4-2014

Turbo-SMT: Accelerating Coupled Sparse Matrix-Tensor Factorizations by 200x

Evangelos E. Papalexakis
Carnegie Mellon University

Christos Faloutsos
Carnegie Mellon University, christos@cs.cmu.edu

Tom Mitchell
Carnegie Mellon University, mitchell@andrew.cmu.edu

Partha Pratim Talukdar
Carnegie Mellon University

Nicholas D. Sidiropoulos
University of Minnesota

See next page for additional authors

Published In
Proceedings of the 2014 SIAM International Conference on Data Mining, 118-126.

Turbo-SMT: Accelerating Coupled Sparse Matrix-Tensor Factorizations by 200x

Evangelos E. Papalexakis* epapalex@cs.cmu.edu
Tom M. Mitchell * tom.mitchell@cmu.edu
Nicholas D. Sidiropoulos † nikos@ece.umn.edu
Christos Faloutsos * christos@cs.cmu.edu
Partha Pratim Talukdar * partha.talukdar@cs.cmu.edu
Brian Murphy ‡ brian.murphy@qub.ac.uk

Abstract

How can we correlate the neural activity in the human brain as it responds to typed words, with properties of these terms (like 'edible', 'fits in hand')? In short, we want to find latent variables, that jointly explain both the brain activity, as well as the behavioral responses. This is one of many settings of the *Coupled Matrix-Tensor Factorization* (CMTF) problem.

Can we accelerate *any* CMTF solver, so that it runs within a few minutes instead of tens of hours to a day, while maintaining good accuracy? We introduce Turbo-SMT, a meta-method capable of doing exactly that: it boosts the performance of *any* CMTF algorithm, by up to *200x*, along with an up to *65 fold* increase in sparsity, with comparable accuracy to the baseline.

We apply Turbo-SMT to BrainQ, a dataset consisting of a (nouns, brain voxels, human subjects) tensor and a (nouns, properties) matrix, with coupling along the nouns dimension. Turbo-SMT is able to find meaningful latent variables, as well as to predict brain activity with competitive accuracy.

1 Introduction

How is knowledge mapped and stored in the human brain? How is it expressed by people answering simple questions about specific words? If we have data from both worlds, are we able to combine them and jointly analyze them? In a very different scenario, suppose we have the social network graph of an online social network, and we also have additional information about how and when users interacted with each other. What is a comprehensive way to combine those two pieces of data? Both, seemingly different, problems may be viewed as instances of what is called *Coupled Matrix-Tensor Factorization* (CMTF), where a data tensor and matrices that hold additional information are jointly decomposed into a set of low-rank factors.

Applications that fall within the CMTF formulation usually span many gigabytes of data. Additionally, current state of the art approaches operate very slowly, even on moderately large datasets. In this work, we introduce Turbo-SMT, a fast, scalable, and sparsity promoting CMTF meta-algorithm. Our main contributions are the following:

- *Fast, parallel & triple-sparse algorithm:* We provide an approximate, novel, scalable, and triple-sparse (see Sec. 3) meta-method, Turbo-SMT, that is able to accelerate *any* CMTF core algorithm.
- *Effectiveness & Knowledge Discovery:* We analyze BrainQ, a brain scan dataset which is coupled to a semantic matrix (see Sec. 4 for details).
- *Reproducibility:* Our code is publicly available [1] ; the BrainQ dataset we use (see Section 4) is also publicly available.

Figure 1 shows the accuracy of Turbo-SMT (compared to the baseline), as a function of portion of the wall-clock time that our algorithm took, again compared to the traditional one. The result indicates a speedup of up to 200 times, while maintaining very good accuracy.[2] In the supplementary material [2], we show how Turbo-SMT can handle missing values.

In the knowledge discovery part, the brain scan part of the dataset consists of fMRI scans first used in [11], a work that first demonstrated that brain activity can be predictably analyzed into component semantic features. Here, we demonstrate a disciplined way to combine both datasets and carry out a variety of data mining/machine learning tasks, through this joint analysis. In the supplementary material [2], we apply Turbo-SMT to

*Carnegie Mellon University
†University of Minnesota
‡Queen's University of Belfast

[1] www.cs.cmu.edu/~epapalex/src/turbo_smt.zip

[2] Accuracy or relative cost is defined in Section 5 as the ratio of the squared approximation error of Turbo-SMT, divided by that of the baseline CMTF solver.

a time-evolving social network with side information, discovering anomalies.

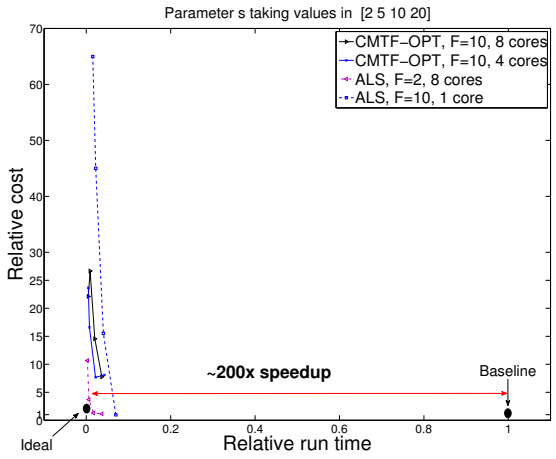

Figure 1: TURBO-SMT is up to 200x faster, for comparable accuracy. Relative execution time vs relative cost (lower is better), for various settings (see Sec. 3. TURBO-SMT boosting either ALS or CMTF-OPT [5]) (i.e. the baselines) vs. plain execution of the baselines (on a single core), on the BRAINQ dataset (see Section 4.). For more details, see Section 5.

Disclaimer: An earlier version of this work was uploaded on Arxiv.org [16] for quick dissemination thereof and early feedback. However, the Arxiv.org version has not been officially published in any conference proceedings or journal, and has evolved into this present work.

