Stochastic Gradients for Large-Scale Tensor Decomposition*

Tamara G. Kolda† and David Hong‡

Abstract. Tensor decomposition is a well-known tool for multiway data analysis. This work proposes using stochastic gradients for efficient generalized canonical polyadic (GCP) tensor decomposition of large-scale tensors. GCP tensor decomposition is a recently proposed version of tensor decomposition that allows for a variety of loss functions such as Bernoulli loss for binary data or Huber loss for robust estimation. The stochastic gradient is formed from randomly sampled elements of the tensor and is efficient because it can be computed using the sparse matricized-tensor-times-Khatri-Rao product (MTTKRP) tensor kernel. For dense tensors, we simply use uniform sampling. For sparse tensors, we propose two types of stratified sampling that give precedence to sampling nonzeros. Numerical results demonstrate the advantages of the proposed approach and its scalability to large-scale problems.

Key words. tensor decomposition, stochastic gradients, stochastic optimization, stratified sampling

1. Introduction.

Tensor decomposition is the higher-order analogue of matrix decomposition and is becoming an everyday tool for data analysis. It can be used for unsupervised learning, dimension reduction, tensor completion, feature extraction in supervised machine learning, data visualization, and more. For a given d-way data tensor $\mathcal{X}$ of size $n_1 \times n_2 \times \cdots \times n_d$, the goal is to find a *low-rank* approximation $\mathcal{M}$, i.e.,

$$\mathcal{X} \approx \mathcal{M} \quad \text{where} \quad \mathcal{M} = \sum_{j=1}^{r} \mathbf{a}_1(:,j) \circ \mathbf{a}_2(:,j) \circ \cdots \circ \mathbf{a}_d(:,j). \tag{1.1}$$

The low-rank structure of $\mathcal{M}$ reveals patterns within the data, as defined by the *factor matrices*. The kth factor matrix of size $n_k \times r$ is denoted $\mathbf{A}_k$. The jth *factor* (column) in mode k is denoted $\mathbf{a}_k(:,j)$. Each *component* of $\mathcal{M}$ is a d-way outer product (denoted by $\circ$) of d factors, forming a *rank-one* tensor. We say $\operatorname{rank}(\mathcal{M}) \leq r$ because $\mathcal{M}$ can be written as the sum of r rank-one tensors. We define $n \equiv (\prod_{k=1}^{d} n_k)^{1/d}$ and assume $r \ll n^d$. The storage for $\mathcal{X}$ is n^d if it is dense and $O(\operatorname{nnz}(\mathcal{X}))$ if it is sparse. The storage for $\mathcal{M}$ is $O(r \sum_{k=1}^{d} n_k)$, which is usually much smaller than that for $\mathcal{X}$.

The standard canonical polyadic or CANDECOMP/PARAFAC (CP) tensor decomposition seeks the best low-rank approximation with respect to sum of squared errors [9, 19]. The generalized CP (GCP) tensor decomposition is a novel approach that allows for an arbitrary

*This work was supported by the Laboratory Directed Research and Development Program at Sandia National Laboratories and the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research (ASCR) Applied Mathematics Program. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government. Work by D.H. was also supported in part by NSF Grant IIS 1838179.

†Sandia National Laboratories, Livermore, CA (tgkolda@sandia.gov)

‡University of Michigan, Ann Arbor, MI (dahong@umich.edu)

elementwise loss function that is summed across all tensor entries [22]; i.e., the user provides a scalar loss function $f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ resulting in the optimization problem

$$\text{minimize } F(\mathcal{X}, \mathcal{M}) \equiv \sum_{i_1=1}^{n_1} \cdots \sum_{i_d=1}^{n_d} f(x_{i_1 \ldots i_d}, m_{i_1 \ldots i_d}) \quad \text{subject to} \quad \operatorname{rank}(\mathcal{M}) \leq r. \tag{1.2}$$

GCP is useful in situations where non-standard choices of the scalar loss function f may be appropriate. For instance, if the entries of $\mathcal{X}$ are binary values in $\{0, 1\}$, we may use the Bernoulli loss, i.e., $f(x, m) = \log(m+1) - x \log(m)$ where m represents the *odds* of observing a one. For positive data that has a Gamma distribution, we may use $f(x, m) = x/m + \log(m)$. Standard CP tensor decomposition corresponds to $f(x, m) = (x-m)^2$, and Poisson tensor decomposition [54, 12] to $f(x, m) = m - x \log m$. See [22] for full details.

In this paper, we consider the problem of fitting GCP for large-scale tensors. The GCP gradient involves a sequence of d matricized-tensor times Khatri-Rao products (MTTKRPs) with a *dense* tensor of size n^d that costs $O(rn^d)$ operations, even when $\mathcal{X}$ is sparse.[1] Thus, the computational and storage costs of computing the gradient may be prohibitive. Our solution is a flexible framework for *stochastic* gradient computation for GCP, which can be used with stochastic gradient descent (SGD) or popular variants such as Adam [25]. Our stochastic framework replaces the dense tensor with a (stochastic) sparse tensor that equals the dense tensor in expectation. If the stochastic gradient samples $s \ll n^d$ tensor entries, the resulting *sparse* MTTKRP reduces the gradient cost to $O(sr)$ flops and $O(s)$ intermediate storage, albeit with a potentially significant sacrifice in accuracy due to stochasticity.

A distinguishing feature of our framework is stratified sampling. If $\mathcal{X}$ is sparse, uniform sampling of the indices rarely samples nonzeros even though they are extremely important to the minimization. Stratified sampling randomly selects nonzeros disproportionately, which can reduce the variance of the stochastic gradient, accelerating convergence. Since stratified sampling can itself be expensive, we also introduce a semi-stratified sampling approach to further improve efficiency. Note that simply ignoring the zero entries, as is done in recommender systems where zeros indicate missing data, yields a fundamentally different problem. Section 2 surveys related work, section 3 provides notation and background, and section 4 discusses the framework.

We present computational experiments on both artificial and real-world problems in section 5, demonstrating the reliability and efficiency of using stochastic gradients. Experimentally, we can set the number of gradient samples to be the sum of the tensor dimensions, i.e., $\sum_{k=1}^{d} n_k$. For dense problems, stochastic optimization can be an order of magnitude faster than non-stochastic optimization. For sparse problems, stochastic optimization enables computing GCP for sparse tensors that would otherwise be intractable. Additionally, stratified and semi-stratified sampling of zeros and nonzeros have a clear advantage over uniform sampling in this setting. We also compare to CP-APR [12, 18] on real-world Chicago crime data. In our experiments, the stochastic methods are faster and find equivalent solutions.

[1] Both standard and Poisson CP have special structure that allows the gradient to be formed implicitly, as described in Appendix A so that sparsity of $\mathcal{X}$ can be exploited.

2. Related Work. This work is a follow-on to the introduction of GCP by Hong, Kolda, and Duersch [22]. They only considered small-scale problems that could be treated as dense. The goal of this work is to tackle larger scale problems via stochastic gradient methods.

A major motivation for the present work was that of Acar et al. [2] which focused on standard CP factorization for tensors with irretrievably missing data. They considered cases where the vast majority of the data is missing, i.e., *scarce* tensors in the terminology of [22], and observed that a scarce data tensor leads to a sparse MTTKRP in computing the gradient. That observation proves to be important here because our sampled data tensors are scarce.

In the specific case of third-order ($d = 3$) and one sample per iteration ($s = 1$), Beutel et al. [7] have proposed SGD for standard CP tensor decomposition. Since a single tensor element only updates one row in each of the three factor matrices, they are able to parallelize updates that do not operate on the same rows. They do not consider sampling strategies or the MTTKRP structure in the gradient.

Vervliet and De Lathauwer [51] pursue an alternative form of sampling for standard CP: rather than sampling *elements*, they sample *indices from each mode* of the tensor. The resulting random subtensor is used in either one outer iteration of alternating least squares (ALS) or one iteration of Gauss-Newton, with careful attention to step sizes as is needed in stochastic optimization. The updates only modify the rows of the factor matrices corresponding to the subsampled indices. This block sampling strategy is not readily amenable to sparse data tensors, which are a focus of our work.

For alternating least squares (CP-ALS), randomized matrix sketching can be used to solve the linear least squares problems in the dense [6] and sparse [11] cases; this approach does not directly apply to GCP because its subproblems are not least squares problems. Also in the vein of sketching, random projections can be used to build a sketch (or multiple sketches) of the complete tensor [38, 41, 59], including the special case of symmetric factorizations [53, 45].

In the streaming context, one dimension is typically treated as time with a new slice arriving at each time step and can be updated via SGD; see, e.g., [34]. Maehara, Hayashi, and Kawarabayashi [32] factorize tensors that arise as sums of streaming tensor samples. Similar to our approach, they apply SGD on the samples to update the factorization of the sum. In contrast to our focus on effective sampling, they primarily consider applications where a stream of samples is already given. There are also other works on updates for streaming tensors that are not based on SGD [37, 49, 31, 17].

In recommender systems and tensor completion, it is typically assumed that most data is missing, resulting in scarce tensors (sometimes incorrectly conflated with sparse tensors). Smith, Park, and Karypis [43] consider SGD for scarce tensors and only sample nonzeros as they correspond to observed entries. For symmetric tensors with missing data, Ge et al. [14] prove that SGD finds a local minimizer rather than a saddle point.

We close by highlighting that SGD has already had broad success in matrix factorization, primarily for recommender systems where zeros are treated as missing data (scarce matrices); see, e.g., [28] and references therein. However, we omit a full discussion of stochastic gradients in the matrix setting because the field is vast. To the best of our knowledge, nothing like the stratified and semi-stratified sampling proposed in this work has been proposed in the matrix regime. Gemulla et al. [15] propose a stratified version of SGD as a technique to partition the data into independent segments that can be processed in parallel, but their use of stratification

is different than our proposal.

3. Notation and background.

We provide some context for the remainder of the paper.

3.1. Probability notation and sampling background.

In this paper, we use a tilde to indicate random variables and instances thereof, e.g., $\tilde{x}$. The *expected value* is denoted $\mathbb{E}[\tilde{x}]$. If $\tilde{x} \in \{ v_1, v_2, \dots v_m \}$ is a discrete random variable that samples v_i with probability p_i, then

$$\mathbb{E}[\tilde{x}] \equiv \sum_{i=1}^{m} p_i \, v_i.$$

The random sample is *uniform* if each element has equal probability; i.e., $p_i = 1/m$ for $i = 1, 2, \dots, m$. Sampling *with replacement* means that the same v_i can be sampled more than once, i.e., that subsequent samples are independent identically distributed (i.i.d.) draws. If we let $\tilde{s}_i$ denote the number of times that v_i is sampled from a uniform distribution over s draws with replacement, then $\mathbb{E}[\tilde{s}_i] = s/m$ since there are s independent draws and each draw selects sample v_i with probability $1/m$.

