
\end{aligned} \tag{5}$$

$$= \mathbb{E}\left[\frac{\lambda}{1-r_p} \operatorname{sim}\left(\mathbf{f}\left(\mathcal{X}_p\right), \mathbf{f}\left(\mathcal{Y}_q\right)\right)\right] - \mathbb{E}\left[\left(\frac{\lambda r_p}{1-r_p} + 1\right) \operatorname{sim}\left(\mathbf{f}\left(\mathcal{X}_q\right), \mathbf{f}\left(\mathcal{Y}_q\right)\right) \mid p = q\right]. \tag{6}$$

From Eqn. (5) to Eqn. (6), we use the results in Eqn. (4)

Two-sided Bound.

The above form still requires label rate information, i.e., $r_p, \forall p$, we therefore consider using the following approximation to the above loss $\mathcal{L}_{ss}$,

$$\mathcal{L}_{ss}^{\Theta}(\gamma) = (\gamma + 1)\mathbb{E}\left[\operatorname{sim}\left(\mathbf{f}\left(\mathcal{X}_p\right), \mathbf{f}\left(\mathcal{Y}_q\right)\right)\right] - \mathbb{E}\left[\operatorname{sim}\left(\mathbf{f}\left(\mathcal{X}_p\right), \mathbf{f}\left(\mathcal{Y}_q\right)\right) \mid p = q\right]. \tag{7}$$

Here, $\gamma \geq 0$ is a hyperparameter, while $\mathcal{L}_{ss}$ can be bounded as (derivations in appendix E),

$$C_1 \mathcal{L}_{ss}^{\Theta}\left(\frac{\lambda - 1}{C_1}\right) \leq \mathcal{L}_{ss} \leq C_2 \mathcal{L}_{ss}^{\Theta}\left(\frac{\lambda - 1}{C_2}\right), \; C_1 = 1 + \max_p \frac{\lambda r_p}{1-r_p}, C_2 = 1 + \min_p \frac{\lambda r_p}{1-r_p}. \tag{8}$$

The equivalence is established when $C_1 = C_2$, i.e., the class labels are balanced. To simplify the derivation, we ignore λ in the following and let γ be a new hyperparameter. Also, the constants C_1 and C_2 can be absorbed into a weight hyperparameter β, given in the next

section. This bound implies that, an easy-to-compute $\beta\mathscr{L}_{ss}^{\Theta}(\gamma)$ is often a good approximation of $\mathscr{L}_{ss}$ for some β. The next section specifies how to compute $\beta\mathscr{L}_{ss}^{\Theta}(\gamma)$ unsupervisedly as our *empirical self-supervised loss*.

3.2 The Objective of ATD

Empirical Estimator.—We obtain an empirical estimator of $\mathscr{L}_{ss}^{\Theta}$ with Monte Carlo method. Suppose $\mathcal{T}$ and $\widetilde{\mathcal{T}}$ are the input tensor and the augmented tensor respectively, and $\mathbf{X} = \mathbf{f}(\mathcal{T})$, $\widetilde{\mathbf{X}} = \mathbf{f}(\widetilde{\mathcal{T}}) \in \mathbb{R}^{N \times R}$ are the coefficient/feature matrices. We use the row vectors of $\mathbf{X}$, $\widetilde{\mathbf{X}}$ to estimate Eqn. (7).

The first term $\mathbb{E}\left[\mathrm{sim}\left(\mathbf{f}\left(\mathcal{X}_p\right), \mathbf{f}\left(\mathcal{Y}_q\right)\right)\right]$ essentially requires a random sampler, which is approximated by the average cosine similarity of all possible pairs of non-corresponding row vectors of $\mathbf{X}$, $\widetilde{\mathbf{X}}$, while the second term $\mathbb{E}\left[\mathrm{sim}\left(\mathbf{f}\left(\mathcal{X}_p\right), \mathbf{f}\left(\mathcal{Y}_q\right)\right) \mid p = q\right]$ requires a positive sampler, which is estimated by the average cosine similarity of all pairs of corresponding row vectors,

$$\begin{aligned}\widetilde{\mathscr{L}}_{ss}^{\Theta}(\gamma) = (\gamma+1)\cdot\frac{1}{N(N-1)}\sum_{n=1}^{N}\sum_{s\neq n}^{N}\left\langle\frac{\mathbf{x}^{(n)}}{\|\mathbf{x}^{(n)}\|_2},\frac{\widetilde{\mathbf{x}}^{(s)}}{\|\widetilde{\mathbf{x}}^{(s)}\|_2}\right\rangle \\ -\frac{1}{N}\sum_{n=1}^{N}\left\langle\frac{\mathbf{x}^{(n)}}{\|\mathbf{x}^{(n)}\|_2},\frac{\widetilde{\mathbf{x}}^{(n)}}{\|\widetilde{\mathbf{x}}^{(n)}\|_2}\right\rangle \\ = \mathrm{Tr}\left(\mathbf{X}^{\mathsf{T}}\mathbf{D}(\mathbf{X})\mathbf{G}(\gamma)\mathbf{D}(\widetilde{\mathbf{X}})\widetilde{\mathbf{X}}\right),\end{aligned} \tag{9}$$

where $\mathbf{D}(\mathbf{X}) = \mathrm{diag}\left(\frac{1}{\|\mathbf{x}^{(1)}\|_2}, \cdots, \frac{1}{\|\mathbf{x}^{(N)}\|_2}\right)$ is the row-wise scaling matrix and

$$\mathbf{G}(\gamma) = \begin{bmatrix} -\frac{1}{N} & \frac{\gamma+1}{N(N-1)} & \cdots & \frac{\gamma+1}{N(N-1)} \\ \frac{\gamma+1}{N(N-1)} & -\frac{1}{N} & \cdots & \frac{\gamma+1}{N(N-1)} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\gamma+1}{N(N-1)} & \frac{\gamma+1}{N(N-1)} & \cdots & -\frac{1}{N} \end{bmatrix}.$$

Note that, the form in Eqn. (9) is significantly different from tensor discriminant analysis (Jia et al., 2014; Tao et al., 2007), which integrates the actual label information as a regularizer and is also different from graph regularized tensor decomposition (Cai et al., 2010), which incorporates side information, such as geometrical positions (Maki et al., 2018). Compared to standard noise contrastive estimation (NCE) (Gutmann and Hyvärinen, 2010; Chen et al., 2020) in the area of contrastive SSL, our SSL form in Eqn. (9) considers a subtraction form instead of the softmax formulation, making it amenable to quadratic optimization, as we will show in Sec. 3.3.

Overall Objective.—According to Eqn. (8), the self supervised loss $\mathscr{L}_{ss}$ is bounded by $\mathscr{L}_{ss}^{\Theta}(\gamma)$, while the constants (i.e., C_1, C_2) can be absorbed into a weight hyperparameter β. We let the empirical self-supervised loss, $\widetilde{\mathscr{L}}_{ss} = \beta\widetilde{\mathscr{L}}_{ss}^{\Theta}(\gamma)$. Our objective follows both the *fitness* (i.e., CPD reconstruction loss) and *alignment* (i.e., self-supervised loss) principles, while

also considering Tikhonov regularization (Golub and Von Matt, 1997) to constrain the scale of all parameters,

$$\mathscr{L} = \mathscr{L}_{cpd} + \mathscr{L}_{reg} + \widetilde{\mathscr{L}}_{ss}, \tag{10}$$

where

$$\begin{aligned} \mathscr{L}_{cpd} &= \|\mathscr{T} - [\mathbf{X}, \mathbf{A}, \mathbf{B}, \mathbf{C}]\|_F^2 + \left\|\widetilde{\mathscr{T}} - [\widetilde{\mathbf{X}}, \mathbf{A}, \mathbf{B}, \mathbf{C}]\right\|_F^2, \\ \mathscr{L}_{reg} &= \alpha\left(\|\mathbf{X}\|_F^2 + \|\widetilde{\mathbf{X}}\|_F^2 + \|\mathbf{A}\|_F^2 + \|\mathbf{B}\|_F^2 + \|\mathbf{C}\|_F^2\right), \\ \widetilde{\mathscr{L}}_{ss} &= \beta\widetilde{\mathscr{L}}_{ss}^{\Theta}(\gamma) = \beta\mathrm{Tr}\left(\mathbf{X}^{\mathsf{T}}\mathbf{D}(\mathbf{X})\mathbf{G}(\gamma)\mathbf{D}(\widetilde{\mathbf{X}})\widetilde{\mathbf{X}}\right). \end{aligned} \tag{11}$$

The objective has (i) three hyperparameters, $\gamma, \alpha, \beta > 0$; (ii) five factor matrices, $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{X}, \widetilde{\mathbf{X}}$.

3.3 Alternating Least Squares Optimization

Ideally, we would like to use alternative least squares (ALS) algorithm (Sidiropoulos et al., 2017) for optimizing the objective, where each factor matrix is updated in a sequence by solving least squares subproblems. With large scale tensors, we can also resort to mini-batch stochastic ALS (Cao et al., 2020) to reduce memory footprint of the decomposition. However, the objective in Eqn. (11) is non-convex to $\mathbf{X}$ and $\widetilde{\mathbf{X}}$, respectively. For addressing the non-convex subproblem, we design an iterative method in this section, which only involves solving least square problems.