2 Preliminaries

Symbol	Description
CMTF	Coupled Matrix-Tensor Factorization
ALS	Alternating Least Squares
CMTF-OPT	Algorithm introduced in [5]
$x, \mathbf{x}, \mathbf{X}, \underline{\mathbf{X}}$	scalar, vector, matrix, tensor (respectively)
$\|\mathbf{A}\|_F$	Frobenius norm of $\mathbf{A}$.
(i) as superscript	Indicates the i-th iteration
$\mathbf{A}_1^i$, $\mathbf{a}_1^i$	series of matrices or vectors, indexed by i.
$\mathcal{I}$	Set of indices.
$\mathbf{x}(\mathcal{I})$	Spanning indices $\mathcal{I}$ of $\mathbf{x}$.
nucleus	a biased random sample of the data tensor.

Table 1: Table of symbols

2.1 Introduction to Tensors Matrices record dyadic properties, like "people recommending products". Tensors are the n-mode generalizations, capturing 3- and higher-way relationships. For example "subject-verb-object" relationships, such as the ones recorded by the Read the Web - NELL project [1] (and

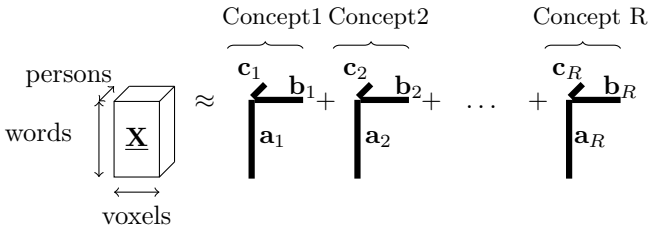

Figure 2: PARAFAC decomposition of a three-way tensor of a brain activity tensor as sum of F outer products (rank-one tensors), reminiscent of the rank-F singular value decomposition of a matrix. Each component corresponds to a **latent** concept of, e.g. "insects", "tools" and so on, a set of brain regions that are most active for that particular set of words, as well as groups of persons.

have been recently used in this context [9] [15]) naturally lead to a 3-mode tensor. In this work, our working example of a tensor has three modes. The first mode contains a number of nouns; the second mode corresponds to the brain activity, as recorded by an fMRI machine; and the third mode identifies the human subject corresponding to a particular brain activity measurement.

In [15], the authors introduced a scalable and parallelizable tensor decomposition which uses mode sampling. In this work, we focus on a more general and expressive framework, that of *Coupled Matrix-Tensor Factorizations*.

2.2 Coupled Matrix-Tensor Factorization Oftentimes, two tensors, or a matrix and a tensor, may have one mode in common; consider the example that we mentioned earlier, where we have a word by brain activity by human subject tensor, we also have a semantic matrix that provides additional information for the same set of words. In this case, we say that the matrix and the tensor are *coupled* in the 'word' mode.

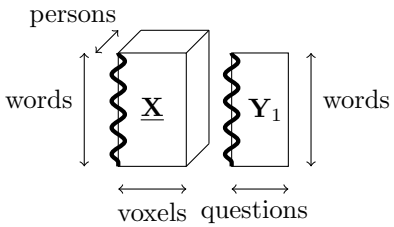

Figure 3: *Coupled Matrix - Tensor* example: Tensors often share one or more modes (with thick, wavy line): $\underline{\mathbf{X}}$ is the brain activity tensor and $\mathbf{Y}$ is the semantic matrix. As the wavy line indicates, these two datasets are coupled in the 'word' dimension.

In this work we focus on three mode tensors,

however, everything we mention extends directly to higher modes. In the general case, a three mode tensor $\underline{\mathbf{X}}$ may be coupled with at most three matrices $\mathbf{Y}_i,\ i = 1 \cdots 3$, in the manner illustrated in Figure 3 for one mode. The optimization function that encodes this decomposition is:

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{D},\mathbf{E},\mathbf{G}} \|\underline{\mathbf{X}} - \sum_k \mathbf{a}_k \circ \mathbf{b}_k \circ \mathbf{c}_k\|_F^2 + \|\mathbf{Y}_1 - \mathbf{A}\mathbf{D}^T\|_F^2 + \|\mathbf{Y}_2 - \mathbf{B}\mathbf{E}^T\|_F^2 + \|\mathbf{Y}_3 - \mathbf{C}\mathbf{G}^T\|_F^2 \quad (2.1)$$

where $\mathbf{a}_k$ is the k-th column of $\mathbf{A}$. The idea behind the coupled matrix-tensor decomposition is that we seek to jointly analyze $\underline{\mathbf{X}}$ and $\mathbf{Y}_i$, decomposing them to latent factors who are coupled in the shared dimension. For instance, the first mode of $\underline{\mathbf{X}}$ shares the same low rank column subspace as $\mathbf{Y}_1$; this is expressed through the latent factor matrix $\mathbf{A}$ which jointly provides a basis for that subspace.

2.3 Solving CMTF One of the most popular algorithms to solve CMTF (as introduced in Figure 3) is the so-called Alternating Least Squares (ALS), a block-coordinate descent method; we present an outline of the ALS algorithm in the supplementary material [2]. Besides ALS, there exist other algorithms for CMTF. For example, [5] uses a first order optimization algorithm for the same objective. In their paper [5], the authors demonstrate that their algorithm, which we henceforth refer to as CMTF-OPT, often converges to a better solution than ALS; Throughout this work, we use both algorithms as core CMTF solvers for TURBO-SMT. The strength of TURBO-SMT, however, is that it can be used as-is with any underlying core CMTF implementation. Hence, TURBO-SMT is *CMTF solver independent.*

3 Proposed Method

3.1 Algorithm description There are three main concepts behind TURBO-SMT (outlined in Algorithm 1):

Phase 1 Obtain a *nucleus* of our data by using biased sampling.
Phase 2 Fit CMTF to the reduced data (possibly on more than one *nuclei*)
Phase 3 stitch the partial results

Phase1: Sampling An efficient way to reduce the size of the dataset, yet operate on a representative subset thereof is to use *biased* sampling. In particular, given a three-mode tensor $\underline{\mathbf{X}}$ we sample as follows. We calculate three vectors as shown in equation (3.2), one for each mode of $\underline{\mathbf{X}}$. These vectors, which we henceforth refer to as *density vectors* are the marginal absolute sums with respect to all but one of the modes of the tensor, and in essence represent the importance of each index of the respective mode. We then sample *indices* of each mode according to the respective density vector. For instance, assume an $I \times J \times K$ tensor; suppose that we need a sample of size $\frac{I}{s}$ of the indices of the first mode. Then, we just define $p_{\mathcal{I}}(i) = \mathbf{x}_A(i) / \sum_{i=1}^{I} \mathbf{x}_A(i)$ as the probability of sampling the i-th index of the first mode, and we simply sample without replacement from the set $\{1 \cdots I\}$, using $p_{\mathcal{I}}$ as bias. The very same idea is used for matrices $\mathbf{Y}_i$. Doing so is preferable over sampling uniformly, since our bias makes it more probable that high density indices of the data will be retained on the sample, and hence, it will be more representative of the entire set.