3.2. Tensor notation.

For a d-way data tensor $\mathcal{X}$ of size $n_1 \times n_2 \times \cdots \times n_d$, we define

$$n = \sqrt[d]{\prod_{k=1}^{d} n_k} \quad \text{and} \quad \bar{n} = \frac{1}{d} \sum_{k=1}^{d} n_k.$$

These quantities provide convenient measures of how large the tensor is. We treat d as constant for big-O notation. Throughout, we let $i = (i_1, i_2, \dots, i_d)$ where $i_k \in \{ 1, 2, \dots, n_k \}$ and refer to this as a *multi-index* or simply an *index*. Every multi-index has a corresponding *linear index* between 1 and n^d [26]. We let Ω denote the set of all tensor indices, so necessarily $|\Omega| = n^d$. Excepting the discussion of missing data in subsection 4.4, we assume all tensor entries are known. We let $\operatorname{nnz}(\mathcal{X})$ denote the number of nonzeros in $\mathcal{X}$ and say that $\mathcal{X}$ is sparse if $\operatorname{nnz}(\mathcal{X}) \ll n^d$.

If $\mathcal{X}$ is dense, then the storage is n^d. If $\mathcal{X}$ is sparse, then only the nonzeros and their indices are stored, so the storage is $(d+1)\operatorname{nnz}(\mathcal{X})$ using a coordinate format [3]. The total storage for a rank-r approximation $\mathcal{M}$ as in (1.1) is the storage for the factor matrices, i.e., $\bar{n}dr$, and is usually significantly less than the storage for $\mathcal{X}$

3.3. MTTKRP background.

Given a tensor $\mathcal{Y}$ of size $n_1 \times n_2 \times \cdots \times n_d$ and factor matrices $\mathbf{A}_k$ (from a low-rank $\mathcal{M}$) of size $n_k \times r$ for $k = 1, \dots, d$, the MTTKRP in mode k is

$$\mathbf{Y}_{(k)} \underbrace{\left(\mathbf{A}_d \odot \cdots \odot \mathbf{A}_{k+1} \odot \mathbf{A}_{k-1} \odot \cdots \odot \mathbf{A}_1 \right)}_{\mathbf{Z}_k}. \tag{3.1}$$

The matrix $\mathbf{Z}_k$ is of size $n^d/n_k \times r$ and is the Khatri-Rao product (denoted by $\odot$) of all the factor matrices except $\mathbf{A}_k$, and the matrix $\mathbf{Y}_{(k)}$ is of size $n_k \times n^d/n_k$ and denotes the mode-k unfolding of $\mathcal{Y}$.

Much work has gone into efficient computation of MTTKRP. Bader and Kolda [3] consider both dense and sparse $\mathcal{Y}$ tensors, showing that the cost is $O(rn^d)$ for dense $\mathcal{Y}$ and $O(r \operatorname{nnz}(\mathcal{Y}))$

for sparse $\mathcal{Y}$. Phan, Tichavsky, and Cichocki [39] propose methods to reuse partial computations when computing the MTTKRP for all d modes in sequence. Much recent work has focused on more efficient representations of sparse tensors and parallel MTTKRP computations [44, 24, 29, 40]. There is also continued work on improving the efficiency of dense MTTKRP calculations [20, 5].

4. Stochastic GCP Gradient. GCP tensor decomposition generalizes CP tensor decomposition, minimizing the GCP loss function in (1.2). From [22, Theorem 3], the GCP gradient is calculated via a sequence of MTTKRP computations. Key to this calculation is the *elementwise partial gradient tensor* $\mathcal{Y}$ that is the same size as $\mathcal{X}$ and is defined as

$$y_i = \frac{\partial f}{\partial m}(x_i, m_i). \tag{4.1}$$

That is, $\mathcal{Y}$ is the tensor of first derivatives of f evaluated elementwise w.r.t. $\mathcal{X}$ and $\mathcal{M}$. Whether the data tensor $\mathcal{X}$ is dense or sparse, the elementwise partial gradient tensor $\mathcal{Y}$ is dense almost everywhere. The partial derivative, denoted $\mathbf{G}_k$, of the objective F in (1.2) w.r.t. $\mathbf{A}_k$ is

$$\mathbf{G}_k = \mathbf{Y}_{(k)}\mathbf{Z}_k, \tag{4.2}$$

where $\mathbf{Z}_k$ is the Khatri-Rao product defined in (3.1). Because $\mathcal{Y}$ is dense (even when $\mathcal{X}$ is sparse), $\mathcal{Y}$ requires n^d storage and the cost of computing MTTKRP for gradients $\mathbf{G}_k$ is $O(rn^d)$. Such costs may be prohibitive. For instance, if $n = 1000$ and $d = 4$, then $\mathcal{Y}$ would require 8 TB of storage. There are a couple of special cases (see Appendix A) where dense computation can be avoided because $\mathcal{Y}$ is formed *implicitly*, notably for standard CP [3], but this is not the case for general loss functions and so motivates stochastic approaches.

The stochastic approach is based on the following observation: $\mathcal{Y}$ becomes *sparse* when the data tensor $\mathcal{X}$ is *scarce* (i.e., most of the elements are missing). This is because, by [22, Theorem 3], the gradient in this case is the same as in (4.2) except that $\mathcal{Y}$ is defined as

$$y_i = \begin{cases} \frac{\partial f}{\partial m}(x_i, m_i) & \text{if } x_i \text{ known,} \\ 0 & \text{if } x_i \text{ missing.} \end{cases}$$

If $\mathcal{Y}$ is sparse, then the MTTKRP in (4.2) can be computed in time $O(r\,\mathrm{nnz}(\mathcal{Y}))$.

We develop a stochastic gradient whose general form is

$$\tilde{\mathbf{G}}_k = \tilde{\mathbf{Y}}_{(k)}\mathbf{Z}_k \quad \text{where} \quad \mathbb{E}[\tilde{\mathcal{Y}}] = \mathcal{Y} \quad \text{and} \quad \mathrm{nnz}(\tilde{\mathcal{Y}}) \leq s \ll n^d. \tag{4.3}$$

By linearity of expectation, $\mathbb{E}[\tilde{\mathcal{Y}}] = \mathcal{Y}$ implies $\mathbb{E}[\tilde{\mathbf{G}}_k] = \mathbf{G}_k$. Making $\tilde{\mathcal{Y}}$ sparse unlocks efficient sparse computation of the MTTKRP. Consequently, storage for $\tilde{\mathcal{Y}}$ is $O(s)$ and the cost to compute the gradients is $O(rs)$, a reduction of roughly n^d/s compared to computing the full gradient, albeit at the cost of lower accuracy.

In the remainder of this section, we discuss the pros and cons of several different choices for $\tilde{\mathcal{Y}}$ based on uniform sampling, stratified sampling, and semi-stratified sampling.

4.1. Uniform Sampling. To create a random instance of $\tilde{\mathcal{Y}}$, we sample s indices uniformly with replacement. The number of times that i is sampled is denoted as $\tilde{s}_i$, so $\sum_{i\in\Omega} \tilde{s}_i = s$. Using this sampling strategy, in expectation we have

$$\mathbb{E}[\tilde{s}_i] = \frac{s}{n^d} \quad \text{for all} \quad i \in \Omega.$$

Note that the $\tilde{s}_i$ values are not stored as a dense object but rather as just a list of the samples or in some other sparse data structure. We then define the stochastic tensor $\tilde{\mathcal{Y}}$ to be

$$\tilde{y}_i = \tilde{s}_i \frac{n^d}{s} y_i \quad \text{for all} \quad i \in \Omega.$$

The stochastic $\tilde{\mathcal{Y}}$ is sparse because at most s entries are nonzero (since at most s values of $\tilde{s}_i$ are nonzero). Clearly $\mathbb{E}[\tilde{\mathcal{Y}}] = \mathcal{Y}$ since $\mathbb{E}[\tilde{y}_i] = \mathbb{E}[\tilde{s}_i] \frac{n^d}{s} y_i = y_i$ for all $i \in \Omega$. A similar argument applies for sampling *without* replacement, but there is no practical difference when $s \ll n^d$.

The simplest way to generate a random index i is to generate d random mode indices i_k:

$$i_k = \texttt{randi}(n_k) \quad \text{for} \quad k \in \{1, 2, \ldots, d\}.$$

Here, $\texttt{randi}(m)$ indicates selecting a random integer between 1 and m. This requires d random numbers per index i. Alternatively, we can generate a random *linear* index via $\texttt{randi}(n^d)$ and then convert it to a tensor multi-index.[2] Either way is $O(1)$ work (we treat d as a constant).

The procedure is presented in Algorithm 4.1. The function `MTTKRP` corresponds to (3.1), and this can be computed efficiently because specialized sparse implementations exist as discussed in subsection 3.3. The model entries m_i and elementwise partial gradient tensor entries y_i are only computed for the s randomly selected indices. The most expensive operations are computing the model entries m_i and the corresponding MTTKRP calculations. In the implementation, we are able to share some intermediate computations across these two steps.

Algorithm 4.1 Stochastic Gradient with Uniform Sampling

function StocGrad($\mathcal{X}, \{\mathbf{A}_k\}, s$)
  $\omega \leftarrow \prod_k n_k$ // ω = # entries in $\mathcal{X}$
  $\tilde{\mathcal{Y}} \leftarrow 0$ // initialize $\tilde{\mathcal{Y}}$ to all-zero *sparse* tensor
  for $c = 1, 2, \ldots, s$ **do** // loop to sample $s \ll \omega$ indices for $\tilde{\mathcal{Y}}$
    for $k = 1, 2, \ldots, d$, **do** $i_k \leftarrow \texttt{randi}(n_k)$, **end** // sample random index $i \equiv (i_1, i_2, \ldots, i_d)$
    $m_i \leftarrow \sum_{j=1}^{r} \prod_{k=1}^{d} a_k(i_k, j)$ // compute m_i at sampled index
    $\tilde{y}_i \leftarrow \tilde{y}_i + (\omega/s)\, g(x_i, m_i)$ // compute $\tilde{y}_i$ at sampled index, $g \equiv \partial f/\partial m$
  end for
  for $k = 1, 2, \ldots, d$, **do** $\tilde{\mathbf{G}}_k \leftarrow \texttt{MTTKRP}(\tilde{\mathcal{Y}}, \mathbf{A}_k, k)$, **end** // use stochastic sparse $\tilde{\mathcal{Y}}$ to compute $\tilde{\mathbf{G}}_k$
  return $\{\tilde{\mathbf{G}}_k\}$
end function

[2]Generating d separate entries helps prevent overflow if $n^d > 2^{64}$, i.e., the total number of entries in the tensor is more than the size of the largest unsigned 64-bit integer (`uint64` in MATLAB). Larger integers can be generated by instead using extended-precision arithmetic (e.g., `int` in Python).