Addressing Non-convex Subproblem.—We use the subproblem of $\mathbf{X}$ as an example. Given $\mathbf{A}, \mathbf{B}, \mathbf{C}, \widetilde{\mathbf{X}}$ fixed, we want to minimize Eqn. (10) by finding the optimal solution, denotd as $\mathbf{X}^*$,

$$\mathbf{X}^* \leftarrow \arg\min_{\mathbf{X}}\left(\|\mathscr{T} - [\mathbf{X}, \mathbf{A}, \mathbf{B}, \mathbf{C}]\|_F^2 + \alpha\|\mathbf{X}\|_F^2 + \beta\mathrm{Tr}\left(\mathbf{X}^{\mathsf{T}}\mathbf{D}(\mathbf{X})\mathbf{G}(\gamma)\mathbf{D}(\widetilde{\mathbf{X}})\widetilde{\mathbf{X}}\right)\right). \tag{12}$$

- First, we are interested to find that the matrix-formed problem in Eqn. (12) can be decomposed into row-wise subproblems and to obtain the solution of Eqn. (12), it is suffice to solve each subproblem independently. Let us consider the subproblem of the k-th row, which is

$$\arg\min_{\mathbf{x}}\left(\left\|\mathscr{T}^{(k)} - [\mathbf{x}, \mathbf{A}, \mathbf{B}, \mathbf{C}]\right\|_F^2 + \alpha\|\mathbf{x}\|_F^2 + \beta\mathrm{Tr}\left(\frac{\mathbf{x}^{\mathsf{T}}}{\|\mathbf{x}\|_2}\mathbf{g}^{(k)}\mathbf{D}(\widetilde{\mathbf{X}})\widetilde{\mathbf{X}}\right)\right), \tag{13}$$

  where $\mathscr{T}^{(k)}$ is the k-th slice of $\mathscr{T}$, and $\mathbf{g}^{(k)}$ is the k-th row of $\mathbf{G}(\gamma)$.

- Here, Eqn. (13) is still non-convex. We let the derivative of Eqn. (13) objective to be zero and arrange the terms, which yields,

$$\mathbf{x} = \mathbf{v}_1\mathbf{V}_3 - \frac{\beta\mathbf{v}_2}{2\|\mathbf{x}\|_2}\left(\mathbf{I} - \frac{\mathbf{x}^{\mathsf{T}}\mathbf{x}}{\|\mathbf{x}\|_2^2}\right)\mathbf{V}_3, \tag{14}$$

  where

$$\mathbf{v}_1 = \mathbf{T}_1^{(k)}(\mathbf{A} \odot \mathbf{B} \odot \mathbf{C}), \quad \mathbf{v}_2 = \mathbf{g}^{(k)}\mathbf{D}(\widetilde{\mathbf{X}})\widetilde{\mathbf{X}}, \quad \mathbf{V}_3 = (\mathbf{A}^\top\mathbf{A} * \mathbf{B}^\top\mathbf{B} * \mathbf{C}^\top\mathbf{C} + \alpha\mathbf{I})^{-1}.$$

Here $\mathbf{x}, \mathbf{v}_1, \mathbf{v}_2$ are row vectors and $\mathbf{V}_3$ is a matrix. $\mathbf{T}_1^{(k)}$ is the 1-mode unfolding of $\mathcal{T}^{(k)}$, $\odot$ is the Khatri-Rao product and * is the Hadamard product (i.e., element-wise product).

- We consider the following iterative rule and the fixed point is a solution for Eqn. (14), which is a stationary point of Eqn. (13),

$$\mathbf{x}_{\text{impr}} = \mathbf{v}_1\mathbf{V}_3 - \frac{\beta\mathbf{v}_2}{2\|\mathbf{x}_{\text{init}}\|_2}\left(\mathbf{I} - \frac{\mathbf{x}_{\text{init}}^\top\mathbf{x}_{\text{init}}}{\|\mathbf{x}_{\text{init}}\|_2^2}\right)\mathbf{V}_3. \tag{15}$$

We use an initial guess $\mathbf{x}_{\text{init}}$ (obtained by solving Eqn. (13) with while $\beta = 0$, which is a least square problem) to start and then we repeat Eqn. (15) for each row (i.e., each k) and let the improved guess be the initial guess, $\mathbf{x}_{\text{init}} \leftarrow \mathbf{x}_{\text{impr}}$, to iteratively improve the result.

Theorem 1 (proof in appendix A) ensures that Eqn. (15) converges linearly if β is chosen to be sufficiently small. In appendix B, we verify the liner convergence and also empirically show that one round of Eqn. (15) is sufficient in our experiment, where $\beta = 2$. The non-convex subproblem of $\widetilde{\mathbf{X}}$ can be solved in the same way.

Theorem 1. Given non-zero row vectors $\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{u}^0 \in \mathbb{R}^d$, non-singular matrix $\mathbf{V}_3 \in \mathbb{R}^{d\times d}$ and $\beta > 0$. The sequence $\{\mathbf{u}^t\}$, generated by $\mathbf{u}^{t+1} = \mathbf{v}_1\mathbf{V}_3 - \frac{\beta\mathbf{v}_2}{2\|\mathbf{u}^t\|_2}\left(\mathbf{I} - \frac{\mathbf{u}^{t\top}\mathbf{u}^t}{\|\mathbf{u}^t\|_2^2}\right)\mathbf{V}_3$, satisfies,

$$\|\mathbf{u}^{t+1} - \mathbf{u}^*\|_2 \le \frac{\beta(2m+M)\|\mathbf{v}_2\|_2\|\mathbf{V}_3\|_F}{m^3}\|\mathbf{u}^t - \mathbf{u}^*\|_2,$$

where $\mathbf{u}^*$ is the fixed point and $m = \min_t \|\mathbf{u}^t\|_2$, $M = \max_t \|\mathbf{u}^t\|_2$ are the bound of the sequence. With a good $\mathbf{u}^0$ and a sufficiently small β, we can safely assume $0 < m \le M < \infty$.

Optimization Procedures.—To minimize Eqn. (11), we alternatively update $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{X}$, and $\widetilde{\mathbf{X}}$, where each subproblem involves only solving least squares problems with closed-form solutions. With large-scale tensors (as in the experiments), we optimizes the factors in mini-batches. Between mini-batches, the basis factors $\mathbf{A}, \mathbf{B}, \mathbf{C}$ are shared and updated incrementally. We summarize the algortihms in appendix C. The computation head of the algorithm is matricized tensor times Khatri-Rao product (MTTKRP). The complexity of our optimization algorithm is asymptotically the same as applying CP-ALS, which costs *O(NIJKR)* to *sweep* over the whole tensor once.

4 Experiments

This section presents the experimental evaluations. Due to space limitation, additional details, including data augmentations and baseline implementation, are presented in appendix D.

4.1 Experimental Setup

Data Preparation.—We use four real-world datasets: (i) *Sleep-EDF* (Kemp et al., 2000), which contains EOG, EMG and EEG Polysomnography recordings; (ii) human activity recognition *(HAR)* (Anguita et al., 2013) with smartphone accelerometer and gyroscope data; (iii) Physikalisch Technische Bundesanstalt large scale cardiology database *(PTB-XL)* (Alday et al., 2020) with 12-lead ECG signals; (iv) Massachusetts General Hospital *(MGH)* (Biswal et al., 2018) datasets with multi-channel EEG waves. All datasets are split into three disjoint sets (i.e., unlabeled, training and test) by subjects, while training and test sets have labels. Basic statistics are shown in Table 1. All models (baselines and our ATD) use the same augmentation techniques: (a) jittering, (b) bandpass filtering, and (c) 3D position rotation. We provide an ablation study on the augmentation methods in appendix D.5.

Baseline Methods.—We include the following comparison models from different perspectives:

- **Tensor based models**: $\texttt{ATD}_{ss-}$ is our variant, which removes the self-supervised loss from the objective in Eqn. (10); *Stochastic alternating least squares (SALS)* applies on the the CPD objective with Tikhonov regularizer, which works on large tensors; *Graph regularized SALS (GR-SALS)* augments the objective of SALS with a graph regularizer (Maki et al., 2018; Cai et al., 2010), define as $\text{Tr}\left(\mathbf{X}^{\mathsf{T}}\mathbf{G}\mathbf{X}\right)$.
- **Self-supervised models**: *SimCLR-r* (Chen et al., 2020) and *BYOL-r* (Grill et al., 2020) are two popular SSL models with their own objective functions, where r indicates the size of the output representation.
- Auto-encoder models: *AE-r* denotes a CNN based autoencoder with mean square error (MSE) reconstruction loss, and AE_{ss}-*r* denotes the same autoencoder model with standard NCE loss in the bottleneck layer, where r is the representation size.

All models use the unlabeled set to train a feature encoder and use training and test sets to evaluate. Note that, deep neural network models use the same CNN backbone. In appendix D.8, we have also compared with two recent supervised tensor learning models, which shows the usefulness of our ATD and the large unlabeled set, especially in low-label rate scenarios.

Evaluation and Environments.—We evaluate model performance mainly based on *classification accuracy*, where we train an additional logistic classifier (He et al., 2020) on top of the feature encoder. Also, for different models, we compare their *number of learnable parameters*. The experiments are implemented by *Python 3.8.5, Torch 1.8.0+cu111* on a Linux workstation with 256 GB memory, 32 core CPUs (3.70 GHz, 128 MB cache), two

RTX 3090 GPUs (24 GB memory each). All training is performed on the GPU. For tensor based models, we use $R = 32$ and implement the pipeline in CUDA manually, instead of using *torch-autograd*.

4.2 Experimental Results

This section shows the experimental results on downstream classification. We use all the unlabeled data to train the encoder or feature extractor, and use training data (since Sleep-EDF and MGH datasets have enough training samples, we randomly selected a subset of them) for learning a downstream classifier and use all test data. Each experiment is conducted with five different random seeds and the mean and standard deviations are reported. The metrics are the *accuracy* and the *number of learnable parameters*. All models have 32-dim features in the end, except that for two self-supervised baselines and autoencoder models, which have 128-dim options.