Suppose that we call $\mathcal{I}, \mathcal{J}, \mathcal{K}$ the index samples for the three modes of $\underline{\mathbf{X}}$. Then, we may take $\underline{\mathbf{X}}_s = \underline{\mathbf{X}}(\mathcal{I}, \mathcal{J}, \mathcal{K})$ (and similarly for matrices $\mathbf{Y}_i$); essentially, what we are left with is a small, yet representative, sample of our original dataset, where the high density blocks are more likely to appear on the sample. It is important to note that the indices of the coupled modes are the same for the matrix and the tensor, e.g. $\mathbf{I}$ randomly selects the same set of indices for $\underline{\mathbf{X}}$ and $\mathbf{Y}_1$. This way, we make sure that the coupling is *preserved* after sampling.

Finally, Phase 1 can be executed very efficiently, since both the calculation of sample biases, as well as the sampling of indices require only 2 passes on the non-zero elements of the (usually, highly sparse) data.

Phase 2: Fit CMTF to *nuclei* The next step of TURBO-SMT is to fit a CMTF model to each *nucleus*, and then, based on the sampled indices, redistribute the result to the original index space. As we have already discussed, TURBO-SMT is not restricted in any way to a specific CMTF solver; in fact, we provide experiments using both an ALS and a Gradient Descent approach.. In more detail, suppose that $\mathbf{A}_s$ is the factor matrix obtained by the aforementioned procedure, and that jointly describes the first mode of $\underline{\mathbf{X}}_s$ and $\mathbf{Y}_{1,s}$. The dimensions of $\mathbf{A}_s$ are going to be $|\mathcal{I}| \times F$ (where $|\cdot|$ denotes cardinality and F is the number of factors). Let us further assume matrix $\mathbf{A}$ of size $I \times F$ which expresses the first mode of the tensor and the matrix, before sampling; due to sampling, it holds that $I \gg |\mathcal{I}|$. If we initially set all entries of $\mathbf{A}$ to zero and we further set $\mathbf{A}(\mathcal{I}, :) = \mathbf{A}_s$ we obtain a highly *sparse* factor matrix whose non-zero values are a 'best effort' approximation of the true ones, i.e. the values of the factor matrix that we would obtain by decomposing the full data.

So far, we have provided a description of the algorithm where only one repetition of sampling is used.

(3.2)
$$\mathbf{x}_A(i) = \sum_{j=1}^{J}\sum_{k=1}^{K}|\underline{\mathbf{X}}(i,j,k)| + \sum_{j=1}^{I_1}|\mathbf{Y}_1(i,j)|, \quad \mathbf{x}_B(j) = \sum_{i=1}^{I}\sum_{k=1}^{K}|\underline{\mathbf{X}}(i,j,k)| + \sum_{i=1}^{I_2}|\mathbf{Y}_2(j,i)|, \quad \mathbf{x}_C(k) = \sum_{i=1}^{I}\sum_{j=1}^{J}|\underline{\mathbf{X}}(i,j,k)| + \sum_{j=1}^{I_3}|\mathbf{Y}_3(k,j)|,$$

(3.3)
$$\mathbf{y}_{1,A}(i) = \sum_{j=1}^{I_1}|\mathbf{Y}_1(i,j)| \quad \mathbf{y}_{2,B}(j) = \sum_{i=1}^{I_2}|\mathbf{Y}_2(j,i)|, \quad \mathbf{y}_{3,C}(k) = \sum_{j=1}^{I_3}|\mathbf{Y}_3(k,j)|$$

(3.4)
$$\mathbf{y}_{1,D}(j) = \sum_{i=1}^{I}|\mathbf{Y}_1(i,j)|, \quad \mathbf{y}_{2,G}(i) = \sum_{j=1}^{J}|\mathbf{Y}_2(j,i)|, \quad \mathbf{y}_{3,E}(i) = \sum_{k=1}^{K}|\mathbf{Y}_3(k,i)|$$

However, the approximation quality of TURBO-SMT improves as we increase the number of *nuclei*. To that end, we allow for multiple sampling repetitions in our algorithm, i.e. extracting multiple sample tensors $\underline{\mathbf{X}}_s$ and side matrices $\mathbf{Y}_{i,s}$, fitting a CMTF model to all of them and combining the results in a way that the true latent patterns are retained. We are going to provide a detailed outline of how to carry the multi-repetition version of TURBO-SMT in the following.

While doing multiple repetitions, we keep a *common.* subset of indices for all different samples. In particular, let p be the percentage of common values across all repetitions and $\mathcal{I}_p$ denote the common indices along the first mode (same notation applies to the rest of the indices); then, all sample tensors $\underline{\mathbf{X}}_s$ will definitely contain the indices $\mathcal{I}_p$ on the first mode, as well as $(1-p)\frac{I}{s}$ indices sampled independently (across repetitions) at random. This common index sample is key in order to ensure that our results are not rank deficient, and all partial results are merged correctly.

We do not provide an exact method for choosing p, however, as a rule of thumb, we observed that, depending on how sparse and noisy the data is, a range of p between 0.2 and 0.5 works well. This introduces a trade-off between redundancy of indices that we sample, versus the accuracy of the decomposition; since we are not dealing solely with tensors, which are known to be relatively more well behaved in terms of decomposition uniqueness (in contrast to matrices), it pays off to introduce some data redundancy (especially when TURBO-SMT runs in a parallel system) so that we avoid rank-deficiency in our data.