4.2. Stratified Sampling. Uniform sampling may not be appropriate for sparse tensors since nonzeros will rarely be sampled (each draw gets a nonzero with probability $\text{nnz}(\mathcal{X})/n^d$). Intuitively, however, we expect nonzeros in a sparse tensor to be important to the factorization.

Non-uniform sampling has been used in other stochastic gradient contexts. Needell et al. [36] analyze SGD and argue that biased sampling toward functionals with larger Lipschitz constants can improve performance. Gopal [16] similarly biases sampling toward functionals with larger gradients to reduce the variance of the stochastic gradients, and similar ideas appear in [58, 57]. This idea has many potential applications in machine learning, and roughly translates in our case to up-sampling a tensor entry if the corresponding elementwise loss function $f_i \equiv f(x_i, m_i)$ has a higher Lipschitz constant for fixed x_i. Consider GCP with the Bernoulli odds loss [22]: $f(x,m) = \log(m+1) - x \log m$ where $x \in \{0, 1\}$ and $m > 0$ corresponds to the odds of $x = 1$. For $x = 0$, $|\frac{\partial f}{\partial m}(0,m)| = 1/(m+1) \leq 1$; for $x = 1$, $|\frac{\partial f}{\partial m}(1,m)| = 1/(m^2+m)$ is unbounded as $m \downarrow 0$. Thus, the elementwise loss functions for nonzeros are not Lipschitz, their gradients can be very large, and sampling them more often by sampling zeros and nonzeros separately can reduce the variance of the stochastic gradients.

Consider a generic partition of Ω into p partitions $\Omega_1, \Omega_2, \ldots, \Omega_p$. If we partition into zeros and nonzeros, then $p = 2$. Let $s_\ell > 0$ be the number of samples from partition Ω_ℓ so that the total number of samples is $s = \sum_\ell s_\ell$. Within each partition, we sample uniformly with replacement. Thus, the expected number of times index i is sampled depends on its partition:

$$\mathbb{E}[\tilde{s}_i] = \frac{s_\ell}{|\Omega_\ell|} \quad \text{where} \quad i \in \Omega_\ell \quad \text{for all} \quad i \in \Omega.$$

We then define the stochastic tensor $\tilde{\mathcal{Y}}$ to be

$$\tilde{y}_i = \tilde{s}_i \frac{|\Omega_\ell|}{s_\ell} y_i \quad \text{where} \quad i \in \Omega_\ell \quad \text{for all} \quad i \in \Omega.$$

Once again, at most s entries of $\tilde{\mathcal{Y}}$ are nonzero, and $\mathbb{E}[\tilde{\mathcal{Y}}] = \mathcal{Y}$ since $\mathbb{E}[\tilde{y}_i] = \mathbb{E}[\tilde{s}_i] \frac{|\Omega_\ell|}{s_\ell} y_i = y_i$ for all $i \in \Omega$. Interestingly, the weight $(|\Omega_\ell|/s_\ell)$ for index i is completely independent of the overall sample or tensor size — it depends only on the partition size and the number of samples for that partition.

Algorithm 4.2 presents the stratified sampling procedure for sparse tensors, where we sample p nonzeros and q zeros. Sampling nonzeros is straightforward because they are stored as a list in coordinate format. However, sampling zeros requires sampling a random index as was done for uniform sampling and then *rejecting* the sample if it is actually a nonzero.

We can estimate how many random multi-indices are needed to obtain q valid zero samples. Let $\eta = \text{nnz}(\mathcal{X})$ and $\zeta = n^d - \eta$. We expect a very small proportion (η/n^d) of indices to be rejected. The number of samples needed to produce an average of q zero indices is

$$\frac{q}{1 - \eta/n^d} = \frac{n^d}{\zeta} q \approx q. \tag{4.4}$$

This is only on average, so we oversample to get sufficiently many zeros with high probability. Namely, we sample $\rho\,(n^d/\zeta)\,q$ indices where $\rho > 1$ is the oversample rate. A default of $\rho = 1.1$ is justified in Appendix B.

Algorithm 4.2 Stochastic Gradient with Stratified Sampling for Sparse Tensor

function StocGrad($\mathcal{X}, \{\mathbf{A}_k\}, p, q$)
  $\eta \leftarrow \text{nnz}(\mathcal{X})$ // $\eta = \#$ of nonzeros in $\mathcal{X}$
  $\zeta \leftarrow \prod_k n_k - \text{nnz}(\mathcal{X})$ // $\zeta = \#$ of zeros in $\mathcal{X}$
  $\tilde{\mathcal{Y}} \leftarrow 0$
  for $c = 1, 2, \ldots, p$ **do** // loop to sample $p \ll (\eta + \zeta)$ nonzero indices for $\tilde{\mathcal{Y}}$
    $\xi \leftarrow$ `randi`(η) // sample random nonzero index in $\{1, \ldots, \eta\}$
    $i \leftarrow$ index of ξth nonzero // extract corresponding tensor index
    $m_i \leftarrow \sum_{j=1}^{r} \prod_{k=1}^{d} a_k(i_k, j)$ // compute m_i at sampled index
    $\tilde{y}_i \leftarrow \tilde{y}_i + (\eta/p)\, g(x_i, m_i)$ // compute $\tilde{y}_i$ at sampled index, $g \equiv \partial f / \partial m$
  end for
  $c \leftarrow 0$
  while $c < q$ **do** // loop to sample $q \ll (\eta + \zeta)$ zero indices for $\tilde{\mathcal{Y}}$
    for $k = 1, 2, \ldots, d$, **do** $i_k \leftarrow$ `randi`(n_k), **end** // sample random index $i \equiv (i_1, i_2, \ldots, i_d)$
    if $x_i \neq 0$ **then** // check against list of nonzeros
      reject sample
    else
      $c \leftarrow c + 1$ // increment count of accepted zero samples
      $m_i \leftarrow \sum_{j=1}^{r} \prod_{k=1}^{d} a_k(i_k, j)$ // compute m_i at sampled index
      $\tilde{y}_i \leftarrow \tilde{y}_i + (\zeta/q)\, g(x_i, m_i)$ // compute $\tilde{y}_i$ at sampled index, $g \equiv \partial f / \partial m$
    end if
  end while
  for $k = 1, 2, \ldots, d$, **do** $\tilde{\mathbf{G}}_k \leftarrow$ `MTTKRP`$(\tilde{\mathcal{Y}}, \mathbf{A}_k, k)$, **end** // use stochastic sparse $\tilde{\mathcal{Y}}$ to compute $\tilde{\mathbf{G}}_k$
  return $\{\tilde{\mathbf{G}}_k\}$
end function

Since we have to check against the entire list of nonzeros for each sample, the cost of the rejection sampling for zeros can be significant. To achieve a speed that does not dominate the other costs, our MATLAB implementation uses various efficiencies such as conversion from multi-indices to linear indices, presorting the list of nonzero linear indices, and using the hidden builtin function `_ismemberhelper`. If $n^d \geq 2^{64}$, conversion to linear indices so that we can use these efficiencies is not possible. Beyond MATLAB, efficiency can be achieved using extended precision, hash tables, etc. As an alternative to *specialized* implementations for efficient rejection sampling (that are potentially language and architecture dependent), we propose a *specialized* algorithm that entirely avoids rejection sampling in the next subsection.

4.3. Semi-Stratified Sampling. To avoid rejection sampling entirely, we propose a variant that we call *semi-stratified* sampling. We sample zeros incorrectly but then correct for it when sampling nonzeros. Specifically, we sample without rejection to obtain "zeros", knowing that a small proportion may actually be nonzeros. We still sample nonzeros explicitly but now add a correction to account for the possibility that they were also wrongly sampled as "zeros."

Namely, we sample p nonzeros and q "zeros" from Ω (the entire set of indices). Let $\tilde{p}_i$ be the number of times that index i was sampled as a nonzero where $\sum_i \tilde{p}_i = p$. Clearly, $\tilde{p}_i = 0$ if $x_i = 0$. Let $\tilde{q}_i$ be the number of times that index i is sampled as a "zero" from the full set of possible indices where $\sum_i \tilde{q}_i = q$. It is possible that some nonzeros are sampled, i.e., we

can have $\tilde{q}_i > 0$ when $x_i \neq 0$. Using these counts and recalling $\eta = \text{nnz}(\mathcal{X})$, we define $\tilde{\mathcal{Y}}$ as

$$\tilde{y}_i = \tilde{p}_i \frac{\eta}{p}(y_i - c_i) + \tilde{q}_i \frac{n^d}{q} c_i \quad \text{where} \quad c_i \equiv \frac{\partial f}{\partial m}(0, m_i).$$

This still satisfies $\mathbb{E}[\tilde{y}_i] = y_i$. For i such that $x_i = 0$, we have $\tilde{p}_i = 0$ and $c_i = y_i$, so

$$\mathbb{E}[\tilde{y}_i] = \mathbb{E}[\tilde{q}_i] \frac{n^d}{q} c_i = \frac{q}{n^d} \frac{n^d}{q} y_i = y_i.$$

For i such that $x_i \neq 0$, we have

$$\mathbb{E}[\tilde{y}_i] = \mathbb{E}[\tilde{p}_i] \frac{\eta}{p}(y_i - c_i) + \mathbb{E}[\tilde{q}_i] \frac{n^d}{q} c_i = \frac{p}{\eta} \frac{\eta}{p}(y_i - c_i) + \frac{q}{n^d} \frac{n^d}{q} c_i = (y_i - c_i) + c_i = y_i.$$

The procedure is implemented in Algorithm 4.3. The procedure for the nonzero samples is identical to that in Algorithm 4.2 except for the adjustment term $-g(0, m_i)$ to ensure that the expectations are correct. The procedure for the "zeros" differs because it samples over the entire index space and does not reject nonzeros; it assumes that $x_i = 0$ in computing y_i.