4.2.1 Better Classification Accuracy with Fewer Parameters—From Table 2, ATD shows comparable or better performance over the baselines. We have also reported the running time per epoch/sweep in appendix D.7 for all models. Compared to the variant ATD_{ss-}, our ATD can improve the accuracy by 1.0% ~ 1.9%, which shows the benefit of the inclusion of self-supervised loss. SALS and ATD_{ss-} have similar performance, while their objectives differ in that ATD_{ss-} considers the Frobenius norm of the augmented data. Thus, their accuracy gap is caused by the use of data augmentation. Also, the experiments show that the *fitness* and *alignment* principles are both important. We observe that with a self-supervised loss (i.e., *alignment*), AE_{ss} can give significant improvement over AE, while ATD shows ~ 8% accuracy gain over the self-supervised models on MGH dataset, since we can better preserve the data with a reconstruction loss (i.e., *fitness*). Moreover, the table shows that tensor based models require fewer parameters, i.e., less than 5% of parameters compared to deep learning models. On HAR, the deep unsupervised models show poor performance due to (i) they may not optimize a large number of parameters on middle-scale dataset; (ii) movement signals in HAR might have few degrees of freedom, which matches well with the low-rank assumption of tensor methods. On large-scale Sleep-EDF, self-supervised models outperforms ATD marginally since they have more parameters thus can capture more information.

4.2.2 Better Performance in Low-label Rate Scenarios—On the MGH dataset, we also show the effect of varying the amount of training data in Figure 2. We include an end-to-end convolutional neural network (CNN) model based on (Biswal et al., 2018), called *Reference CNN*, which is a supervised model and only uses the training and test sets. To be more readable, we separate the comparison figure into two sub-figures: the left compares our ATD model with self-supervised and auto-encoder baselines and the right one compares ATD with tensor baselines and the reference model, and the scale of y-axis on two sub-figures are the same.

We find that all unsupervised models outperform the supervised reference CNN model in scenarios with fewer training samples. With more training data, the performance of all models get improved, especially the reference CNN model, which can optimize the encoder

and predictive layers in an end-to-end way and finally outperforms our ATD when more training samples is available.

4.2.3 Stable Results with Hyperparameter Variation—For a comprehensive evaluation, we also conduct ablation studies on the effect of the data augmentation methods and on hyperparameters. Due to space limitation, we move the experimental settings and results to appendix D.6, while summarizing the general conclusions here: (i) with more diverse data augmentation methods, the final results are relatively better; (ii) with a larger rank R, the performance will be better generally; (iii) our ATD is not sensitive hyperparameters α, β, γ.

5 Conclusion

This paper introduces the concept of self-supervised learning for tensors and proposes *Augmented Tensor Decomposition* (ATD) and the ALS-based optimization. We show that by explicitly contrasting positive and negative samples, the decomposition results are more aligned with downstream classification. On four real-world datasets, we show the advantages of our model over various unsupervised models and with fewer training data, our model even outperforms the reference supervised models.

Compared to deep learning methods, tensor based models are not as flexible in processing multimodal and diverse inputs, such as natural images. However, applying tensor decomposition on the outputs of earlier layers of pre-trained deep neural networks may be a feasible way to address the weaknesses. This direction would be interesting for future work.

A: Proof of theorem

Proof. We provide the proof below (first by submultiplicativity),

$$\begin{aligned}\|\mathbf{u}^{t+1}-\mathbf{u}^{*}\|_2 &= \left\|\frac{\beta\mathbf{v}_2}{2\|\mathbf{u}^{*}\|_2}\left(\mathbf{I}-\frac{\mathbf{u}^{*\top}\mathbf{u}^{*}}{\|\mathbf{u}^{*}\|_2^2}\right)\mathbf{V}_3-\frac{\beta\mathbf{v}_2}{2\|\mathbf{u}^{t}\|_2}\left(\mathbf{I}-\frac{\mathbf{u}^{t\top}\mathbf{u}^{t}}{\|\mathbf{u}^{t}\|_2^2}\right)\mathbf{V}_3\right\|_2\\ &\le \left\|\frac{\beta\mathbf{v}_2}{2}\right\|_2\|\mathbf{V}_3\|_F\left\|\frac{1}{\|\mathbf{u}^{*}\|_2}\left(\mathbf{I}-\frac{\mathbf{u}^{*\top}\mathbf{u}^{*}}{\|\mathbf{u}^{*}\|_2^2}\right)-\frac{1}{\|\mathbf{u}^{t}\|_2}\left(\mathbf{I}-\frac{\mathbf{u}^{t\top}\mathbf{u}^{t}}{\|\mathbf{u}^{t}\|_2^2}\right)\right\|_2\\ &= \left\|\frac{\beta\mathbf{v}_2}{2}\right\|_2\|\mathbf{V}_3\|_F\left\|\left(\frac{1}{\|\mathbf{u}^{*}\|_2}-\frac{1}{\|\mathbf{u}^{t}\|_2}\right)+\left(\frac{\mathbf{u}^{t\top}\mathbf{u}^{t}}{\|\mathbf{u}^{t}\|_2^3}-\frac{\mathbf{u}^{*\top}\mathbf{u}^{*}}{\|\mathbf{u}^{*}\|_2^3}\right)\right\|_2.\end{aligned}$$

Then, we decompose the last term into two parts and have

$$\begin{aligned}\frac{1}{\|\mathbf{u}^{*}\|_2}-\frac{1}{\|\mathbf{u}^{t}\|_2} &= \frac{\|\mathbf{u}^{t}\|_2-\|\mathbf{u}^{*}\|_2}{\|\mathbf{u}^{*}\|_2\|\mathbf{u}^{t}\|_2}\le\frac{\|\mathbf{u}^{t}-\mathbf{u}^{*}\|_2}{\|\mathbf{u}^{*}\|_2\|\mathbf{u}^{t}\|_2}\le\frac{\|\mathbf{u}^{t}-\mathbf{u}^{*}\|_2}{m^2},\\ \frac{\mathbf{u}^{t\top}\mathbf{u}^{t}}{\|\mathbf{u}^{t}\|_2^3}-\frac{\mathbf{u}^{*\top}\mathbf{u}^{*}}{\|\mathbf{u}^{*}\|_2^3} &= \left(\frac{\mathbf{u}^{t\top}\mathbf{u}^{t}}{\|\mathbf{u}^{t}\|_2^3}-\frac{\mathbf{u}^{*\top}\mathbf{u}^{t}}{\|\mathbf{u}^{t}\|_2^3}\right)+\left(\frac{\mathbf{u}^{*\top}\mathbf{u}^{t}}{\|\mathbf{u}^{t}\|_2^3}-\frac{\mathbf{u}^{*\top}\mathbf{u}^{*}}{\|\mathbf{u}^{t}\|_2^3}\right)+\left(\frac{\mathbf{u}^{*\top}\mathbf{u}^{*}}{\|\mathbf{u}^{t}\|_2^3}-\frac{\mathbf{u}^{*\top}\mathbf{u}^{*}}{\|\mathbf{u}^{*}\|_2^3}\right)\\ &= \frac{(\mathbf{u}^{t\top}-\mathbf{u}^{*\top})\mathbf{u}^{t}}{\|\mathbf{u}^{t}\|_2^3}+\frac{\mathbf{u}^{*\top}(\mathbf{u}^{t}-\mathbf{u}^{*})}{\|\mathbf{u}^{t}\|_2^3}+\frac{\|\mathbf{u}^{*}\|_2^2(\|\mathbf{u}^{*}\|_2^3-\|\mathbf{u}^{t}\|_2^3)}{\|\mathbf{u}^{t}\|_2^3\|\mathbf{u}^{*}\|_2^3}\\ &\le \frac{\|\mathbf{u}^{t}-\mathbf{u}^{*}\|_2}{m^2}+\frac{M\|\mathbf{u}^{t}-\mathbf{u}^{*}\|_2}{m^3}+\frac{(M+2m)\|\mathbf{u}^{t}-\mathbf{u}^{*}\|_2}{m^3}.\end{aligned}$$

The proof is complete by triangular inequality.

B: Analysis of the Iterative Rule

In this section, we study how many rounds of the iterative rules are needed to achieve a good classification result. This study is conducted on HAR and Sleep-EDF with five random seeds.

Linear Convergence Speed.

First, we study the convergence speed (when $\beta = 2$). We consider two scenarios: (Scenario 1) when the bases $\{\mathbf{A}, \mathbf{B}, \mathbf{C}\}$ are initialized as random matrices; (Scenario 2) when the based $\{\mathbf{A}, \mathbf{B}, \mathbf{C}\}$ are already learned. We use the average relative difference (of the F-norm) as the convergence measure, i.e., $\frac{1}{N}\sum_{k=1}^{N}\frac{\|\mathbf{x}_{t+1}^{(k)} - \mathbf{x}_{t}^{(k)}\|}{\|\mathbf{x}_{t}^{(k)}\|} = \frac{1}{N}\sum_{k=1}^{N}\frac{\|\mathbf{x}_{\text{impr}}^{(k)} - \mathbf{x}_{\text{init}}^{(k)}\|}{\|\mathbf{x}_{\text{init}}^{(k)}\|}$, where $\mathbf{x}^{(k)}$ means the k-th row of $\mathbf{X}$ and t means the number of rounds of the iterative rule. We test on $t = 1, 2, 3, 4, 8$. The comparison is shown in Figure 3. Both scenarios verify the linear convergence speed.

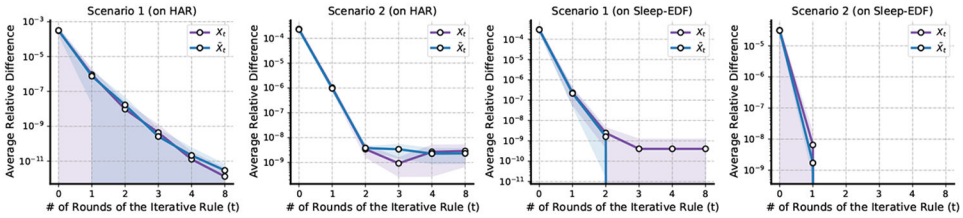

Figure 3:
Verification of Convergence Speed. When the # of iteration t becomes larger, the average relative difference can exceed the minimum precision and go to zero (like Scenario 2 on Sleep-EDF).

One Round is Enough.

We consider the performance of downstream classification with different rounds of iterative rules. The results are shown in Table 3 and Table 4.