Let r be the number of different sampling repetitions, resulting in r different sets of sampled matrix-tensor couples $\underline{\mathbf{X}}_s^{(i)}$ and $\mathbf{Y}_{j,s}^{(i)}$ $(i = 1\cdots r,\ j = 1\cdots 3)$. For that set of coupled data, we fit a CMTF model, using a CMTF solver, obtaining a set of factor matrices $\mathbf{A}^{(i)}$ (and likewise for the rest).

Phase 3: Stitching partial results After having obtained these r different sets of partial results, as a final step, we have to merge them together into a set of factor matrices that we would ideally get had we operated on the full dataset.

In order to make the merging work, we first introduce the following scaling on each column of each factor matrix: Let's take $\mathbf{A}^{(i)}$ for example; we normalize each column of $\mathbf{A}$ by the ℓ_2 norm of the common part, as described in line 8 of Algorithm 1. By doing so, the common part of each factor matrix (for all repetitions) will be unit norm. This scaling is absorbed in a set of scaling vectors $\boldsymbol{\lambda}_A$ (and accordingly for the rest of the factors). The new objective function is shown in Equation 3.5

(3.5)
$$\begin{aligned}\min_{\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{D},\mathbf{E},\mathbf{G}} &\|\underline{\mathbf{X}} - \sum_k \boldsymbol{\lambda}_A(k)\boldsymbol{\lambda}_B(k)\boldsymbol{\lambda}_C(k)\mathbf{a}_k \circ \mathbf{b}_k \circ \mathbf{c}_k\|_F^2 \\ &+ \|\mathbf{Y}_1 - \mathbf{A}\ \mathrm{diag}(\boldsymbol{\lambda}_A * \boldsymbol{\lambda}_D)\ \mathbf{D}^T\|_F^2 \\ &+ \|\mathbf{Y}_2 - \mathbf{B}\ \mathrm{diag}(\boldsymbol{\lambda}_B * \boldsymbol{\lambda}_E)\ \mathbf{E}^T\|_F^2 \\ &+ \|\mathbf{Y}_3 - \mathbf{C}\ \mathrm{diag}(\boldsymbol{\lambda}_C * \boldsymbol{\lambda}_G)\ \mathbf{G}^T\|_F^2\end{aligned}$$

A problem that is introduced by carrying out multiple sampling repetitions is that the correspondence of the output factors of each repetition is very likely to be distorted. In other words, say we have matrices $\mathbf{A}^{(1)}$ and $\mathbf{A}^{(2)}$ and we wish to merge their columns (i.e. the latent components) into a single matrix $\mathbf{A}$, by stitching together columns that correspond to the same component. It might very well be the case that the order in which the latent components appear in $\mathbf{A}^{(1)}$ is not the same as in $\mathbf{A}^{(2)}$.

The sole purpose of the aforementioned normalization is to resolve the correspondence problem. In Algorithm 2, we merge the partial results while establishing the correct correspondence of the columns.

LEMMA 3.1. *Algorithm 2 is able to find the correct correspondence of columns.*

Proof Sketch *Following the example of $r = 2$ of the previous paragraph, according to Algorithm 2, we compute the inner product of the common parts of each column of $\mathbf{A}^{(1)}$ and $\mathbf{A}^{(2)}$. Since the common parts of*

each column are normalized to unit norm, then the inner product of the common part of the column of $\mathbf{A}^{(1)}$ *with that of* $\mathbf{A}^{(2)}$ *will be maximized (and exactly equal to 1) for the matching columns, and by the Cauchy-Schwartz inequality, for all other combinations, it will be less than 1. Additionally, elimination of the already used columns operates as a tie-breaker.*

Finally, Phase 3, due to its low complexity, can be executed very efficiently. In particular, the StitchFactors algorithm requires $O(rF^2)$ steps, where, both r and F are, for most practical cases, very small, compared to the data dimensionality.

3.2 Sparsity through Sampling Besides data size reduction, one merit of sampling is sparsity on the latent factors. Every time Turbo-SMT does one repetition, it operates on a sub-sampled version of the data. Consequently, in the third phase of the algorithm, where the results are re-distributed to their indices in the original, high dimensional space, most of the indices of the latent factors are going to be exactly zero, thus resulting in latent factor sparsity. In this way, Turbo-SMT always operates on a sparse set of data, through the entire lifetime of the algorithm, a thing which is not true for the majority of the algorithms both for tensor and coupled decompositions, which usually operate on dense factors (even when the final output is sparse), and have very high and unnecessary storage needs.

Definition 3.1. (Triple-sparse) *An algorithm is triple-sparse when 1) the input of the algorithm is sparse, 2) the intermediate data during the lifetime of the algorithm is sparse, and 3) the final output of the algorithm is sparse.*

In the above definition, the input of the algorithm need not necessarily be sparse; however, a triple-sparse algorithm still satisfies the second and third requirement, by operating on a sparse, representative subset of the data. We, thus, call Turbo-SMT, a *triple-sparse* algorithm.

3.3 Parallelization Turbo-SMT is, by its nature, parallelizable; in essence, we generate multiple samples of the coupled data, we fit a CMTF model to each sample and then we merge the results. By carefully observing Algorithm 1, we can see that lines 3 to 9 may be carried out entirely in parallel, provided that we have a good enough random number generator that does not generate the very same sample across all r repetitions. In particular, the r repetitions are independent from one another, since computing the set of common indices (line 2), which is the common factor across all repetitions, is done before line 3.

Algorithm 1: Turbo-SMT: Fast, sparse, and parallel CMTF

Input: Tensor $\underline{\mathbf{X}}$ of size $I \times J \times K$, matrices $\mathbf{Y}_i$, $i = 1 \cdots 3$, of size $I \times I_2$, $J \times J_2$, and $K \times K_2$ respectively, number of factors F, sampling factor s, number of repetitions r.