Algorithm 4.3 Stochastic Gradient with Semi-Stratified Sampling for Sparse Tensor

function StocGrad($\mathcal{X}, \{\mathbf{A}_k\}, p, q$)
  $\eta \leftarrow \text{nnz}(\mathcal{X})$ // # of nonzeros in $\mathcal{X}$
  $\omega \leftarrow \prod_k n_k$ // # entries in $\mathcal{X}$
  $\tilde{\mathcal{Y}} \leftarrow 0$
  for $c = 1, 2, \ldots, p$ **do** // loop to sample $p \ll \omega$ indices for $\tilde{\mathcal{Y}}$
    $\xi \leftarrow$ `randi`(η) // sample random nonzero index in $\{1, \ldots, \eta\}$
    $i \leftarrow$ index of ξth nonzero // extract corresponding tensor index
    $m_i \leftarrow \sum_{j=1}^{r} \prod_{k=1}^{d} a_k(i_k, j)$ // compute m_i at sampled index
    $\tilde{y}_i \leftarrow \tilde{y}_i + (\eta/p)\,[g(x_i, m_i) - g(0, m_i)]$ // compute $\tilde{y}_i$ at sampled index with *correction* term
  end for
  for $c = 1, 2, \ldots, q$ **do** // loop to sample $q \ll \omega$ "zero" indices for $\tilde{\mathcal{Y}}$
    for $k = 1, 2, \ldots, d$, **do** $i_k \leftarrow$ `randi`(n_k), **end** // sample index $i \equiv (i_1, i_2, \ldots, i_d)$
    $m_i \leftarrow \sum_{j=1}^{r} \prod_{k=1}^{d} a_k(i_k, j)$ // compute m_i at sampled index
    $\tilde{y}_i \leftarrow \tilde{y}_i + (\omega/q)\, g(0, m_i)$ // compute $\tilde{y}_i$ at sampled index, *assuming* $x_i = 0$
  end for
  for $k = 1, 2, \ldots, d$, **do** $\tilde{\mathbf{G}}_k \leftarrow$ `MTTKRP`$(\tilde{\mathcal{Y}}, \mathbf{A}_k, k)$, **end** // use sparse $\tilde{\mathcal{Y}}$ to compute $\tilde{\mathbf{G}}_k$
  return $\{\tilde{\mathbf{G}}_k\}$
end function

4.4. Adapting to Weighted Formulations. Any of the above sampling methods can be easily adapted to the weighted version of GCP with the *weighted* loss function

$$\text{minimize } F(\mathcal{X}, \mathcal{M}) \equiv \sum_i w_i\, f(x_i, m_i) \quad \text{subject to} \quad \text{rank}(\mathcal{M}) \leq r. \tag{4.5}$$

In estimating the gradient, the only change in the methods is to replace y_i in (4.1) with

$$y_i = w_i \frac{\partial f}{\partial m}(x_i, m_i).$$

We do not explicitly study the weighted formulation in this work, but we briefly mention two scenarios where such methods may be useful.

Adapting to Weighted Formulations. In the context of recommender systems, zeros are generally treated as missing data. The data is assumed to be missing at random (MAR), meaning that the probability of a data item being present does not depend on its value. However, it has been argued in the matrix case that this may be a flawed assumption [35]. Instead, we may consider including the zero terms but *down-weighting* them by using a weighted scheme, e.g., $w_i = 1$ for $x_i \neq 0$ and $w_i = 0.1$ for $x_i = 0$. Many large-scale tensor applications have a recommender system flavor, where down-weighting of the zero entries may be appropriate. More generally, down-weighting can be appropriate for entries that are less reliable or noisier, as has been done in various matrix factorization applications [13, 23, 47, 56]. See also [46, 48] for work on computing weighted matrix factorizations and [21] for a recent analysis of matrix factorization weights when columns have heterogeneous noise levels.

Weighted Formulations for Missing Data. Conversely, if some portion of data is missing, there are various strategies that can be used to avoid sampling missing elements. However, this can also be handled easily by setting the weights of missing entries to be zero. Ideally, elements with a weight of zero should be avoided during sampling.

5. Experimental Results.

All experiments were run using MATLAB (Version 2018a) on a Dual Socket Intel E5-2683v3 2.00GHz CPU with 256 GB memory. The methods are implemented in `gcp_opt` in the Tensor Toolbox for MATLAB [4].

5.1. Stochastic optimization algorithm.

The stochastic gradients can be used with any number of stochastic optimization methods. We use the popular Adam [25] method because it is less sensitive to the learning rate than standard SGD. The method is detailed in Algorithm 5.1. Based on the empirical results that follow, we recommend setting $s = d\bar{n}$. If the total number of gradient samples is s, then we use s samples for uniform sampling in Algorithm 4.1 and $p = \lfloor s/2 \rfloor$ and $q = \lceil s/2 \rceil$ for the stratified sampling in Algorithm 4.2 and semi-stratified sampling in Algorithm 4.3. The parameter α is the learning rate and defaults to 0.01. The Adam parameters are set to the values recommended in the original paper: $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. We employ a few standard modifications. We group the iterates into epochs, and the number of iterations per epoch defaults to $\tau = 1000$. To track progress, we estimate the function value $F(\mathcal{X}, \mathcal{M})$ once per epoch. Whenever the function value fails to decrease, we either decay the learning rate by $\nu = 0.1$ (as motivated by [30]) or quit (after more than $\kappa = 1$ failures). We enforce any lower bounds ℓ given for the parameters by simple projection. In all of our examples, we use $\ell = 0$ (nonnegativity constraint).

We estimate $F(\mathcal{X}, \mathcal{M})$ (via the function EstObj) in a way that is analogous to the stochastic gradient computation. There are two key differences. First, we use a much larger number of samples to ensure reasonable accuracy, which is less important for the gradient computation. Second, the set of samples used for function estimation are fixed across all iterations for consistent evaluation across epochs. For uniform sampling, let $\tilde{s}_i$ be the number of times that index i is selected and then estimate

$$\hat{F} \equiv \sum_{i \in \Omega} \tilde{s}_i \frac{n^d}{s} f(x_i, m_i). \tag{5.1}$$

For stratified sampling, let $\tilde{p}_i$ denote the number of times that nonzero i is selected and $\tilde{q}_i$ be

Algorithm 5.1 GCP with Adam (GCP-Adam)

function GcpAdam($\mathcal{X}$, r, s, α, β_1, β_2, ϵ, τ, κ, ν, ℓ)
  for $k = 1, 2, \ldots, d$ **do**
    $\mathbf{A}_k \leftarrow$ random matrix of size $n_k \times r$
    $\mathbf{B}_k, \mathbf{C}_k \leftarrow$ all-zero matrices of size $n_k \times r$ // temporary variables used for Adam
  end for
  $\hat{F} \leftarrow$ EstObj($\mathcal{X}, \{\mathbf{A}_k\}$) // estimate loss with fixed set of samples
  $c \leftarrow 0$ // c = # of bad epochs (i.e., without improvement)
  $t \leftarrow 0$ // t = # of Adam iterations
  while $c \leq \kappa$ **do** // κ = max # of bad epochs
    Save copies of $\{\mathbf{A}_k\}$, $\{\mathbf{B}_k\}$, $\{\mathbf{C}_k\}$ // save in case of failed epoch
    $\hat{F}_{\text{old}} \leftarrow \hat{F}$ // save to check for failed epoch
    for τ iterations **do** // τ = # iterations per epoch
      $\{\tilde{\mathbf{G}}_k\} \leftarrow$ StocGrad($\mathcal{X}, \{\mathbf{A}_k\}, s$) // s = # samples per stochastic gradient
      for $k = 1, \ldots, d$ **do**
        $\mathbf{B}_k \leftarrow \beta_1 \mathbf{B}_k + (1-\beta_1)\tilde{\mathbf{G}}_k$
        $\mathbf{C}_k \leftarrow \beta_2 \mathbf{C}_k + (1-\beta_2)\tilde{\mathbf{G}}_k^2$ // Adam update depends on
        $\hat{\mathbf{B}}_k \leftarrow \mathbf{B}_k / (1-\beta_1^t)$ β_1, β_2, ϵ; α = learning rate
        $\hat{\mathbf{C}}_k \leftarrow \mathbf{C}_k / (1-\beta_2^t)$
        $\mathbf{A}_k \leftarrow \mathbf{A}_k - \alpha\,(\hat{\mathbf{B}}_k \oslash \sqrt{\hat{\mathbf{C}}_k + \epsilon})$
        $\mathbf{A}_k \leftarrow \max\{\mathbf{A}_k, \ell\}$ // ℓ = lower bound
      end for
      $t \leftarrow t + 1$
    end for
    $\hat{F} \leftarrow$ EstObj($\mathcal{X}, \{\mathbf{A}_k\}$) // estimate loss with fixed set of samples
    if $\hat{F} > \hat{F}_{\text{old}}$ **then** // check for failure to decrease loss
      Restore saved copied of $\{\mathbf{A}_k\}$, $\{\mathbf{B}_k\}$, $\{\mathbf{C}_k\}$ // revert to last epoch's variables
      $\hat{F} \leftarrow \hat{F}_{\text{old}}$ // revert to prior function value
      $t \leftarrow t - \tau$ // wind back the iteration counter
      $\alpha \leftarrow \alpha\,\nu$ // reduce the learning rate
      $c \leftarrow c + 1$ // increment # of bad epochs
    end if
  end while
  return $\{\mathbf{A}_k\}$
end function

the same for zero i and then estimate

$$\hat{F} \equiv \sum_{x_i \neq 0} \tilde{p}_i \frac{\eta}{p} f(x_i, m_i) + \sum_{x_i = 0} \tilde{q}_i \frac{\zeta}{q} f(x_i, m_i). \tag{5.2}$$

In either case, it is easy to show that $\mathbb{E}[\hat{F}] = F$. When we perform multiple runs of the same problem, we use the same set of samples for the function estimation *across all runs* so that they can be easily compared. The only exception is the non-stochastic method, which computes the full objective function.

5.2. Sample Size and Comparison to Full Method for Dense Tensors. We study the effect of sample size (also known as minibatch size) to understand the relevant trade-offs: larger sample sizes yield lower variance stochastic gradients but higher costs per iteration. We

compute the GCP decomposition on an artificial four-way tensor of size $200 \times 150 \times 100 \times 50$ and rank $r = 5$ using the gamma loss function: $f(x, m) = x/m - \log m$ with a nonnegativity constraint on the factor matrices. Appendix C provides the details of the data generation.

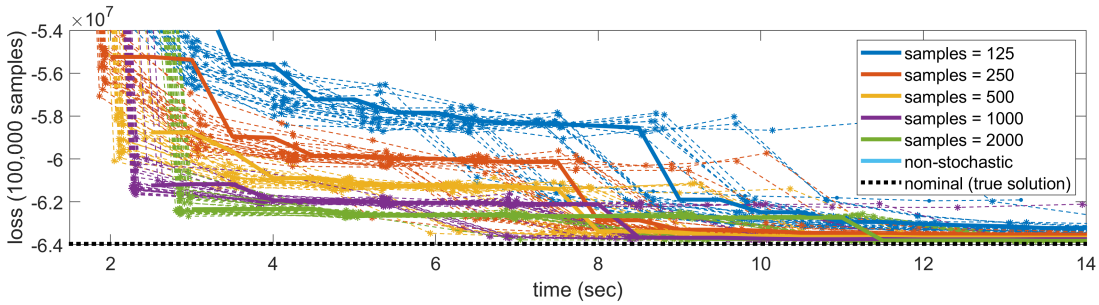
Figure 5.1 shows the results of GCP-Adam with various sample sizes ranging from $s = 125$ to $s = 2000$. For comparison, we also include non-stochastic results based on the bound-constrained limited-memory BFGS method [8] using *full* gradients; this optimization approach is standard in MATLAB toolboxes such as Tensor Toolbox [1, 4] and TensorLab [52]. The same set of 25 initial guesses is used for every instance. The initial guesses comprise factor matrices with entries drawn uniformly from $(0, 1)$. For GCP-Adam, we estimate the loss $\hat{F}$ using 100,000 uniformly sampled entries that are fixed across all epochs and trials. For the stochastic gradient, we use uniform sampling.