Table 3:

Performance with Different Rounds (on HAR)

# of rounds (t)	1	2	3	4	8
time per sweep	8.774s	10.316s	11.224s	12.781s	17.402s
accuracy (%)	93.30 ± 0.413	93.35 ± 0.172	93.35 ± 0.150	93.35 ± 0.122	93.35 ± 0.122

Table 4:

Performance with Different Rounds (on Sleep-EDF)

# of rounds (t)	1	2	3	4	8
time per sweep	148.375s	160.924s	173.361s	188.193s	242.386s
accuracy (%)	85.25 ± 0.209	85.33 ± 0.173	85.31 ± 0.178	85.31 ± 0.177	85.31 ± 0.177

Algorithm 1: Alternating Least Squares

Input: Data tensor $\mathcal{T} \in \mathbb{R}^{N \times I \times J \times K}$; initialized $\mathbf{A}, \mathbf{B}, \mathbf{C}, \widetilde{\mathbf{X}}, \mathbf{X}$; other hyperparameters α, β, γ;
Obtain the augmented tensor $\widetilde{\mathcal{T}}$;
repeat
  Use $\mathbf{A}, \mathbf{B}, \mathbf{C}, \widetilde{\mathbf{X}}$ to update $\mathbf{X}$ by our iterative rules (one iteration) in Eqn. (15);
  Use $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{X}$ to update $\widetilde{\mathbf{X}}$ by our iterative rules (one iteration) in Eqn. (15);
  Use $\mathbf{B}, \mathbf{C}, \mathbf{X}, \widetilde{\mathbf{X}}$ to update $\mathbf{A}$ by solving least square problem;
  Use $\mathbf{A}, \mathbf{C}, \mathbf{X}, \widetilde{\mathbf{X}}$ to update $\mathbf{B}$ by solving least square problem;
  Use $\mathbf{A}, \mathbf{B}, \mathbf{X}, \widetilde{\mathbf{X}}$ to update $\mathbf{C}$ by solving least square problem;
until *max sweep exceeds or change of loss* $< 0.1\%$ *within 3 consecutive sweeps*;
Output: the learned bases $\{\mathbf{A}, \mathbf{B}, \mathbf{C}\}$.

Algorithm 2: Mini-batch Alternating Least Squares

Input: Data tensor $\mathcal{T} \in \mathbb{R}^{N \times I \times J \times K}$; initialized $\{\mathbf{A}^1, \mathbf{B}^1, \mathbf{C}^1\}$; batch size b; learning rate η; other hyperparameters α, β, γ; initial counter $l = 1$;
repeat
  shuffle the data tensor $\mathcal{T}$; / * start a new sweep * /
  for *a tensor batch* $\mathcal{T}^l \in \mathbb{R}^{b \times I \times J \times K}$ *and its augmentation* $\widetilde{\mathcal{T}}^l$ **do**
    use $\mathbf{A}^l, \mathbf{B}^l, \mathbf{C}^l$ to initialize $\mathbf{X}$ based on $\mathcal{T}^l$;
    use $\mathbf{A}^l, \mathbf{B}^l, \mathbf{C}^l$ to initialize $\widetilde{\mathbf{X}}$ based on $\widetilde{\mathcal{T}}^l$;
    Use $\mathbf{A}^l, \mathbf{B}^l, \mathbf{C}^l, \widetilde{\mathbf{X}}$ to update $\mathbf{X}$ by our iterative rules (one iteration) in Eqn. (15);
    Use $\mathbf{A}^l, \mathbf{B}^l, \mathbf{C}^l, \mathbf{X}$ to update $\widetilde{\mathbf{X}}$ by our iterative rules (one iteration) in Eqn. (15);
    Use $\mathbf{B}^l, \mathbf{C}^l, \mathbf{X}, \widetilde{\mathbf{X}}$ to obtain $\mathbf{A}^{l+1}$ by solving least square problem;
    Use $\mathbf{A}^{l+1}, \mathbf{C}^l, \mathbf{X}, \widetilde{\mathbf{X}}$ to obtain $\mathbf{B}^{l+1}$ by solving least square problem;
    Use $\mathbf{A}^{l+1}, \mathbf{B}^{l+1}, \mathbf{X}, \widetilde{\mathbf{X}}$ to obtain $\mathbf{C}^{l+1}$ by solving least square problem;
    $l = l + 1$ / * increment the counter * /;
  end
until *max sweep exceeds or change of loss* $< 0.1\%$ *within 3 consecutive sweeps*;
Outpu: the learned bases $\{\mathbf{A}^L, \mathbf{B}^L, \mathbf{C}^L\}$.

We observe that with an increasing number of rounds of iterative rule, the classification results will not improve further, however, the time consumption increases. Thus, we use only one round of the iterative rule in our experiments.

C: Algorithm

For small tensors, the algorithmic pipeline is presented in Algorithm 1. For large tensors, we use mini-batch alternating least squares optimization in Algorithm 2.

D: Experimental Details

D.1 Dataset Processing

Sleep-EDF is public, which contains 153 full-night EEG (from Fpz-Cz and Pz-Oz electrode locations), EOG (horizontal), and submental chin EMG recordings, under Open Data Commons Attribution License v1.0 and *MGH Sleep* is provided by (Biswal et al., 2018), where F3-M2, F4-M1, C3-M2, C4-M1, O1-M2, O2-M1 channels are used, containing 6,478 recordings. These two EEG datasets are processed in a similar way. First, the raw data are (long) recordings of each subject. On subject-level, these recordings are categorized into unlabeled and labeled sets by 90% : 10%. Then the labeled sets are further separated into training and test by 5% : 5%. Next, within each set (unlabeled, training, test), recordings are further segmented into disjoint 30-second-long periods, which are the data samples in our study. Each data sample is represented as a matrix, *channel by timestamp*, and they are associated with one of five sleep stages, Awake (W), Non-REM stage 1 (N1), Non-REM stage 2 (N2), Non-REM stage 3 (N3), and REM stage (R). *HAR* is also public, collected as 3-axial linear acceleration and 3-axial angular velocity at a constant rate of 50Hz by the embedded accelerometer and gyroscope. It has been randomly partitioned into 70% and 30%. We use 70% as the unlabeled data (labels removed) and split the other part: 15% as training data and 15% as the test data. The license of this dataset is included in their citation. *PTB-XL* is a public ECG dataset, which contains 21,837 clinical 12-lead ECGs (male: 11,379 and female: 10,458) of 10-second length with a sampling frequency of 500 Hz. We randomly split the dataset into unlabeled (remove the labels) and labeled sets by 90% : 10%; then the labeled sets are further separated into training and test by 5% : 5%. This dataset is under Open BSD 3.0.

All datasets are de-identified (e.g., no names, no locations), and there is no offensive content. All the labels are also provided along with the datasets. The label distributions are shown below.

Table 5:

Class Label Distribution

Name	Label Distribution
Sleep-EDF	W: 68.8%, N1: 5.2%, N2: 16.6%, N3: 3.2%, R: 6.2%
HAR	Walk: 16.72%, Walk upstairs: 14.99%, Walk downstairs: 13.65%, Sit: 17.25%, Stand: 18.51%, Lay: 18.88%

Name	Label Distribution
PTB-XL	Male: 52.11%, Female: 47.89%
MGH Sleep	W: 44.3%, N1: 9.9%, N2: 14.4%, N3: 17.6%, R: 13.8%

D.2 STFT Transform

We find that directly using the raw data (spatial information) does not provide good results, even for the deep learning models. Thus, we take Short-Time Fourier Transforms (STFT) as a preprocessing step. From a single channel, we can extract both the amplitude and phase information, which is then stacked together as two different channels. After STFT, each data sample becomes a three-order tensor, *channel by frequency by timestamp*. The FFT size is 256 and hop length is 32 for Sleep-EDF; the FFT size is 64 and hop length is 2 for HAR; the FFT size is 256 and hop length is 64 for PTB-XL; and the FFT size is 512 and hop length is 128 for MGH. We use these third-order tensors as final input data samples for all models.

D.3 Data Augmentation

In our work, we build our feature extractor $\mathbf{f}(\cdot)$ from tensor decomposition tool, which may not be as expressive/flexible as deep neural networks in SSL, and thus we also ask the augmentation methods to allow a similar component-based representation for the perturbed data. Use EEG signals as an example, we consider jittering and bandpass filtering as two augmentation methods in the experiments, which perturb the signal frequency information and will not significantly change the low-rank structure of the data.

As mentioned in the main text, we consider three different augmentation methods: (i) *Jittering* adds additional perturbations to each sample. We consider both high and low-frequency noise on each channel independently. For high-frequency noise, we first generate a noisy sequence $\mathbf{s}$, which has the same length as the signal channel, and each element of $\mathbf{s}$ is i.i.d. sampled from a uniform distribution $U[-1, 1]$. We then control the amplitude of $\mathbf{s}$ by the noisy degree $d \in \mathbb{R}$. Finally, we add the scaled noisy sequence $d \cdot \mathbf{s}$ to the channel. In the experiment, $d = 0.05$ for Sleep-EDF, $d = 0.002$ for HAR, $d = 0.001$ for PTB-XL and $d = 0.01$ for MGH. For low-frequency noise, we generate a short noisy sequence (the length is randomly sampled from a uniform distribution U[100, length of channel]) in the same way and then use *scipy.interpolate.interp1d* to interpolate the noisy sequence to be at the same length as the channel. The choice of high-frequency noise or low-frequency noise, or both are coin-tossed with equal probability. (ii) *Bandpass filtering* reduces signal noise. We use the order-1 Butterworth filter by *scipy.signal.butter* to preserve only the within-band frequency information. The high-pass and low-pass are $(1Hz, 30Hz)$ and $(10Hz, 49Hz)$ for Sleep-EDF, $(1Hz, 20Hz)$ and $(5Hz, 24.5Hz)$ for HAR, $(1Hz, 30Hz)$ and $(10Hz, 50Hz)$ for PTB-XL, $(1Hz, 30Hz)$ and $(10Hz, 50Hz)$ for MGH. Low-pass or high-pass or both are selected with equal probability. Also, the bandpass filtering is applied to each channel independently. The intuition is that the low-pass signals and high-pass signals might be both useful. (iii) *3D position rotation* is an augmentation technique used only for HAR datasets, which have x-y-z axis information from accelerometer and gyroscope sensors. We apply

a 3D x-y-z coordinate system rotation by a rotation matrix to mimic different cellphone positions. All augmentation methods are applied in sequence (i) (ii) (iii). The STFT is performed after the data augmentation.