Output: $\mathbf{A}$ of size $I \times F$, $\mathbf{b}$ of size $J \times F$, $\mathbf{c}$ of size $K \times F$, $\mathbf{D}$ of size $I_2 \times F$, $\mathbf{G}$ of size $J_2 \times F$, $\mathbf{E}$ of size $K_2 \times F$. $\boldsymbol{\lambda}_A$, $\boldsymbol{\lambda}_B$, $\boldsymbol{\lambda}_C$, $\boldsymbol{\lambda}_D$, $\boldsymbol{\lambda}_E$, $\boldsymbol{\lambda}_G$ of size $F \times 1$.

1: Initialize $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}, \mathbf{E}, \mathbf{G}$ to all-zeros.

2: Randomly, *using mode densities as bias*, select a set of $100p\%$ ($p \in [0, 1]$) indices $\mathcal{I}_p, \mathcal{J}_p, \mathcal{K}_p$ to be common across all repetitions. For example, $\mathcal{I}_p$ is sampled with probabilities with $p_{\mathcal{I}}(i) = \mathbf{x}_A(i) / \sum_{i=1}^{I} \mathbf{x}_A(i)$. Probabilities for the rest of the modes are calculated similarly.

3: **for** $i = 1 \cdots r$ **do**

{**Phase 1: Obtain *nuclei* through biased sampling**}

4: Compute densities as in equations 3.2, 3.3, 3.4. Compute set of indices $\mathcal{I}^{(i)}$ as random sample without replacement of $\{1 \cdots I\}$ of size $I/(s(1-p))$ with probability $p_{\mathcal{I}}(i) = \mathbf{x}_A(i) / \sum_{i=1}^{I} \mathbf{x}_a(i)$. Likewise for $\mathcal{J}, \mathcal{K},$, $\mathcal{I}_1$, $\mathcal{I}_2$, and $\mathcal{I}_3$. Set $\mathcal{I}^{(i)} = \mathcal{I} \cup \mathcal{I}_p$. Likewise for the rest.

5: Get *nucleus* $\underline{\mathbf{X}}_s^{(i)} = \underline{\mathbf{X}}(\mathcal{I}^{(i)}, \mathcal{J}^{(i)}, \mathcal{K}^{(i)})$, $\mathbf{Y}_{1s}^{(i)} = \mathbf{Y}_1(\mathcal{I}^{(i)}, \mathcal{I}_1^{(i)})$ and likewise for $\mathbf{Y}_{2s}^{(i)}$ and $\mathbf{Y}_{3s}^{(i)}$. Note that the same index sample is used for *coupled* modes.

{**Phase 2: Fit the model on each *nucleus***}

6: Run a CMTF solver for $\underline{\mathbf{X}}_s^{(i)}$ and $\mathbf{Y}_{js}^{(i)}$, $j = 1 \cdots 3$ and obtain $\mathbf{A}_s, \mathbf{B}_s, \mathbf{C}_s, \mathbf{D}_s, \mathbf{G}_s, \mathbf{E}_s$.

7: $\mathbf{A}^{(i)}(\mathcal{I}^{(i)}, :) = \mathbf{A}_s$. Likewise for the rest.

8: Calculate the ℓ_2 norm of the columns of the common part: $\boldsymbol{\lambda}_A^{(i)}(f) = \|\mathbf{A}^{(i)}(\mathcal{I}_p, f)\|_2$, for $f = 1 \cdots F$. Normalize columns of $\mathbf{A}^{(i)}$ using $\boldsymbol{\lambda}_A^{(i)}$ (likewise for the rest). Note that the common part of each factor will now be normalized to unit norm.

9: **end for**

{**Phase 3: Stitch partial results**}

10: $\mathbf{A}$ =StitchFactors ($\mathbf{A}_1^i$). Likewise for the rest.

11: $\boldsymbol{\lambda}_A$ = average of $\boldsymbol{\lambda}_A 1^i$. Likewise for the rest.

4 Knowledge Discovery

Turbo-SMT on Brain Image Data With Additional Semantic Information

As part of a larger study of neural representations of word meanings in the human brain [11], we applied Turbo-SMT to a combination of datasets which we henceforth jointly refer to as BrainQ. This dataset consists of two parts. The first is a tensor that contains measurements of the fMRI brain activity of 9

Algorithm 2: StitchFactors: Given partial results of factor matrices, merge them correctly

Input: Factor matrices $\mathbf{A}_1^i$ of size $I \times F$ each, and r is the number of repetitions, $\mathcal{I}_p$: set of common indices.
Output: Factor matrix $\mathbf{A}$ of size $I \times F$.

```
Set A = A^(1)
Set ℓ = {1 ··· F}, a list that keeps track of which columns have not been assigned yet.
for i = 2 ··· r do
    for f1 = 1 ··· F do
        for f2 in ℓ do
            Compute similarity
            v(f2) = (A(I_p, f2))^T ( A^(i)(I_p, f1)) )
        end for
        c* = arg max_c v(c) (Ideally, for the matching columns, the inner product should be close to 1; conversely, for the rest of the columns, it should be considerably smaller)
        A(:, c*) = A^(i)(:, f1)|_{A(:,c*)=0}, i.e. update the zero entries of the column.
        Remove c* from list ℓ.
    end for
end for
```

human subjects, when shown each of 60 concrete nouns (5 in each of 12 categories, e.g. dog, hand, house, door, shirt, dresser, butterfly, knife, telephone, saw, lettuce, train). fMRI measures slow changes in blood oxygenation levels, reflecting localized changes in brain activity. Here our data is made up of $3 \times 3 \times 6$mm voxels (3D pixels) corresponding to fixed spatial locations across participants. Recorded fMRI values are the mean activity over 4 contiguous seconds, averaged over multiple presentations of each stimulus word (each word is presented 6 times as a stimulus). Further acquisition and preprocessing details are given in [11]. This dataset is publicly available[3]. The second part of the data is a matrix containing answers to 218 questions pertaining to the semantics of these 60 nouns. A sample of these questions is shown in Fig. 4. This dataset has been used before in works such as [13], [14].

BrainQ's size is $60 \times 77775 \times 9$ with over 11 million non-zeros (tensor), and 60×218 with about 11.000 non-zeros (matrix). The dimensions might not be extremely high, however, the data is *very dense* and it is therefore difficult to handle efficiently. For instance, decomposing the dataset using the simple ALS algorithm took more than 24 hours, whereas Turbo-SMT yielded a speedup of 50-100× over this (cf. Figure 1).