In the top two subfigures, we plot the function value versus time. In Figure 5.1a, we consider just the stochastic methods and see the variation between them. There are $d\bar{n}r = 2500$ free parameters and $\max_k n_k = 200$. Common wisdom is to make one pass through the data per epoch, which would require $s = n^d/\tau = 150,000$ samples per iteration. However, we see that two orders of magnitude fewer samples are needed in practice, arguably due to the low-rank structure in the data. Another option is to set s large enough so that we get at least one sample per row of $\tilde{\mathbf{Y}}_{(k)}$ and thus every row of $\mathbf{A}_k$ is updated. At a minimum, therefore, we may want $s \geq \max_k n_k = 200$.[3] Fewer samples generally corresponds to less progress per epoch; however, sometimes fewer samples is advantageous because it takes a different path to the solution. For instance, we see that $s = 2000$ initially makes better progress, but $s = 500$ and $s = 1000$ find the minimum more quickly. At the other extreme, $s = 125$ is the lowest cost per epoch, but its progress in reducing the loss is hindered by too few samples. In Figure 5.1b we show a longer time range on the x-axis so that the non-stochastic method is visible. The non-stochastic method is approximately an order of magnitude slower.

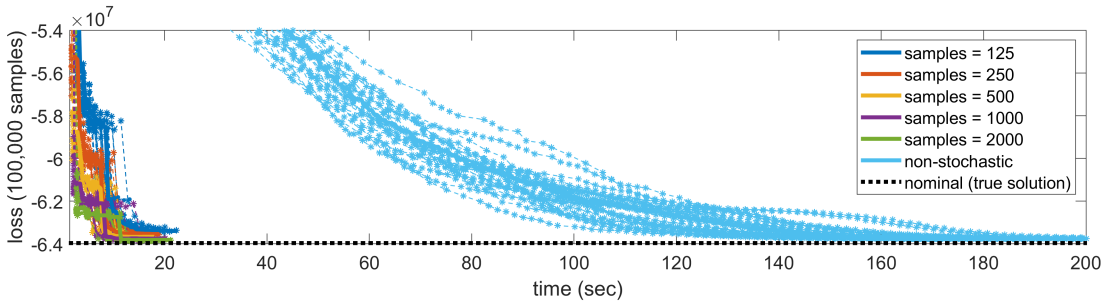
The final function value may not tell the whole story, so we provide another measure of success. Each instance is run with 25 random starts. Figure 5.1c shows the number of times that each instance recovers the true solution, meaning that the cosine similarity score between the true solution and the recovered solution is at least 0.9. (See Appendix E for details of the cosine similarity score.) The only time the true solution is *not* recovered is one run for $s = 125$ samples per gradient.

Figure 5.1d shows box plots of the cost per epoch, which is 1000 stochastic gradient evaluations (with s specified on the x-axis) plus one function value estimation with 100,000 samples. In each box plot, the middle line indicates the median, and the bottom and top edges of the box indicate the 25th and 75th percentiles, respectively. The whiskers extend to the most extreme data points not considered outliers, and the outliers are plotted individually as red '+' symbols. Clearly, the cost per epoch grows linearly with the number of samples, but there are fixed costs that dominate the per iteration cost. Notably, $s = 2000$ uses 16 times as many samples as $s = 125$ but is generally only around twice as slow.

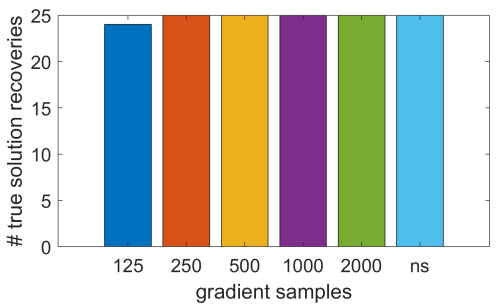
[3] This does not guarantee that every row is updated every time. To do so with high probability, one might use roughly $10 \max_k n_k$ samples. From [27, Appendix A]: To sample ρt distinct "types" (i.e., row indices) from a set of t types where $\rho \in (0, 1)$, the expected number of draws is $t \log(1/(1-\rho)) + O(1)$. Therefore, the expected number of samples needed to collect 99.99% of the members of a set with t types is less than $10t$.



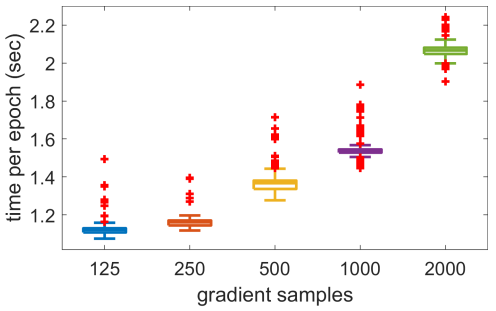
(a) Individual runs with x-axis zoomed in to show the differences in the stochastic runs. Each dashed line represents a single run, and the markers signify epochs. The marker is an asterisk if the true solution was recovered and a dot otherwise. Solid lines represent the median. Dashed black line is the function value estimate for the true solution. The *same* set of samples is used to estimate the loss across every individual run.



(b) Individual runs with x-axis zoomed out to show the non-stochastic method. For the non-stochastic method, each marker is a single iteration and the *true* loss is plotted.



(c) Number of times the true solution was recovered, i.e., cosine similarity ≥ 0.9.



(d) Boxplot of time per epoch. Each epoch is 1000 stochastic gradients plus one estimation of the function value.

Figure 5.1: GCP with Gamma loss $f(x, m) = x/m + \log m$ on artificial dense data tensor of size $200 \times 150 \times 100 \times 50$ and rank $r = 5$, comparing various numbers of samples for the stochastic gradient in GCP-Adam and the non-stochastic GCP. For each instance, we do 25 runs with different initial guesses. (The same 25 initial guesses are used for each instance.)

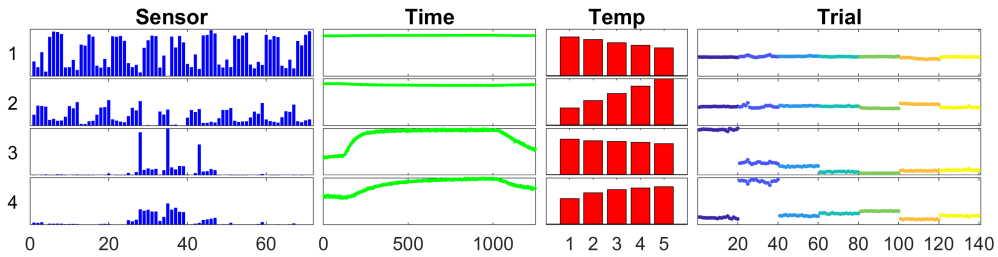
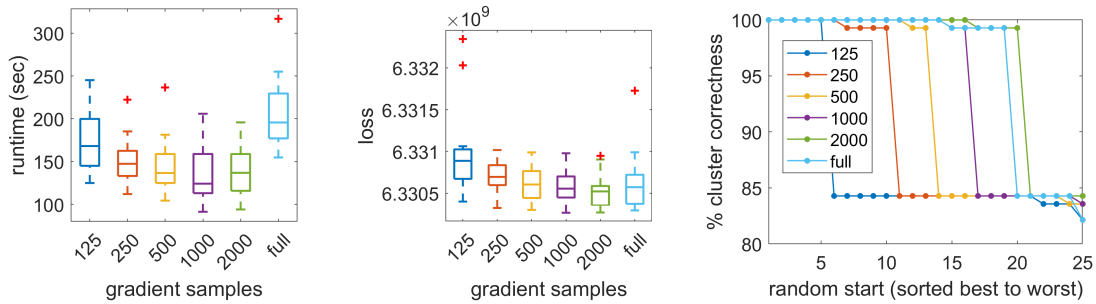


Figure 5.2: Factorization with the lowest overall score. GCP with β-divergence loss $f(x,m) = 2m^{\beta} + 2xm^{-\beta}$ with $\beta = 1/2$, rank $r = 4$, and $s = 1000$ on real-world dense gas data tensor of size $71 \times 1250 \times 5 \times 140$. Sensor components are scaled proportional to component magnitude; the rest are normalized to length one. Trial symbols are color coded by gas.

5.3. Application to Dense Gas Measurements Tensor. This section illustrates the benefits of using stochastic gradients and the effect of sample size on GCP decomposition for dense real data. We consider a tensor based on chemo-sensing data collected by Vergara et al. [50].[4] The dataset consists of measurements as a gas is blown over an array of conductometric metal-oxide sensors in a wind tunnel. The tensor modes correspond to 71 sensors, 1250 time points, 5 temperatures, and 140 trials (7 gases with 20 repeats each). This is a dense, relatively small 0.5 GB tensor. We note that Vervliet and De Lathauwer [51] and Battaglino, Ballard and Kolda [6] considered a similar 2 GB tensor derived from the same original dataset with more time points but only three gases; however, we do not compare directly to their approach because they focus on standard CP tensor decomposition. For the loss, we use β-divergence with $\beta = 1/2$ yielding the loss function $f(x,m) = 2m^{1/2} + 2xm^{-1/2}$ with nonnegativity constraints. This loss function is not necessarily optimal, but it seemed to work well for this data in our experiments. Furthermore, it is an attractive choice since the tensor is nonnegative and may have some outliers. We use rank $r = 4$ because this is the smallest rank that sufficiently distinguished the different gases in our experiments.

We run GCP-Adam using uniform sampling and the non-stochastic GCP under the same experimental conditions as in the previous subsection; initial guesses are scaled to match the magnitude of the data tensor. Figure 5.2 shows the results of the best overall run in terms of the final objective value. The components are ordered by magnitude, the sensor mode is normalized to the magnitude of the component, and the other modes are normalized to unit length. The trial mode is color coded by gas. The first two components focus largely on sensor variations due to temperature. The final two components capture some temporal patterns for each gas that impact sensors near the middle of the array, where the gas is likely most concentrated. The factorization identifies generally smooth temporal profiles and tends to group the same gas (indicated by color) in the trial mode.