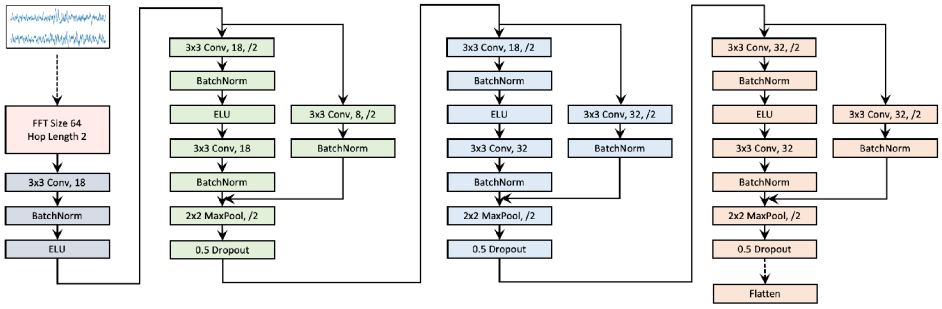

Figure 4:
Backbone CNN Architecture (HAR). The architectures for other three datasets are similar.

D.4 Implementation

Since all deep learning baselines are based on CNN, we use the same backbone model, shown in Figure 5. The model is adopted from (Cheng et al., 2020). Based on the backbone model, we add a fully connected layer for the reference CNN model, add non-linear layers for self-supervised models (they also have their respective loss), and add corresponding deconvolutional layers for autoencoder models. The reference CNN models is end-to-end, and they are trained on the training set; other baselines learn a low-dimensional feature extractor from an unlabeled set, and then a logistic classifier is trained on the training set, on top of the feature extractor. Without loss of generality, we use 128 as batch size. For deep learning models, we use Adam optimizer with a learning rate 1×10^{-3} and weight decaying 5×10^{-4}. We use 2×10^{-3} as the learning rate for our `ATD`. Since the paper deals with unsupervised learning and uses standard logistic regression (*sklearn.linear_model.LogisticRegression*) to evaluate, the hyperparameters of all models (except supervised model) are chosen based on the classification performance on the training data. For our ATD model, by default, we use $R = 32$, $\alpha = 1 \times 10^{-3}$, $\beta = 2$, $\gamma = b$ (which is the batch size), and all the reference configurations for each experiments are listed in code appendix. The choice of R depends on the trade-off between model fitness and time complexity. Specifically, a larger R means better fitness and more preserved information in the extracted representations, while the number of learnable parameters and time complexity also increases linearly with R. In our paper, we run simple CP decomposition on a small subset of the tensor and monitor the fitness curve. We find that the fitness does not improve much around $R = 32$ for all datasets (which means the real tensor might have a smaller rank and the residual part might be just noise). Thus, we choose $R = 32$ throughout the paper.

D.5 Ablation Studies on Data Augmentation

Effect of High-quality Data Augmentations.

We study the effects of high-quality data augmentation methods. Let us assume *(a): jittering, (b): bandpass filtering, (c): 3D position rotation*. We test on different combinations of the

augmentation methods. The experiments are conducted on two datasets: (i) for the MGH dataset, we use 50,000 unlabeled data, 5,000 training samples, and all test samples; (ii) for the HAR dataset, we use all data. We use 128 as the batch size for MGH Sleep, 64 for HAR, and $R = 32$ as the tensor rank.

Table 6 and Table 7 conclude that the augmentation methods influence the final classification results. However, for different datasets, the effects are different. We observe that for the MGH Sleep dataset, bandpass filtering works better than jittering. In HAR, combinations of augmentations work better than the individual augmentations. Overall, we find that with more diverse data augmentation methods, the final results are relatively better. The study of how to choose/design (or even automatically generate) better augmentation techniques will be our future work.

Table 6:

Ablation Studies on Data Augmentation (MGH Sleep)

Method	Acc (%)	Method	Acc (%)	Method	Acc (%)
(a)	72.53 ± 0.652	(b)	73.60 ± 0.270	(a)+(b)	74.15 ± 0.203

Table 7:

Ablation Studies on Data Augmentation (HAR)

Method	Acc (%)	Method	Acc (%)	Method	Acc (%)
(a)	92.06 ± 0.621	(a)+(b)	92.70 ± 0.266	(a)+(b)+(c)	93.35 ± 0.357
(b)	91.99 ± 0.274	(a)+(c)	92.93 ± 0.590	/	/
(c)	92.29 ± 0.330	(b)+(c)	92.26 ± 0.479	/	/

Table 8:

Performance of Low-quality Data Augmentation (on MGH and HAR)

MGH	$d = 0.02$	$d = 0.05$	$d = 0.1$	$d = 0.5$	$d = 5$
Accuracy	74.18 ± 0.326	74.10 ± 0.302	73.85 ± 0.530	73.39 ± 0.493	72.18 ± 0.676
MGH	SALS	ATD	(A)	(B)	(C)
Accuracy	71.93 ± 0.379	74.15 ± 0.431	72.73 ± 0.624	72.10 ± 0.719	70.75 ± 0.771
HAR	SALS	ATD	(A)	(B)	(C)
Accuracy	91.86 ± 0.295	93.35 ± 0.357	92.04 ± 0.308	92.48 ± 0.469	91.43 ± 0.835

Impact of Low-quality Data Augmentations.

We also study the impact of low-quality data augmentations. We use the SALS model (mini-batch tensor baseline without data augmentation) and our ATD as the reference models and conduct the following experiments on MGH and HAR:

- MGH-d: we change the d values for the jittering data augmentation, d means the ratio of the amplitude of the high/low frequency noise over the amplitude of the signal.
- MGH-(A): on MGH data, we set the degree of the *jittering* method to an unrealistic value $d = 1$ as the low-quality augmentation method, meaning that the magnitude of the noise is the same as the magnitude of the signals. We keep the *bandpass filtering* unchanged.
- MGH-(B): on MGH data, we randomly create a *bandpass filter* from ($1Hz$, $10Hz$) or ($30Hz$, $50Hz$) as the low-quality augmentation method, which drops the critical middle-band information. We keep the *jittering* unchanged.
- MGH-(C): on MGH data, we combine the above two low-quality augmentation methods.
- HAR-(A)(B)(C): follow the similar low-quality data augmentation method design on HAR. For (B), we use ($1Hz$, $5Hz$) and ($20Hz$, $24.5Hz$) as the low-quality bandpass.

The performances are shown in Table 8. We find that low-quality data augmentations will hurt the learned tensor bases and hinder the downstream classification. With d changing from 0.02 to 5, the *jittering* method becomes more unrealistic and the generated samples deviate from the real signal data distribution. Thus, we can find that the final classification performance also becomes worse gradually. If all data augmentation methods are of low-quality, the performance cannot surpass the base SALS model. We also find that the performance drop is not significant. The reason might be that even the augmentation methods are of low-quality, the Frobenius loss can still enforce the model to learn decent subspaces for better fitness and find generic low-rank features (opposed to the classification-oriented low-rank features), in this case, the features only follow fitness principle not the alignment principle, so the performance will be similar compared to SALS.

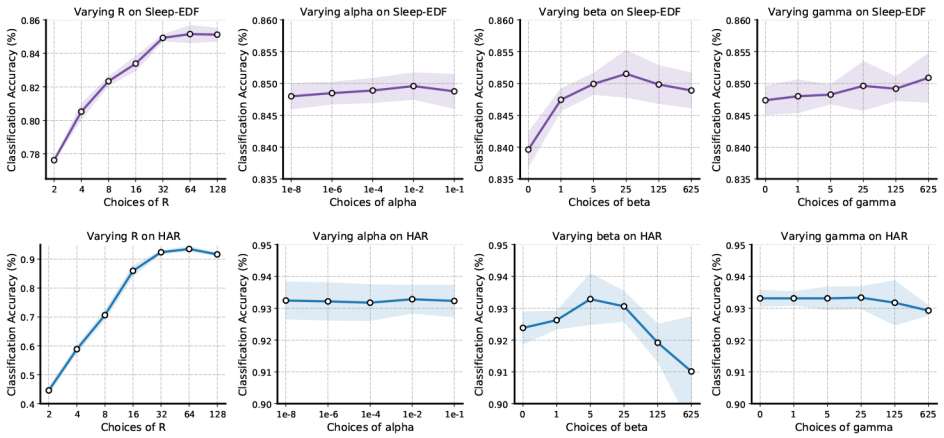

Figure 5:
Ablation Studies on Hyperparameters

Table 9:

Comparison with Tensor-based Methods with Different R (on HAR)

Model	$R = 8$	$R = 16$	$R = 32$	$R = 64$	$R = 128$
SALS	69.59 ± 0.526	83.92 ± 0.416	91.84 ± 0.295	91.89 ± 0.217	91.55 ± 0.388
GR-SALS	69.62 ± 0.458	84.20 ± 0.727	92.33 ± 0.282	92.28 ± 0.359	91.84 ± 0.534
ATD_{ss-}	70.27 ± 0.488	84.84 ± 0.557	92.41 ± 0.391	92.71 ± 0.243	92.32 ± 0.330
ATD	71.91 ± 0.253	85.61 ± 0.294	93.35 ± 0.357	93.43 ± 0.411	92.97 ± 0.273

Table 10:

Comparison with Tensor-based Methods with Different R (on Sleep-EDF)

Model	$R = 8$	$R = 16$	$R = 32$	$R = 64$	$R = 128$
SALS	81.26 ± 0.345	82.59 ± 0.638	84.27 ± 0.481	84.55 ± 0.527	84.49 ± 0.317
GR-SALS	81.72 ± 0.664	82.74 ± 0.481	84.33 ± 0.356	84.87 ± 0.486	84.90 ± 0.781
ATD_{ss-}	81.27 ± 0.568	82.5 ± 0.674	84.19 ± 0.221	84.47 ± 0.258	84.44 ± 0.577
ATD	82.49 ± 0.464	83.31 ± 0.591	85.01 ± 0.224	85.30 ± 0.483	85.32 ± 0.305

D.6 Ablation Studies on Hyperparameters

This section conducts ablation studies for decomposition rank R and other hyperparameters, α, β, γ. The experiments are conducted on Sleep-EDF with 50,000 random unlabeled data, 5,000 random training samples, and all test samples, and the HAR dataset.