Simultaneous Clustering of Words, Questions and Regions of the Brain

One of the strengths of CMTF is its expressiveness in terms of simultaneously soft-clustering all involved entities of the problem. By taking a low rank decomposition of the BrainQ data (using $r = 5$ and $s_I = 3,\ s_J = 86,\ s_K = 1$ for the tensor and s_I for the questions dimension of the matrix)[4], we are able to find groups that jointly express words, questions and brain voxels (we can also derive groups of human subjects; however, it is an active research subject in neuroscience, whether brain-scans should differ significantly between people, and is out of the scope of the present work).

In Figure 4, we display 4 such groups of brain regions that are activated given a stimulus of a group of similar words; we also display the most prominent words, along with groups of similar questions that were highly correlated with the words of each group. Moreover, we were able to successfully identify high activation of the *premotor cortex* in Group 3, which is associated with concepts such as holding or picking items up.

Predicting Brain Activity from Questions

In addition to soft-clustering, the low rank joint decomposition of the BrainQ data offers another significant result. This low dimensional embedding of the data into a common semantic space, enables the prediction of, say, the brain activity of a subject, for a given word, given the corresponding vector of question answers for that word. In particular, by projecting the question answer vector to the latent semantic space and then expanding it to the brain voxel space, we obtain a fairly good prediction of the brain activity.

To evaluate the accuracy of these predictions of brain activity, we follow a *leave-two-out* scheme, where we remove two words entirely from the brain tensor and the question matrix; we carry out the joint decomposition, in some very low dimension, for the remaining set of words and we obtain the usual set of matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$. Due to the randomized nature of Turbo-SMT, we did 100 repetitions of the procedure described below.

Let $\mathbf{q}_i$ be the question vector for some word i, and $\mathbf{v}_i$ be the brain activity of one human subject, pertaining to the same word. By left-multiplying $\mathbf{q}_i$ with $\mathbf{D}^T$, we project $\mathbf{q}_i$ to the latent space of the decomposition; then, by left-multiplying the result with $\mathbf{B}$, we project the result to the brain voxel space. Thus, our estimated (predicted) brain activity is obtained as $\hat{\mathbf{v}}_i = \mathbf{B}\mathbf{D}^T\mathbf{q}_i$

Given the predicted brain activities $\hat{\mathbf{v}}_\mathbf{1}$ and $\hat{\mathbf{v}}_\mathbf{2}$ for the two left out words, and the two actual brain images

[3] http://www.cs.cmu.edu/afs/cs/project/theo-73/www/science2008/data.html

[4] We may use imbalanced sampling factors, especially when the data is far from being 'rectangular'.

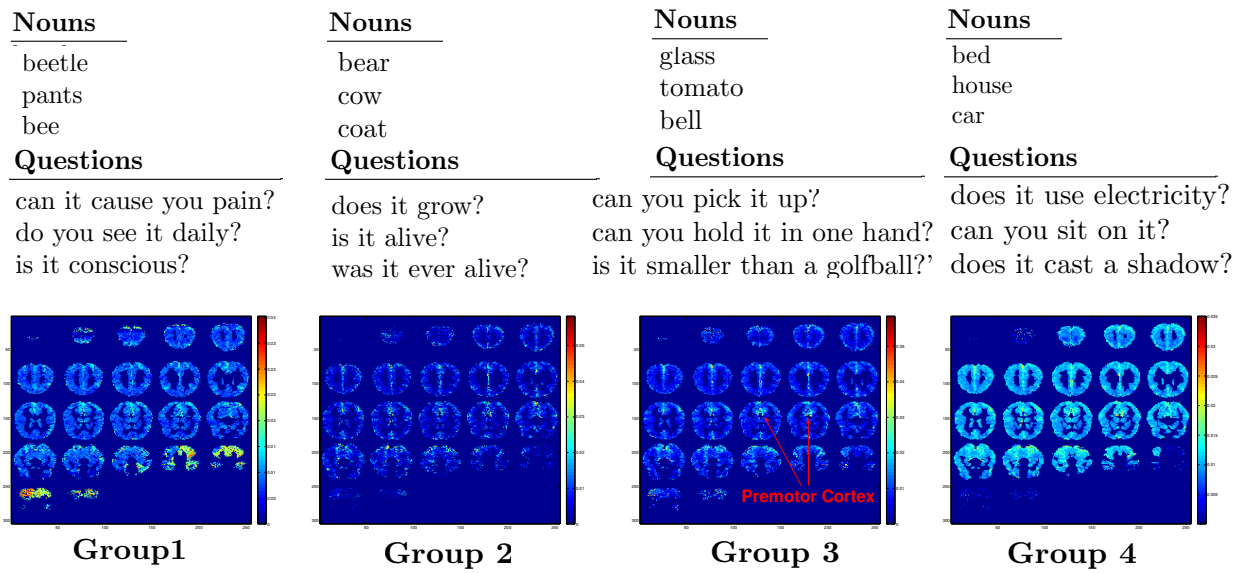

Figure 4: TURBO-SMT finds meaningful groups of words, questions, and brain regions that are (both negatively and positively) correlated, as obtained using TURBO-SMT. For instance, Group 3 refers to small items that can be held in one hand,such as a tomato or a glass, and the activation pattern is very different from the one of Group 1, which mostly refers to insects, such as bee or beetle. Additionally, Group 3 shows high activation in the *premotor cortex* which is associated with the concepts of that group.