[4] Available at http://archive.ics.uci.edu/ml/datasets/Gas+sensor+arrays+in+open+sampling+settings. The dataset contains data for 11 gases; we used 7 that have more distinctive behaviors. It also has 6 sensor positions; we used position 3 (middle of the wind tunnel) where some of the interesting behaviors occur. The data is recorded at ∼100 Hz. We downsampled to 5 Hz, using the nearest measurement when needed, and skipped the first 9 seconds. We also removed sensor 33, which seemed to have erratic measurements.



(a) Boxplots of total runtimes.

(b) Boxplots of final model losses.

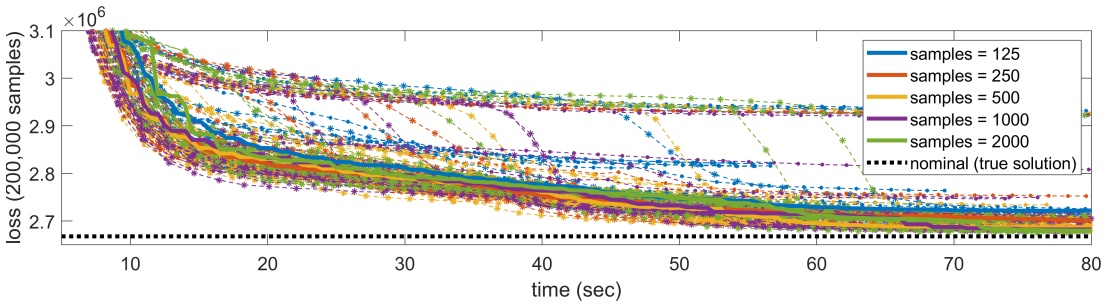
(c) Clustering performance of each run. We apply k-means to the fourth factor matrix and measure what percentage of the 140 trials are clustered correctly by gas.

Figure 5.3: Comparison of GCP-Adam with uniform sampling and sample sizes from $s = 125$ to $s = 2000$ as well as non-stochastic GCP ("full"), for fitting a dense gas tensor of size $71 \times 1250 \times 5 \times 140$ with β-divergence loss using $\beta = 1/2$, i.e., $f(x, m) = 2m^{1/2} + 2xm^{-1/2}$. Each box plot is based on 25 runs with different initial guesses. (The same 25 initial guesses are used for each box plot.)

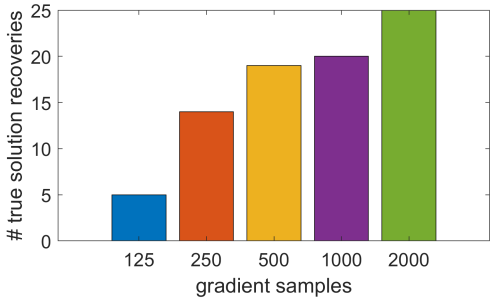
Figure 5.3 shows performance results. Figure 5.3a shows boxplots of the runtimes for the varying sample sizes and the non-stochastic GCP ("full"). The smaller sample sizes take longer because they converge more slowly even though each epoch is cheaper. The non-stochastic instance is overall slowest. Figure 5.3b shows the range of objective function values for each instance, which is very small. For the stochastic instances, larger sample sizes tend to achieve a marginally better loss, as indicated by improved median and percentile losses. The stochastic instances with $s \geq 500$ samples per gradient perform at least as well as the non-stochastic instance, while being much faster. Figure 5.3c shows performance on a clustering task. For each run (i.e., a given random start and instance), the rows of the fourth factor matrix are clustered via k-means[5] and we measure what percentage of the 140 trials (7 gases with 20 trials each) get clustered correctly. Larger sample sizes have generally better clustering performance here. Non-stochastic GCP performs similarly to $s = 2000$ samples per gradient.

5.4. Sample Size for Sparse Tensors. We consider a four-way *sparse* binary tensor of size $200 \times 150 \times 100 \times 50$ and rank $r = 5$ generated according to an *odds* model tensor, i.e., where m_i gives the odds that $x_i = 1$. The procedure to generate the data is described in detail in Appendix D. The factor matrices in the solution $\mathcal{M}$ have $(r - 1)$ sparse columns and one constant column. The result is a tensor that has 150,452 'structural' nonzeros (from the sparse columns) and 374,435 'noise' nonzeros (from the dense column), with an overall density of 0.35%. We use the loss corresponding to Bernoulli data with an odds link, i.e., $f(x, m) = \log(m + 1) - x \log m$ and a nonnegativity constraint on the factor matrices.

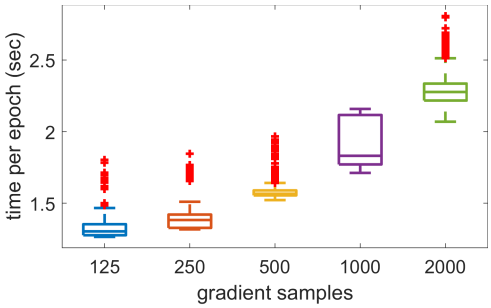
[5]We used 500 replicates with MATLAB's built-in `kmeans` command to avoid local minima.



(a) Individual runs. Each dashed line represents a single run, and the markers signify epochs. The marker is an asterisk if the true solution was recovered and a dot otherwise. Solid lines represent the median. Dashed black line is the function value estimate for the true solution. The *same* set of samples is used to estimate the loss across every individual run.



(b) Number of times the true solution was recovered, i.e., cosine similarity ≥ 0.9.



(c) Boxplot of time per epoch. Each epoch is 1000 stochastic gradients plus one estimation of the function value.

Figure 5.4: GCP with Bernoulli loss $f(x, m) = \log(m + 1) - x \log m$ on artificial sparse data tensor of size $200 \times 150 \times 100 \times 50$ and rank $r = 5$ with 524,468 (0.35%) nonzeros. Comparing various numbers of samples for the stochastic gradient in GCP-Adam with stratified sampling. For each instance, we do 25 runs with different initial guesses. (The same 25 initial guesses are used for each instance.)

The results of GCP-Adam with various sample sizes ranging from $s = 125$ to $s = 2000$ are shown in Figure 5.4. We use stratified sampling to compute the stochastic gradient, and the gradient samples are evenly divided between zeros and nonzeros. The same set of 25 initial guesses is used for every instance. The initial guesses comprise factor matrices with entries drawn uniformly from $(0, 1)$ and then scaled to match the magnitude of the true solution tensor. For GCP-Adam, we estimate the loss $\hat{F}$ using 200,000 stratified sampled entries (evenly divided between zeros and nonzeros) that are fixed across all epochs and trials.

Figure 5.4a plots the individual runs. For the stochastic method, the overall run time is about four times more than for the (easier) Gamma case. We omit the non-stochastic method since it is again significantly slower. This is arguably a difficult test problem in terms of recovering the true factors, especially compared to the Gamma problem in subsection 5.2. From Figure 5.4b, observe that $s = 125$ fails to find the true solution more often than it

succeeds. For $s \geq 250$, the true solution is recovered in the majority of cases, and the recovery rate improves as the number of samples increases; all 25 runs in this experiment succeeded for $s = 2000$. Figure 5.4c shows boxplots of the time per epoch. As was the case with uniform sampling of a same-sized dense tensor in subsection 5.2, the time per epoch mainly consists of costs that grow linearly with the number of samples and fixed costs that, at this scale, remain significant. As before, $s = 2000$ uses 16 times as many samples as $s = 125$ but is generally only around twice as slow.

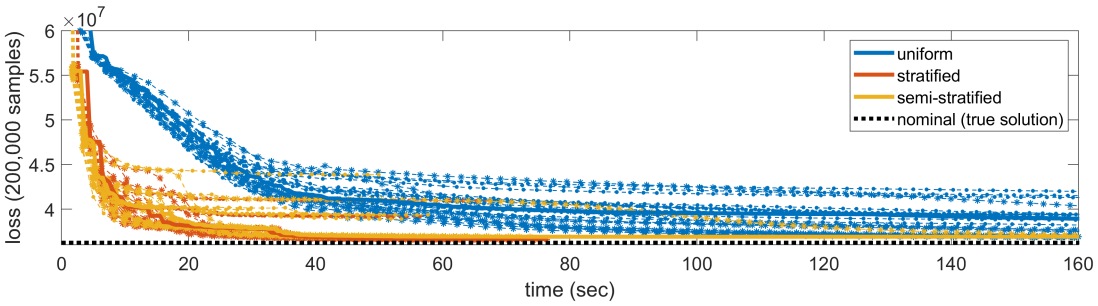
5.5. Comparison of Sampling Strategies for Sparse Tensors. This section shows the benefit of using stratified and semi-stratified sampling over uniform. We use the same procedure as subsection 5.4 to create a sparse binary tensor (detailed in Appendix D). We generate a tensor of size $400 \times 300 \times 200 \times 100$ that is 0.38% dense. This example has 4,402,374 'structural' nonzeros and 4,788,052 'noise' nonzeros, with a total of 9,181,549 nonzeros (less than the sum due to overlap). Storing it takes 0.37 GB as a sparse tensor, but would take 19 GB as a dense tensor.

The results of uniform, stratified, and semi-stratified sampling are shown in Figure 5.5. We calculate the estimated loss $\hat{F}$ once per epoch using 200,000 stratified sampled entries, evenly divided between zeros and nonzeros. We use $s = 1000$ samples per stochastic gradient evaluation ($s = d\bar{n}$), evenly divided between nonzeros and zeros (or "zeros") for the stratified samplers. We use 25 initial guesses with random positive values, scaled so that the norm of the initial guess is the same as the norm of the tensor. Note that uniform sampling on a sparse tensor is not as fast as it is on a dense tensor since only nonzeros are stored explicitly and every sampled index has to be checked against the list of nonzeros to determine its value.

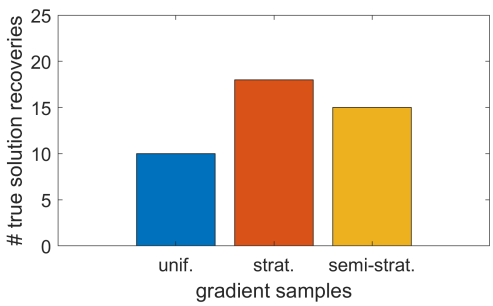
The proportion of nonzeros is arguably higher than what is observed in many real-world datasets, which is a favorable condition for uniform sampling since every sample will more likely include nonzeros. Nevertheless, stratified and semi-stratified approaches clearly outperform uniform sampling. They converge faster and more often find the true solution, all at less cost per epoch. This result is expected because stratified sampling should reduce the variance. The time advantage of the semi-stratified approach is minimal for this small example. However, the speed of the MATLAB implementation of stratified sampling depends on the ability to use linearized indices, which means that the total size of the tensor must be less than 2^{64}. For larger tensors where we cannot use this approach, the sampling efficiency can degrade by more than an order of magnitude. The semi-stratified approach has no such limitation.