The results are shown in Figure 5. First, we can conclude that with a larger decomposition rank R, the performance will be better generally. Though we observe that the performance worsens from $R = 64$ to $R = 128$, with limited training data, if the representation size (equals to R) becomes larger, the logistic regression model can overfit. We compare our model to other tensor-based methods with $R = 8, 16, 32, 64, 128$ and the Table 9 and Table 10 shows that ATD outperforms the tensor baselines consistently with different R.

Also, we find that the choices of α do not affect the final performance a lot. Finally, we find that it is easy for users to select the β values from a large range in general. For example, selecting a $\beta \in [5, 125]$ would guarantee good results on Sleep-EDF while on HAR, the selection range is [5, 25]. The choice of β does affect the final performance, but it is not tricky to search for a β for good performance. We also find that the accuracy score first increases then decreases with an increasing value of β. The reason might be that a large β will negatively affect the fitness loss.

D.7 Statistical Testing and Running Time Comparison

In this section, we conduct T-test on the result in Section 4.2.1 and calculate the p-values in the parenthesis of Table 11 and Table 12 (the experimental results are copied from Table 2).

Commonly, a p-value smaller than 0.05 would be considered as significant. We can see that our model show significant performance gain over all baselines on all tasks.

Table 11:

Result Significance and Running Time (%) for Sleep-EDF and HAR

	Sleep-EDF (5,000)			HAR (1,473)		
	Accuracy	**#of Params.**	**Time per sweep**	**Accuracy**	**# of Params.**	**Time per sweep**
Self-sup models:						
SimCLR-32	84.98 ± 0.358	210,384	260.299s	74.75 ± 0.723	53,286	8.459s
SimCLR-128	**85.19** ± **0.358**	222,768	265.809s	76.69 ± 0.697	65,670	8.532s
BYOL-32	84.29 ± 0.405	211,440	255.614s	73.71 ± 2.832	54,342	8.430s
BYOL-128	83.26 ± 0.337	239,280	257.266s	71.79 ± 1.866	82,182	8.478s
Auto-encoders:						
AE-32	74.78 ± 0.723	217,216	153.684s	63.13 ± 0.775	62,940	7.530s
AE-128	75.17 ± 0.897	241,888	156.813s	60.52 ± 1.604	87,612	7.662s
AE_{ss}-32	80.92 ± 0.345	217,216	301.773s	71.70 ± 2.135	62,940	7.765s
AE_{ss}-128	81.84 ± 0.259	241,888	307.546s	72.43 ± 1.370	87,612	7.804s
Tensor models:						
SALS	84.27 ± 0.481 (0.0041)	7,328	86.281s	91.86 ± 0.295 (2e-5)	2,688	7.535s
GR-SALS	84.33 ± 0.356 (0.0019)	7,328	109.916s	92.33 ± 0.282 (0.0003)	2,688	7.829s
ATD_{ss-}	84.19 ± 0.221 (9e-5)	7,328	147.568s	92.41 ± 0.391 (0.0011)	2,688	8.604s
ATD	85.01 ± 0.224	7,328	148.375s	**93.35** ± **0.357**	2,688	8.672s

result format: mean ± standard deviation (p-value)

Table 12:

Result Significance and Running Time (%) for PTB-XL and MGH

	PTB-XL (2,183)			MGH (5,000)		
	Accuracy	**# of Params.**	**Time per sweep**	**Accuracy**	**# of Params.**	**Time per sweep**
Self-sup models:						
SimCLR-32	69.25 ± 0.355	200,960	18.714s	67.34 ± 0.970	212,624	1449.368s
SimCLR-128	68.19 ± 0.793	237,920	19.037s	66.98 ± 1.331	246,608	1457.283s
BYOL-32	65.08 ± 1.535	202,016	18.410s	68.83 ± 1.168	214,736	1451.468s
BYOL-128	65.49 ± 0.612	254,432	18.680s	68.55 ± 1.339	279,632	1461.181s
Auto-encoders:						
AE-32	59.01 ± 0.896	224,528	11.229s	68.58 ± 0.427	220,088	851.118s
AE-128	58.29 ± 0.412	298,352	11.396s	67.05 ± 1.375	257,048	815.858s
AE_{ss}-32	68.47 ± 0.231	224,528	18.263s	71.46 ± 0.386	220,088	1486.244s
AE_{ss}-128	68.88 ± 0.604	298,352	18.465s	70.19 ± 0.617	257,048	1504.545s

	PTB-XL (2,183)			MGH (5,000)		
	Accuracy	**# of Params.**	**Time per sweep**	**Accuracy**	**# of Params.**	**Time per sweep**
Tensor models:						
SALS	69.15 ± 0.483 (0.0023)	7,296	8.988s	71.93 ± 0.379 (5e-6)	9,984	782.763s
GR-SALS	69.02 ± 0.477 (0.0012)	7,296	9.747s	72.35 ± 0.228 (8e-6)	9,984	970.292s
ATD_{ss-}	69.38 ± 0.612 (0.0129)	7,296	12.560s	72.78 ± 0.522 (0.0005)	9,984	1327.188s
ATD	**70.26** ± **0.523**	7,296	12.599s	**74.15** ± **0.431**	9,984	1360.569s

result format: mean ± standard deviation (p-value)

We have also reported the running time per sweep/epoch in the tables. When recording the running time, we duplicated the environment mentioned in Section 4.1, stopped other programs and ran all the models one by one on GPUs. We record the first 8 sweeps/epochs of all models and drop the first 3 sweeps (since they might be unstable). The average running time of the last 5 epochs are reported in Table 11 and Table 12 while the accuracy results are from Table 2. Note that on HAR, PTB-XL and Sleep-EDF, all methods use 128 as the batch size and on MGH, all methods use 512 as the batch size. The tensor based methods all use $R = 32$ as the rank. We can conclude that the tensor based methods are generally more time-efficient than the deep learning methods with fewer parameters. Since our model ATD and the variant ATD_{ss-} use the augmented tensors, they cost more compared to other tensor based methods (since the size of training tensors doubles), however, we also observe empirically that they can converge faster with around half number of the epochs.

D.8 Comparison with Supervised Tensor Learning

In this subsection, we compare our model with two supervised tensor learning baselines, UMLDA Lu et al. (2008) and supervised tensor learning (STL) Tao et al. (2005). **UMLDA** extracts uncorrelated discriminative features by sequential tensor-to-vector projections, and it includes two stages: in the first stage (supervised pre-training stage), it uses label information to maximize the Fisher's discrimination criterion (FDC) as the objective and extract uncorrelated features (i.e., representations); in the second stage (supervised learning stage), it uses another supervised model to map the representations to the labels. To make a fair comparison, we use 32 as the uncorrelated feature dimension (thus, it has the same number of learnable parameters as our model) and also use logistic regression (LR) for the second stage. **STL** is an end-to-end supervised tensor learning baseline, and it is originally proposed for binary classification with a rank-one parameterized tensor (i.e., outer product of multiple vectors). In the comparison, we extend STL for mult-class classification by including more rank-one parameterized tensors (one for each class). We use cross entropy loss to optimize the revised STL model. Both baselines are implemented with PyTorch and use the training and test set only. We have carefully turned the baseline models to achieve higher accuracy.

Table 13:

Comparison with Supervised Tensor Learning

Model	Sleep-EDF (5,000)	HAR (1,473)	PTB-XL (2,183)	MGH (5,000)
UMLDA (supervised pretrain + supervised LR)	81.06 ± 0.093	85.73 ± 1.16	65.55 ± 0.267	62.04 ± 0.722
STL (end-to-end supervised)	77.86 ± 0.816	80.52 ± 0.189	61.83 ± 0.712	41.44 ± 0.597
ATD (unsupervised pretrain + supervised LR)	**85.01** ± **0.224**	**93.35** ± **0.357**	**70.26** ± **0.523**	**74.15** ± **0.431**

Result Analysis.

The results are shown in Table 13. We can conclude that our methods outperform these two supervised tensor learning baselines significantly. The performance gap between UMLDA and our model can be explained by (i) UMLDA uses FDC criterion to design the loss function and also forces the learned feature dimensions to be uncorrelated, which might discard some essential class-dependent information; (ii) our model utilizes a large set of unlabeled data. STL gives poor performance because it is essentially a multilinear method with much fewer parameters than UMLDA and our ATD, which hurts the expressive.