$\mathbf{v_1}$ and $\mathbf{v_2}$ which were withheld from the training data, the *leave-two-out* scheme measures prediction accuracy by the ability to choose which of the observed brain images corresponds to which of the two words. After mean-centering the vectors, this classification decision is made according to the following rule:

$$\|\mathbf{v}_1 - \hat{\mathbf{v}}_1\|_2 + \|\mathbf{v}_2 - \hat{\mathbf{v}}_2\|_2 < \|\mathbf{v}_1 - \hat{\mathbf{v}}_2\|_2 + \|\mathbf{v}_2 - \hat{\mathbf{v}}_1\|_2$$

Although our approach is not designed to make predictions, preliminary results are very encouraging: Using only F=2 components, for the noun pair *closet/watch* we obtained mean accuracy of about 0.82 for 5 out of the 9 human subjects. Similarly, for the pair *knife/beetle*, we achieved accuracy of about 0.8 for a somewhat different group of 5 subjects. For the rest of the human subjects, the accuracy is considerably lower, however, it may be the case that brain activity predictability varies between subjects, a fact that requires further investigation.

5 Experiments

We implemented TURBO-SMT in Matlab. Our implementation of the code is publicly available.[5] For the parallelization of the algorithm, we used Matlab's Parallel Computing Toolbox. For tensor manipulation, we used the Tensor Toolbox for Matlab [7] which is optimized especially for sparse tensors (but works very well for dense ones too). We use the ALS and the CMTF-OPT [5] algorithms as baselines, i.e. we compare TURBO-SMT when using one of those algorithms as their core CMTF implementation, against the plain execution of those algorithms. We implemented our version of the ALS algorithm, and we used the CMTF Toobox[6] implementation of CMTF-OPT. We use CMTF-OPT for higher ranks, since that particular algorithm is more accurate than ALS, and is the state of the art. All experiments were carried out on a machine with 4 Intel Xeon E74850 2.00GHz, and 512Gb of RAM. Whenever we conducted multiple iterations of an experiment (due to the randomized nature of TURBO-SMT), we report error-bars along the plots. For all the following experiments we used either portions of the BRAINQ dataset, or the whole dataset.

5.1 Speedup As we have already discussed in the Introduction and shown in Fig. 1, TURBO-SMT achieves a speedup of 50-200 on the BRAINQ dataset; For all cases, the approximation cost is either same as the baselines, or is larger by a small factor, indicating that TURBO-SMT is both fast and accurate. Key facts that

[5] http://www.cs.cmu.edu/~epapalex/src/turbo_smt.zip

[6] http://www.models.life.ku.dk/joda/CMTF_Toolbox

contribute to this observed speedup are: 1) dimensionality reduction through sampling, 2) the fact that Turbo-SMT operates on sparse data throughout its lifetime, and 3) that Turbo-SMT is *highly parallelizable*. Figure 1 illustrates this behaviour. It is crucial to note that the speedup achieved is very significant: The ALS algorithm required more than 24 *hours* to be computed, and the CMTF-OPT algorithm took about 12 *hours*; Turbo-SMT was able to successfully *boost* both algorithms, while being almost as accurate.

Finally, we compare method to a technique which, to the best of our knowledge, has not been applied in CMTF. In short, in Section 5.3 of [10], it is implied that one can use the so-called Tucker3 model, in order to compress the tensor, and consequently the side matrices, and do the decomposition on the compressed data. Our experiments indicate that Turbo-SMT is at about 20 times faster on the BrainQ dataset, since the main bottleneck of this technique is compression. Moreover, such a compression-based technique is not triple-sparse, therefore, its resulting factors are *dense*, giving rise to storage and interpretability issues. We refer the interested reader to the supplementary material [2] for a detailed discussion of the compression technique and our comparison.

5.2 Accuracy In Figure 5 we demonstrate that the algorithm operates correctly, in the sense that it reduces the model cost (Equation 2.1) when doing more repetitions. In particular, the vertical axis displays the relative cost, i.e. $\frac{\text{Turbo-SMT cost}}{\text{ALS cost}}$ (with ideal being equal to 1) and the horizontal axis is the number of repetitions in the sampling. We observed that for a few executions of the algorithm, the cost was not monotonically decreasing; however, we ran the algorithm 1000 times, keeping the executions that decreased the relative cost monotonically and plotted them in Fig. 5.

5.3 Sparsity One of the main advantages of Turbo-SMT is that, it is triple-sparse, i.e. starting from (possibly) sparse data, every intermediate result of Turbo-SMT is sparse, as well as the final output. In Fig. 6 we demonstrate the sparsity of Turbo-SMT's results by introducing the relative sparsity metric; this intuitive metric is simply the ratio of the output size of the baseline algorithm, divided by the output size of Turbo-SMT. The output size is simply calculated by adding up the number of non-zero entries for all factor matrices output by the algorithm. We use a portion of the BrainQ dataset in order to execute this experiment. We can see that for the relatively dense BrainQ dataset, we obtained significantly more sparse results;, e.g. up to 65 times more sparse with almost same approximation error, for the case of CMTF-OPT. We observe a large difference of result sparsity when using CMTF-OPT, as opposed to ALS; most likely, this difference is due to the fact that, according to [5], CMTF-OPT converges to a better local minimum than ALS. The results of Fig. 6 indicate that our triple-sparse algorithm is able to capture the most useful variation of the data, successfully suppressing noise.

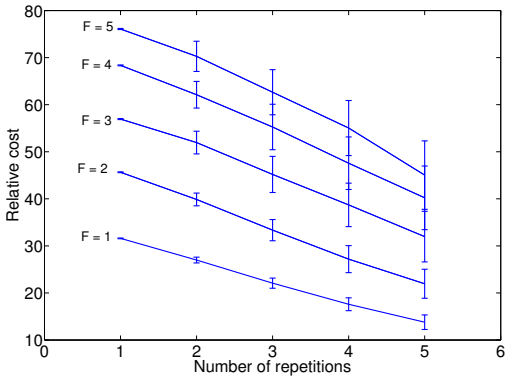

Figure 5: The relative cost of the model, as a function of the number of repetitions r is decreasing, which empirically shows that Turbo-SMT actually reduces the approximation error of the CMTF model.