Table 5.1 provides empirical validation that the variances of the stratified and semi-stratified sampling are lower than for uniform. We consider the variances with respect to the vectorized gradient, i.e., $\mathbf{g} = [\text{VEC}(\mathbf{G}_1); \text{VEC}(\mathbf{G}_2); \ldots; \text{VEC}(\mathbf{G}_d)] \in \mathbb{R}^{d\bar{n}r}$. Let $\tilde{\mathbf{g}}$ denote the random variable representing the vectorized stochastic gradient, and recall that $\mathbb{E}[\tilde{\mathbf{g}}] = \mathbf{g}$ because the sampling methods lead to unbiased estimators. Letting $\tilde{\mathbf{g}}_\xi$ denote the ξ-th of N realizations of the random variable, the quantities of interest are

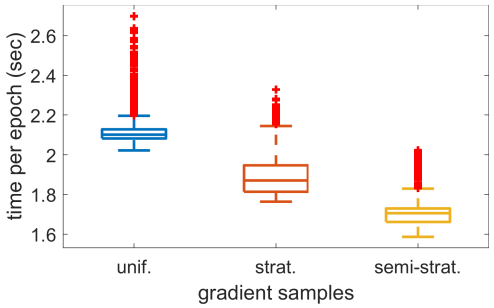
$$\text{empirical bias } = \|\hat{\mathbf{g}} - \mathbf{g}\|_2 \quad \text{where} \quad \hat{\mathbf{g}} = \frac{1}{N}\sum_{\xi=1}^{N} \tilde{\mathbf{g}}_\xi, \quad \text{and}$$



(a) Individual runs. Each dashed line represents a single run, and the markers signify epochs. The marker is an asterisk if the true solution was recovered and a dot otherwise. Solid lines represent the median. The dashed black line is the function value estimate for the true solution. Across all runs, the *same* set of samples is used to estimate the loss.



(b) Number of times the true solution was recovered, i.e., cosine similarity $\geq$ 0.9, for each number of gradient samples.



(c) Box plot of time per epoch for the different sampling methods, with the midline representing the median time.

Figure 5.5: GCP with Bernoulli loss $f(x, m) = \log(m + 1) - x \log m$ on artificial sparse data tensor of size $400 \times 300 \times 200 \times 100$ with rank $r = 5$ and 9,181,549 nonzeros (0.38% dense). Comparing different sampling strategies in GCP-Adam with $s = 1000$ samples per stochastic gradient, evenly divided between nonzeros and zeros (or "zeros") for the stratified samplers. For each strategy, we do 25 runs with different initial guesses. (The same 25 initial guesses are used for each instance.)

$$\text{empirical variance } = \text{trace}\left(\frac{1}{N}\sum_{\xi=1}^{N}(\tilde{\mathbf{g}}_\xi - \hat{\mathbf{g}})(\tilde{\mathbf{g}}_\xi - \hat{\mathbf{g}})^\intercal\right) = \frac{1}{N}\sum_{\xi=1}^{N}\|\tilde{\mathbf{g}}_\xi - \hat{\mathbf{g}}\|_2^2.$$

Table 5.1 considers two distinct cases: an initial guess (far from the solution with a larger gradient) and the overall best final solution (close to the solution with a smaller gradient). In both cases, stratified and semi-stratified sampling had lower variances than uniform. Consequently, these have lower empirical biases as well. This helps to explain the superior convergence of the stratified and semi-stratified approaches.

5.6. Application to Sparse Count Crime Data and Comparison to CP-APR. We consider a real-world crime statistics dataset comprising more than 15 years of crime data from

Table 5.1: Empirical bias and variance of stochastic gradients at an initial guess and a final model for GCP with Bernoulli loss using the artificial sparse tensor and sampling methods of Figure 5.5. For each method and model, we use $N = 1000$ stochastic gradient realizations.

sampling method	Initial guess: $\|\mathbf{g}\|_2$ = 2.00e+07 emp. bias	emp. var.	Final model: $\|\mathbf{g}\|_2$ = 3.10e+06 emp. bias	emp. var.
uniform	1.08e+06	1.26e+15	3.90e+07	1.52e+18
stratified	7.48e+05	5.64e+14	3.17e+06	9.84e+15
semi-stratified	7.68e+05	5.70e+14	3.14e+06	9.93e+15

the city of Chicago. The data is available at www.cityofchicago.org, and we downloaded a 4-way tensor version from FROSTT [42]. The tensor modes correspond to 6,186 days from 2001 to 2017, 24 hours per day, 77 communities, and 32 types of crimes. Each $\mathcal{X}(i, j, k, \ell)$ is the number of times that crime ℓ happened in neighborhood k during hour j on day i. The tensor has 5,330,673 nonzeros. Stored as a sparse tensor, it requires 0.21 GB of storage.

Since this is count data, we use GCP with a Poisson loss function, i.e., $f(x, m) = m - x \log m$ and nonnegativity constraints. We run GCP-Adam with both stratified and semi-stratified sampling using $s = d\bar{n} = 6319$ samples for the stochastic gradient. We compare to the state-of-the-art for CP alternating Poisson Regression (CP-APR) [12], using the Quasi-Newton method described in [18]. We run each method with 20 different starting points. We compute rank $r = 10$ factorizations.

Timing results are shown in Figure 5.6. The CP-APR method has to do extensive pre-processing, which is why the first iteration does not complete until approximately 140 seconds. The GCP-Adam methods descend much more quickly but do not reduce the loss quite as much, though this failure to achieve the same final minima is likely an artifact of the function estimation and/or nuance of the Adam parameters.

Although the final loss functions values are slightly different, the factorizations computed by the three methods are similar. We show the results from the first random starting point in Figure 5.7. Each rank-1 component corresponds to a row in the figure. The first column is the day, shown as a line graph. Components in this column are scaled to capture the magnitude of the overall component, while all the other components are normalized. The second column is hour of the day, shown as a bar graph. The third column is the neighborhood, shown as a bar graph. (The third column is somewhat difficult to interpret visualized this way but we show it on a map in Appendix F.) The fourth column is the crime type, sorted by overall frequency, and only showing the 13 most prevalent crimes. When working with real-world data, we generally have to experiment. Nevertheless, certain trends emerge over and over again for different starting points, ranks, and methods. In this case, we see strong commonalities among the three methods, as follows. Component 1 for CP-APR is similar to component 2 for the GCP-Adam methods, with "theft" being the most prevalent crime and a similar pattern in time. Component 2 for CP-APR is similar to component 1 for the GCP-Adam methods. Component 3 for the semi-stratified GCP-Adam has a very strong seasonal signature, becoming most active in summer months. Component 4 for stratified GCP-Adam is similar, as is component 6 for CP-APR. Component 8 in CP-APR and component 6 in

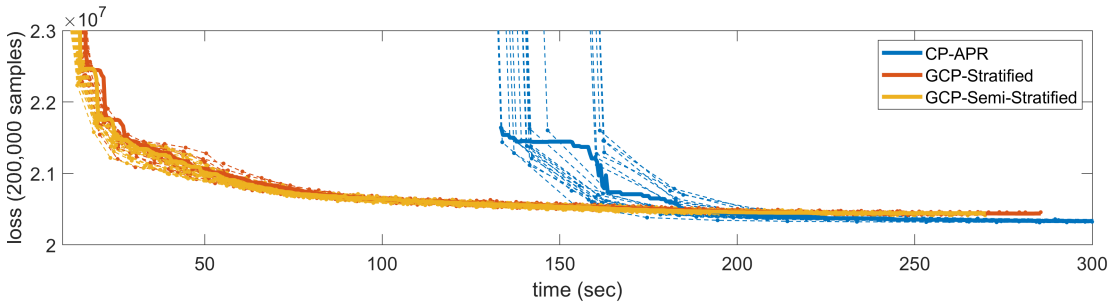
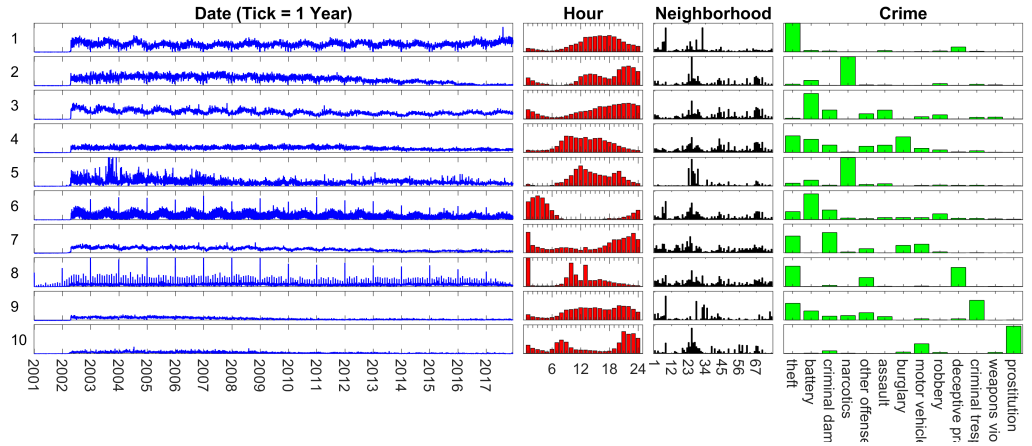


Figure 5.6: GCP with Poisson loss $f(x,m) = m - x\log m$, rank $r = 10$, and $s = 6319$ (sum of the dimensions) on real-world Chicago crime data tensor of size $6186 \times 24 \times 77 \times 32$ with 5,330,673 nonzeros. Comparing GCP-Adam using both stratified and semi-stratified sampling with CP-APR (using quasi-Newton). We run each method from 20 different initial guesses. Each dashed line represents a single run, and the markers signify epochs for GCP and iterations for CP-APR. Solid lines represent the median. Across all GCP runs, the *same* set of samples is used to estimate the loss. CP-APR computes the exact loss, and the differences between the final losses seem to be in part an artifact of the estimation.