E: Derivation of Two-sided Bound

We recall the definition of $\mathscr{L}_{ss}$ and $\mathscr{L}_{ss}^{\Theta}$ from Section 3.1.

$$\begin{aligned}\mathscr{L}_{ss} &= \mathscr{L}_{pos} + \lambda \mathscr{L}_{neg} \\ &= \mathbb{E}\left[\frac{\lambda}{1-r_p}\mathrm{sim}\left(\mathbf{f}\left(\mathscr{X}_p\right), \mathbf{f}\left(\mathscr{Y}_q\right)\right)\right] - \mathbb{E}\left[\left(\frac{\lambda r_p}{1-r_p}+1\right)\mathrm{sim}\left(\mathbf{f}\left(\mathscr{X}_p\right), \mathbf{f}\left(\mathscr{Y}_q\right)\right) \mid p=q\right], \\ \mathscr{L}_{ss}^{\Theta}(\gamma) &= (\gamma+1)\mathbb{E}\left[\mathrm{sim}\left(\mathbf{f}\left(\mathscr{X}_p\right), \mathbf{f}\left(\mathscr{Y}_q\right)\right)\right] - \mathbb{E}\left[\mathrm{sim}\left(\mathbf{f}\left(\mathscr{X}_p\right), \mathbf{f}\left(\mathscr{Y}_q\right)\right) \mid p=q\right],\end{aligned}$$

and we want to prove that

$$C_1 \mathscr{L}_{ss}^{\Theta}\left(\frac{\lambda-1}{C_1}\right) \le \mathscr{L}_{ss} \le C_2 \mathscr{L}_{ss}^{\Theta}\left(\frac{\lambda-1}{C_2}\right),\ C_1 = 1 + \max_p \frac{\lambda r_p}{1-r_p}, C_2 = 1 + \min_p \frac{\lambda r_p}{1-r_p},$$

where r_p is the label rate of class-p.

Proof. We start by arranging $\mathscr{L}_{ss}$,

$$\begin{aligned}\mathscr{L}_{ss} &= \mathbb{E}\left[\frac{\lambda}{1-r_p}\mathrm{sim}\left(\mathbf{f}\left(\mathscr{X}_p\right),\mathbf{f}\left(\mathscr{Y}_q\right)\right)\right] - \mathbb{E}\left[\left(\frac{\lambda r_p}{1-r_p}+1\right)\mathrm{sim}\left(\mathbf{f}\left(\mathscr{X}_p\right),\mathbf{f}\left(\mathscr{Y}_q\right)\right) \mid p=q\right] \\ &= \mathbb{E}\left[\left(\frac{\lambda r_p}{1-r_p}+\lambda\right)\mathrm{sim}\left(\mathbf{f}\left(\mathscr{X}_p\right),\mathbf{f}\left(\mathscr{Y}_q\right)\right)\right] \\ &- \mathbb{E}\left[\left(\frac{\lambda r_p}{1-r_p}+1\right)\mathrm{sim}\left(\mathbf{f}\left(\mathscr{X}_p\right),\mathbf{f}\left(\mathscr{Y}_q\right)\right) \mid p=q\right] \\ &= \mathbb{E}\left[\left(\frac{\lambda r_p}{1-r_p}+1\right)\mathrm{sim}\left(\mathbf{f}\left(\mathscr{X}_p\right),\mathbf{f}\left(\mathscr{Y}_q\right)\right)\right] + \mathbb{E}[(\lambda-1)\,\mathrm{sim}\left(\mathbf{f}\left(\mathscr{X}_p\right),\mathbf{f}\left(\mathscr{Y}_q\right)\right)] \\ &\quad - \mathbb{E}\left[\left(\frac{\lambda r_p}{1-r_p}+1\right)\mathrm{sim}\left(\mathbf{f}\left(\mathscr{X}_p\right),\mathbf{f}\left(\mathscr{Y}_q\right)\right) \mid p=q\right]\end{aligned} \tag{16}$$

$$\begin{aligned}&= \mathbb{E}_p\mathbb{E}_{q,\mathscr{X}_p,\mathscr{Y}_q}\left[\left(\frac{\lambda r_p}{1-r_p}+1\right)\mathrm{sim}\left(\mathbf{f}\left(\mathscr{X}_p\right),\mathbf{f}\left(\mathscr{Y}_q\right)\right)\right] + \mathbb{E}\left[(\lambda-1)\,\mathrm{sim}\left(\mathbf{f}\left(\mathscr{X}_p\right),\mathbf{f}\left(\mathscr{Y}_q\right)\right)\right] \\ &\quad - \mathbb{E}_p\mathbb{E}_{q,\mathscr{X}_p,\mathscr{Y}_q}\left[\left(\frac{\lambda r_p}{1-r_p}+\lambda\right)\mathrm{sim}\left(\mathbf{f}\left(\mathscr{X}_p\right),\mathbf{f}\left(\mathscr{Y}_q\right)\right) \mid p=q\right]\end{aligned} \tag{17}$$

$$\begin{aligned}&= \mathbb{E}_p\left[\left(\frac{\lambda r_p}{1-r_p}+1\right)\mathbb{E}_{q,\mathscr{X}_p,\mathscr{Y}_q}\left[\mathrm{sim}\left(\mathbf{f}\left(\mathscr{X}_p\right),\mathbf{f}\left(\mathscr{Y}_q\right)\right)\right]\right] + \mathbb{E}\left[(\lambda-1)\,\mathrm{sim}\left(\mathbf{f}\left(\mathscr{X}_p\right),\mathbf{f}\left(\mathscr{Y}_q\right)\right)\right] \\ &\quad - \mathbb{E}_p\left[\left(\frac{\lambda r_p}{1-r_p}+1\right)\mathbb{E}_{q,\mathscr{X}_p,\mathscr{Y}_q}\left[\mathrm{sim}\left(\mathbf{f}\left(\mathscr{X}_p\right),\mathbf{f}\left(\mathscr{Y}_q\right)\right) \mid p=q\right]\right] \\ &= \mathbb{E}_p\left[\left(\frac{\lambda r_p}{1-r_p}+1\right)\left(\mathbb{E}_{q,\mathscr{X}_p,\mathscr{Y}_q}\left[\mathrm{sim}(\mathbf{f}\left(\mathscr{X}_p\right),\mathbf{f}\left(\mathscr{Y}_q\right))\right] - \mathbb{E}_{q,\mathscr{X}_p,\mathscr{Y}_q}\left[\mathrm{sim}(\mathbf{f}\left(\mathscr{X}_p\right),\mathbf{f}\left(\mathscr{Y}_q\right)) \mid p=q\right]\right)\right] \\ &\quad + \mathbb{E}\left[(\lambda-1)\,\mathrm{sim}\left(\mathbf{f}\left(\mathscr{X}_p\right),\mathbf{f}\left(\mathscr{Y}_q\right)\right)\right]\end{aligned} \tag{18}$$

$$\begin{aligned}&\le \mathbb{E}_p\left[C_2\left(\mathbb{E}_{q,\mathscr{X}_p,\mathscr{Y}_q}\left[\mathrm{sim}\left(\mathbf{f}\left(\mathscr{X}_p\right),\mathbf{f}\left(\mathscr{Y}_q\right)\right)\right] - \mathbb{E}_{q,\mathscr{X}_p,\mathscr{Y}_q}\left[\mathrm{sim}\left(\mathbf{f}\left(\mathscr{X}_p\right),\mathbf{f}\left(\mathscr{Y}_q\right)\right) \mid p=q\right]\right)\right] \\ &\quad + \mathbb{E}\left[(\lambda-1)\,\mathrm{sim}\left(\mathbf{f}\left(\mathscr{X}_p\right),\mathbf{f}\left(\mathscr{Y}_q\right)\right)\right] \\ &= (C_2+\lambda-1)\mathbb{E}\left[\mathrm{sim}\left(\mathbf{f}\left(\mathscr{X}_p\right),\mathbf{f}\left(\mathscr{Y}_q\right)\right)\right] - C_2\mathbb{E}\left[\mathrm{sim}\left(\mathbf{f}\left(\mathscr{X}_p\right),\mathbf{f}\left(\mathscr{Y}_q\right)\right) \mid p=q\right] \\ &= C_2\mathscr{L}_{ss}^{\Theta}\left(\frac{\lambda-1}{C_2}\right).\end{aligned} \tag{19}$$

From Eqn. (16) to Eqn. (17), we use the fact that "$\mathbb{E}[\cdot]$ means the expectation is taken over four interdependent random variables, i.e., $p, q, \mathscr{X}_p, \mathscr{Y}_q$, which is mentioned in Section 3.1. From Eqn. (18) to Eqn. (19), we use the fact that given p, the similarity of random pairs is smaller than the similarity of positive pairs $\mathbb{E}_{q,\mathscr{X}_p,\mathscr{Y}_q}[\mathrm{sim}\left(\mathbf{f}\left(\mathscr{X}_p\right),\mathbf{f}\left(\mathscr{Y}_q\right)\right)] \le \mathbb{E}_{q,\mathscr{X}_p,\mathscr{X}_q}[\mathrm{sim}\left(\mathbf{f}\left(\mathscr{X}_p\right),\mathbf{f}\left(\mathscr{Y}_q\right)\right) \mid p=q]$. The upper bound is derived by replacing $\frac{\lambda r_p}{1-r_p}+1, \forall p$ with $C_2 = 1+\min_p \frac{\lambda r_p}{1-r_p}$. Similarly, we can also derive the other side (lower bound) by using $C_1 = 1+\max_p \frac{\lambda r_p}{1-r_p}$, which eventually gives $C_1\mathscr{L}_{ss}^{\Theta}\left(\frac{\lambda-1}{C_1}\right)$.

References

Alday EAP, Gu A, Shah AJ, Robichaux C, Wong A-KI, Liu C, Liu F, Rad AB, Elola A, Seyedi S, et al. (2020). Classification of 12-lead ecgs: the physionet/computing in cardiology challenge 2020. Physiological measurement, 41(12):124003.

Anguita D, Ghio A, Oneto L, Parra X, and Reyes-Ortiz JL (2013). A public domain dataset for human activity recognition using smartphones. In Esann, volume 3, page 3.

Arora S, Khandeparkar H, Khodak M, Plevrakis O, and Saunshi N (2019). A theoretical analysis of contrastive unsupervised representation learning. arXiv preprint arXiv:1902.09229.

Biswal S, Sun H, Goparaju B, Westover MB, Sun J, and Bianchi MT (2018). Expert-level sleep scoring with deep neural networks. Journal of the American Medical Informatics Association, 25(12):1643–1650. [PubMed: 30445569]

Cai D, He X, Han J, and Huang TS (2010). Graph regularized nonnegative matrix factorization for data representation. IEEE transactions on pattern analysis and machine intelligence, 33(8):1548–1560. [PubMed: 21173440]

Cao Y, Das S, Oeding L, and van Wyk H-W (2020). Analysis of the stochastic alternating least squares method for the decomposition of random tensors. arXiv preprint arXiv:2004.12530.