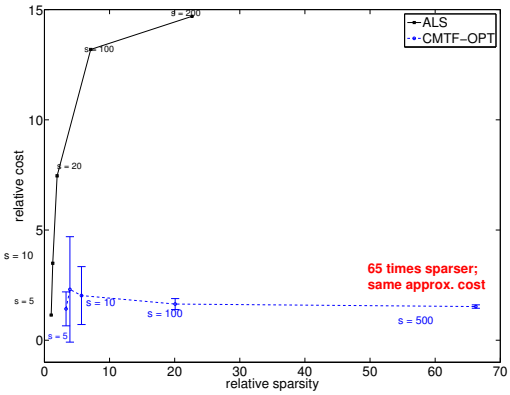

Figure 6: The relative output size vs. the relative cost indicates that, even for very dense datasets such as BrainQ, we are able to get d up to 65 fold (for CMTF-OPT) decrease in the output size, while maintaining almost same approximation error as the baseline.

6 Related Work

Coupled, Multi-block, Multi-set Models Most related to the present work is the work of Acar et

al. in [5], where a first order optimization approach is proposed, in order to solve the CMTF problem. In [6], Acar et. al apply the CMTF model, using the aforementioned first-order approach in a bioinformatics setting. In [4], Acar et. al introduce a coupled matrix decomposition, where two matrices match on one of the two dimensions, and are decomposed in the same spirit as in CMTF, while imposing explicit sparsity constraints (via ℓ_1 norm penalties); although TURBO-SMT also produces sparse factors, this so happens as a fortuitous byproduct of sampling, whereas in [4] an explicit sparsity penalty is considered.

To the best of our knowledge, TURBO-SMT is the first algorithm that is able to speed up, parallelize, and sparsify any (possibly highly fine tuned) core algorithm for CMTF.

Fast & Scalable Tensor Decompositions In [15] the authors introduced a parallel algorithm for the regular PARAFAC decomposition, where a sampling scheme of similar nature as here is exploited; in [9], a scalable MapReduce implementation of PARAFAC is presented. Additionally, the mechanics behind the Tensor Toolbox for Matlab [7] are very powerful when it comes to memory-resident tensors.

Tensor applications to brain data There has been substantial related work, which utilizes tensors for this purpose, e.g. [3], [12], and [8].

7 Conclusions

Our main contributions are the following:

- *Fast, parallel & triple-sparse algorithm:* TURBO-SMT is able to speed *any* CMTF solver up to *200 times*, producing up to *65* times sparser results, with very good accuracy.
- *Effectiveness and Knowledge Discovery*: TURBO-SMT, applied to the BRAINQ dataset, discovers meaningful triple-mode clusters: clusters of words, of questions, and of brain regions have similar behavior; as a by-product, TURBO-SMT predicts brain activity, with performance that matches state of the art predictors.
- *Reproducibility*: We make our code public, enabling reproducibility and re-usability of our work.

Acknowledgements

Research was funded by grants NSF IIS-1247489, NSF IIS-1247632, NSF CDI 0835797, and NIH/NICHD 12165321. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding parties. The authors would also like to thank Leila Wehbe and Alona Fyshe for their initial help with the BRAINQ data.

References

[1] Read the web. `http://rtw.ml.cmu.edu/rtw/`.

[2] Supplementary material. `http://www.cs.cmu.edu/~epapalex/papers/sdm14_turbo_smt_supp.pdf`.

[3] E. Acar, C. Aykut-Bingol, H. Bingol, R. Bro, and B. Yener. Multiway analysis of epilepsy tensors. *Bioinformatics*, 23(13):i10–i18, 2007.

[4] E. Acar, G. Gurdeniz, M.A. Rasmussen, D. Rago, L.O. Dragsted, and R. Bro. Coupled matrix factorization with sparse factors to identify potential biomarkers in metabolomics. In *IEEE ICDM Workshops*, pages 1–8. IEEE, 2012.

[5] E. Acar, T.G. Kolda, and D.M. Dunlavy. All-at-once optimization for coupled matrix and tensor factorizations. *arXiv preprint arXiv:1105.3422*, 2011.

[6] E. Acar, G.E. Plopper, and B. Yener. Coupled analysis of in vitro and histology tissue samples to quantify structure-function relationship. *PloS one*, 7(3):e32227, 2012.

[7] B.W. Bader and T.G. Kolda. Matlab tensor toolbox version 2.2. *Albuquerque, NM, USA: Sandia National Laboratories*, 2007.

[8] Ian Davidson, Sean Gilpin, Owen Carmichael, and Peter Walker. Network discovery via constrained tensor analysis of fmri data. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 194–202. ACM, 2013.

[9] U. Kang, E. Papalexakis, A. Harpale, and C. Faloutsos. Gigatensor: scaling tensor analysis up by 100 times-algorithms and discoveries. In *SIGKDD*, pages 316–324. ACM, 2012.

[10] T.G. Kolda and B.W. Bader. Tensor decompositions and applications. *SIAM review*, 51(3), 2009.

[11] T.M. Mitchell, S.V. Shinkareva, A. Carlson, K.M. Chang, V.L. Malave, R.A. Mason, and M.A. Just. Predicting human brain activity associated with the meanings of nouns. *Science*, 320(5880):1191–1195, 2008.

[12] Morten Mørup, Lars Kai Hansen, Sidse Marie Arnfred, Lek-Heng Lim, and Kristoffer Hougaard Madsen. Shift-invariant multilinear decomposition of neuroimaging data. *NeuroImage*, 42(4):1439–1450, 2008.

[13] Brian Murphy, Partha Talukdar, and Tom Mitchell. Selecting corpus-semantic models for neurolinguistic decoding. In *First Joint Conference on Lexical and Computational Semantics (* SEM)*, pages 114–123, 2012.

[14] Mark Palatucci, Dean Pomerleau, Geoffrey Hinton, and Tom Mitchell. Zero-shot learning with semantic output codes. *Advances in neural information processing systems*, 22:1410–1418, 2009.

[15] E. Papalexakis, C. Faloutsos, and N. Sidiropoulos. Parcube: Sparse parallelizable tensor decompositions. *Machine Learning and Knowledge Discovery in Databases*, pages 521–536, 2012.

[16] Evangelos E Papalexakis, Tom M Mitchell, Nicholas D Sidiropoulos, Christos Faloutsos, Partha Pratim Talukdar, and Brian Murphy. Scoup-smt: Scalable coupled sparse matrix-tensor factorization. *arXiv preprint arXiv:1302.7043*, 2013.