Chen T, Kornblith S, Norouzi M, and Hinton G (2020). A simple framework for contrastive learning of visual representations. In International conference on machine learning, pages 1597–1607. PMLR.

Cheng JY, Goh H, Dogrusoz K, Tuzel O, and Azemi E (2020). Subject-aware contrastive learning for biosignals. arXiv preprint arXiv:2007.04871.

Chuang C-Y, Robinson J, Yen-Chen L, Torralba A, and Jegelka S (2020). Debiased contrastive learning. arXiv:2007.00224.

Cong F, Lin Q-H, Kuang L-D, Gong X-F, Astikainen P, and Ristaniemi T (2015). Tensor decomposition of EEG signals: a brief review. Journal of neuroscience methods, 248:59–69. [PubMed: 25840362]

Dao T, Gu A, Ratner A, Smith V, De Sa C, and Ré C (2019). A kernel theory of modern data augmentation. In International Conference on Machine Learning, pages 1528–1537. PMLR.

Golub GH and Von Matt U (1997). Tikhonov regularization for large scale problems. Citeseer.

Grill J-B, Strub F, Altché F, Tallec C, Richemond PH, Buchatskaya E, Doersch C, Pires BA, Guo ZD, Azar MG, et al. (2020). Bootstrap your own latent: A new approach to self-supervised learning. arXiv:2006.07733.

Gutmann M and Hyvärinen A (2010). Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In Proceedings of the thirteenth international conference on artificial intelligence and statistics, pages 297–304. JMLR Workshop and Conference Proceedings.

Hamdi SM, Wu Y, Boubrahimi SF, Angryk R, Krishnamurthy LC, and Morris R (2018). Tensor decomposition for neurodevelopmental disorder prediction. In International Conference on Brain Informatics, pages 339–348. Springer.

He K, Fan H, Wu Y, Xie S, and Girshick R (2020). Momentum contrast for unsupervised visual representation learning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 9729–9738.

Hong D, Kolda TG, and Duersch JA (2020). Generalized canonical polyadic tensor decomposition. SIAM Review, 62(1):133–163.

Jia C, Zhong G, and Fu Y (2014). Low-rank tensor learning with discriminant analysis for action classification and image recovery. In Twenty-eighth AAAI conference on artificial intelligence.

Kemp B, Zwinderman AH, Tuk B, Kamphuisen HA, and Oberye JJ (2000). Analysis of a sleep-dependent neuronal feedback loop: the slow-wave microcontinuity of the eeg. IEEE Transactions on Biomedical Engineering, 47(9):1185–1194. [PubMed: 11008419]

Kolda TG and Bader BW (2009). Tensor decompositions and applications. SIAM review, 51(3):455–500.

Lu H, Plataniotis KN, and Venetsanopoulos AN (2008). Uncorrelated multilinear discriminant analysis with regularization and aggregation for tensor object recognition. IEEE Transactions on Neural Networks, 20(1):103–123. [PubMed: 19095532]

Maki H, Tanaka H, Sakti S, and Nakamura S (2018). Graph regularized tensor factorization for single-trial eeg analysis. In ICASSP, pages 846–850. IEEE.

Sidiropoulos ND, De Lathauwer L, Fu X, Huang K, Papalexakis EE, and Faloutsos C (2017). Tensor decomposition for signal processing and machine learning. IEEE Transactions on Signal Processing, 65(13):3551–3582.

Singh N, Zhang Z, Wu X, Zhang N, Zhang S, and Solomonik E (2021). Distributed-memory tensor completion for generalized loss functions in python using new sparse tensor kernelsate randomized algorithms for low-rank tensor decompositions. arXiv preprint arXiv:1910.02371.

Tao D, Li X, Hu W, Maybank S, and Wu X (2005). Supervised tensor learning. In Fifth IEEE International Conference on Data Mining (ICDM'05), pages 8-pp. IEEE.

Tao D, Li X, Wu X, and Maybank SJ (2007). General tensor discriminant analysis and gabor features for gait recognition. IEEE transactions on pattern analysis and machine intelligence, 29(10):1700–1715. [PubMed: 17699917]

Wang T and Isola P (2020). Understanding contrastive representation learning through alignment and uniformity on the hypersphere. In International Conference on Machine Learning, pages 9929–9939. PMLR.

Wang Y, Peng J, Zhao Q, Leung Y, Zhao X-L, and Meng D (2017). Hyperspectral image restoration via total variation regularized low-rank tensor decomposition. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 11(4):1227–1243.

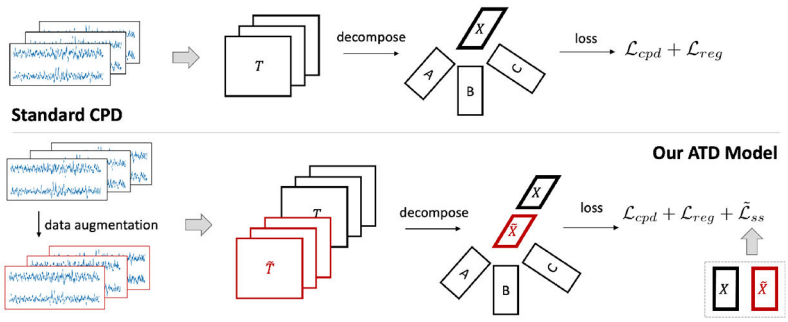

Figure 1:
Standard CPD vs Our ATD Model

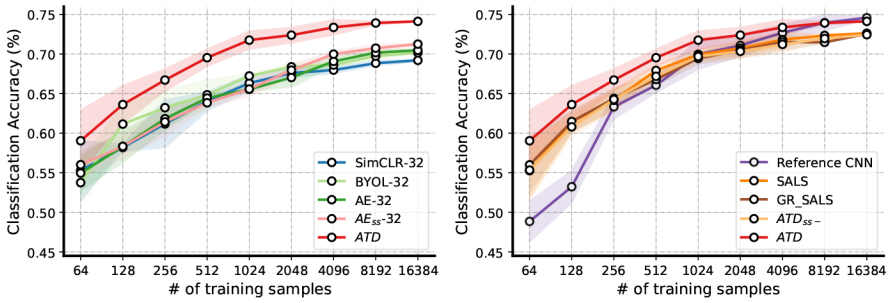

Figure 2:
Varying the # of Training Data

Table 1:

Dataset Statistics

Name	Data Sample Format	Augmentations	# Unlabeled (N)	# Training	# Test	Task	# Class
Sleep-EDF	$I \times J \times K$: 14 × 129 × 86	(a), (b)	331,208	42,803	41,078	Sleep Staging	5
HAR	$I \times J \times K$: 18 × 33 × 33	(a), (b), (c)	7,352	1,473	1,474	Activity Recognition	6
PTB-XL	$I \times J \times K$: 24 × 129 × 75	(a), (b)	17,469	2,183	2,185	Gender Identification	2
MGH	$I \times J \times K$: 12 × 257 × 43	(a), (b)	4,377,170	238,312	248,041	Sleep Staging	5

Table 2:

Result of Downstream Classification (%). The table shows that our ATD can provide comparable or better performance over all baselines with fewer parameters, especially deep learning models. It also shows the usefulness of considering both *fitness* and *alignment* as part of the objective.

	Sleep-EDF (5,000)		HAR (1,473)		PTB-XL (2,183)		MGH (5,000)	
	Accuracy	**# of Params.**	**Accuracy**	**# of Params.**	**Accuracy**	**# of Params.**	**Accuracy**	**# of Params.**
Self-sup models:								
SimCLR-32	84.98 ± 0.358	210,384	74.75 ± 0.723	53,286	69.25 ± 0.355	200,960	67.34 ± 0.970	212,624
SimCLR-128	**85.19** ± **0.358**	222,768	76.69 ± 0.697	65,670	68.19 ± 0.793	237,920	66.98 ± 1.331	246,608
BYOL-32	84.29 ± 0.405	211,440	73.71 ± 2.832	54,342	65.08 ± 1.535	202,016	68.83 ± 1.168	214,736
BYOL-128	83.26 ± 0.337	239,280	71.79 ± 1.866	82,182	65.49 ± 0.612	254,432	68.55 ± 1.339	279,632
Auto-encoders:								
AE-32	74.78 ± 0.723	217,216	63.13 ± 0.775	62,940	59.01 ± 0.896	224,528	68.58 ± 0.427	220,088
AE-128	75.17 ± 0.897	241,888	60.52 ± 1.604	87,612	58.29 ± 0.412	298,352	67.05 ± 1.375	257,048
AE_{ss}-32	80.92 ± 0.345	217,216	71.70 ± 2.135	62,940	68.47 ± 0.231	224,528	71.46 ± 0.386	220,088
AE_{ss}-128	81.84 ± 0.259	241,888	72.43 ± 1.370	87,612	68.88 ± 0.604	298,352	70.19 ± 0.617	257,048
Tensor models:								
SALS	84.27 ± 0.481	7,328	91.86 ± 0.295	2,688	69.15 ± 0.483	7,296	71.93 ± 0.379	9,984
GR-SALS	84.33 ± 0.356	7,328	92.33 ± 0.282	2,688	69.02 ± 0.477	7,296	72.35 ± 0.228	9,984
ATD_{ss-}	84.19 ± 0.221	7,328	92.41 ± 0.391	2,688	69.38 ± 0.612	7,296	72.78 ± 0.522	9,984
ATD	85.01 ± 0.224	7,328	**93.35** ± **0.357**	2,688	**70.26** ± **0.523**	7,296	**74.15** ± **0.431**	9,984

*Parenthesis shows the number of training samples. Our improvements are statistically significant with $p < 0.05$ (details in appendix D.7).