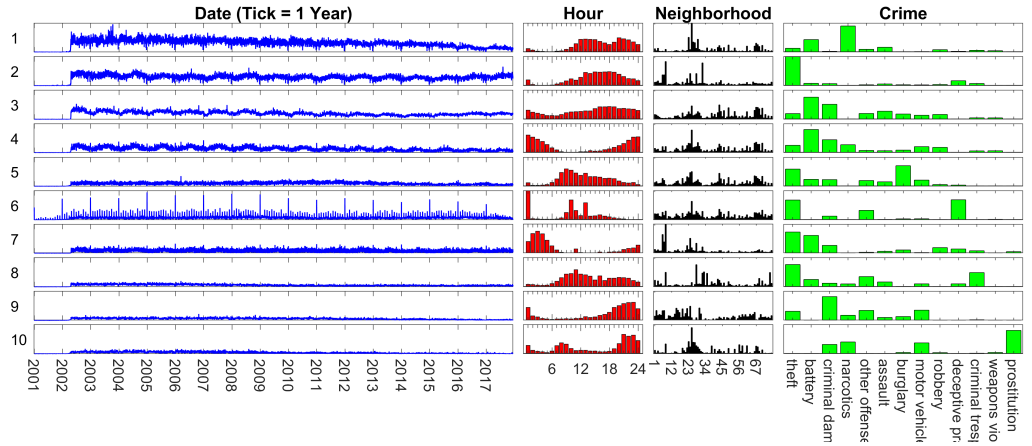
both GCP-Adam methods has a special pattern of a spike on the first of each year and again on the first of each month. There is also a spike at midnight in the hour mode. This is likely a feature of how the associated crimes were recorded in the dataset. To help further with interpretation of the rank-1 components, we zoom in on the components from the semi-stratified GCP-Adam solution in Appendix F, where we show each individual component, including drawing a heatmap of the neighborhoods on a map.

6. Conclusions. We propose a stochastic gradient for GCP tensor decomposition with general loss functions. The structure of the GCP gradient means that there is no general way to maintain sparsity in its computation even when the input tensor is sparse. Our investigation was prompted by the sparse case, but the stochastic approach applies equally well to dense tensors. A unique feature of our approach is the use of stratified and semi-stratified sampling in the gradient computation for sparse tensors.

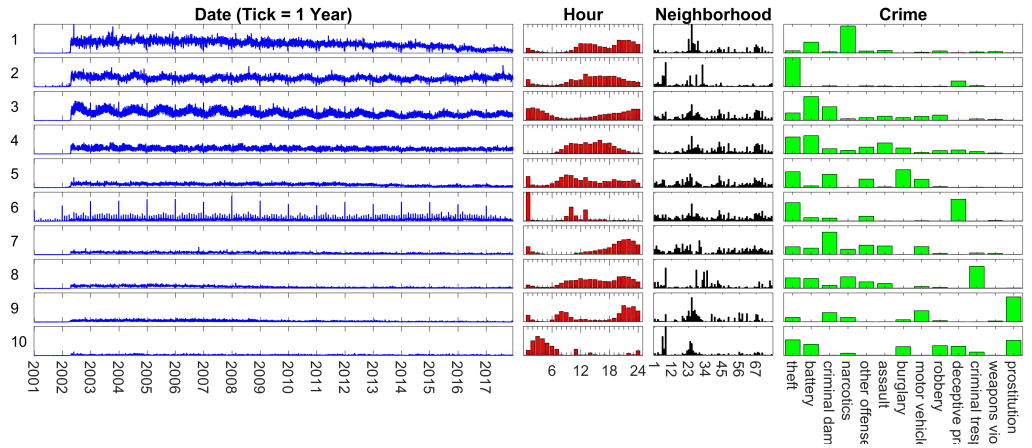
We tested the stochastic gradient using Adam [25] for GCP tensor decomposition and made several findings. Stochastic gradient methods are effective in practice in terms of driving down the objective function and recovering the true solutions. Empirically, we find that the number of samples should be roughly equal to the sum of the dimensions, i.e., $s = \sum_{k=1}^{d} n_k = d\bar{n}$. This is much lower than would be required to cycle through the entire dataset each epoch, i.e., n^d divided by the number of iterations per epoch. For dense problems, stochastic gradient methods can be much faster than the non-stochastic prior approach using L-BFGS-B [22]. For sparse problems, stochastic gradients enable us to circumvent formation of the dense tensor needed by the gradient, making it possible to solve much larger problems. Additionally, we propose stratified and semi-stratified sampling, which are typically superior to uniform sampling. We have not discussed how to determine the rank since that is a difficult problem even for standard CP.



(a) CP-APR (non-stochastic)



(b) GCP-Adam with stratified sampler



(c) GCP-Adam with semi-stratified sampler

Figure 5.7: Visualization of factorizations of the Chicago crime tensor as produced by three different methods.

Overall, stochastic methods have proved to be a promising approach for GCP tensor decomposition, especially for large-scale sparse tensors which have no viable alternative. However, many open questions remain. We can likely further improve the results by using more sophisticated stochastic optimization methods, e.g., weight decay strategies in Adam [30]. Likewise, more sophisticated sampling strategies such as leverage score sampling from matrix sketching [33, 55, 10] may further improve performance by reducing the variance. Another important line of investigation is developing appropriate theory to describe the improvement gains of the stratified approaches.

In terms of implementations, an interesting consequence of sampling in the context of parallel tensor decomposition [44, 24, 29, 40] is that we can reduce the computation and/or communication by sampling only a subset of the entries. Moreover, we may be able to stratify the samples in such a way that is amenable to more structured communications.

Appendix A. Special Cases where Gradient Does Not Require Dense Calculations. Computing the gradient is not a major issue for standard CP due to its special structure. Specifically, the computation of $\mathbf{G}_k$ can be simplified so that the primary work is computing $\mathbf{X}_{(k)}\mathbf{Z}_k$, which is a sparse MTTKRP whenever $\mathcal{X}$ is sparse [22, Appendix A]. For Poisson CP [12], the primary work is computing $\mathbf{V}_{(k)}\mathbf{Z}_k$ where $\mathcal{V}$ is the sparse tensor defined as

$$v_i = \begin{cases} x_i/m_i & \text{if } x_i \neq 0, \\ 0 & \text{otherwise.} \end{cases}$$

In these cases, the gradient can be computed in $O(\text{nnz}(\mathcal{X})\, rd)$ flops with $O(\text{nnz}(\mathcal{X}))$ additional storage. Generally, however, we may not have such structure and we have to compute with a dense $\mathcal{Y}$ tensor at a cost of $O(rn^d)$ flops and $O(\text{nnz}(\mathcal{X}))$ extra storage.

Appendix B. Determining the Oversampling Rate. Subsection 4.2 mentions that we oversample to get sufficiently many zeros with high probability. Namely, we sample

$$\rho \frac{n^d}{n^d - \text{nnz}(\mathcal{X})}\, s_{\text{zero}} = \rho \frac{1}{1 - \text{nnz}(\mathcal{X})/n^d}\, s_{\text{zero}} = \rho \frac{s_{\text{zero}}}{p_{\text{zero}}}$$

indices to get the desired s_{zero} zeros, where $p_{\text{zero}} = 1 - \text{nnz}(\mathcal{X})/n^d$ is the proportion of zeros in the tensor. Here we discuss the oversampling rate ρ.

We can use the inverse cumulative distribution function (CDF) of a negative binomial distribution to determine an appropriate ρ. The negative binomial distribution models the number of failures before a given number of successes with a specified success rate. In our case, we want s_{zero} successes and the success rate is p_{zero}. We can use the inverse CDF to determine the number of rejections at the 99.9999% percentile. For instance, in MATLAB:

$$s_{\text{reject}} = \texttt{icdf('Negative Binomial', 0.999999,}\ s_{\text{zero}}\texttt{,}\ p_{\text{zero}}\texttt{)}.$$

This means that with probability 0.999999, no more than s_{reject} nonzeros will be drawn before s_{zero} zeros are obtained. So we want to choose the oversampling rate ρ so that

$$\rho \geq (s_{\text{zero}} + s_{\text{reject}}) \frac{p_{\text{zero}}}{s_{\text{zero}}} = \frac{s_{\text{zero}} + s_{\text{reject}}}{s_{\text{zero}}}\, p_{\text{zero}}.$$

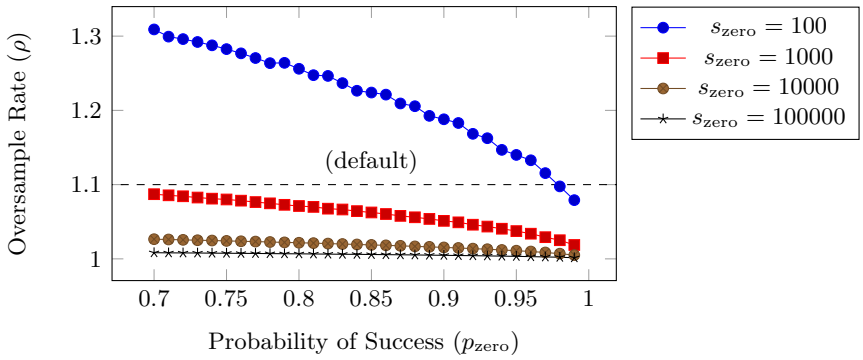


Figure B.1: Required oversampling rate for 99.9999% probability of generating at least s (non-rejected) samples.

In Figure B.1, we plot the oversample rate that would be needed in different scenarios. The x-axis is the proportion of nonzeros. The y-axis is the oversample rate. We plot four lines corresponding to different values for s_{zero}. We observe two trends:

1. For fixed p_{zero}, ρ decreases as s_{zero} increases, and
2. For fixed s_{zero}, ρ decreases as p_{zero} increases.

For most real-world examples of sampling zeros from a sparse tensor, $p_{\text{zero}} \geq 0.99$ because the tensors are extremely sparse. Additionally, we usually use a sample size of at least $s_{\text{zero}} = 1000$. Thus, oversampling by $\rho = 1.1$ should be adequate for most scenarios we expect to encounter.

We could also determine the oversampling rate for any individual problem using this procedure, but the inverse CDF calculation can be expensive.

Appendix C. Creating Gamma-Distributed Test Problems. To create the Gamma-distributed test problem used in subsection 5.2, we generate factor matrices whose entries are drawn from the uniform distribution on $(0, 1)$:

$$\mathbf{A}_k(i_k, j) \sim \text{uniform}(0, 1) \quad \text{for all} \quad i_k = 1, \ldots, n_k,\ j = 1, \ldots, r, \text{ and } k = 1, \ldots, d.$$

Using these factor matrices, we create $\mathcal{M}_{\text{true}}$. The data tensor $\mathcal{X}$ is generated as

$$x_i \sim \text{gamma}(k, m_i) \quad \text{with} \quad k = 1.$$

Appendix D. Details of Creating Binary Test Problems. We assume an odds link with the data, so the factor matrices must be nonnegative. The probability of a one is given by $m/(1+m)$ where m corresponds to the odds. For simplicity in the model and in generating the data tensor, we assume that factors 1 through $(r-1)$ are relatively sparse (i.e., sparsity specified by $\delta \in (0, 1/2)$ and factor r is dense. The idea here is the last dense component corresponds to noise in the model, i.e., random but infrequent observations of ones. Otherwise, the ones have a pattern as dictated by the sparse components.

We specify a density of factor matrix nonzeros and a probability of a one for nonzero values in the resulting model, denoted ρ_{high}. To obtain that probability, the nonzero factor matrix entries should be $\sqrt[d]{\rho_{\text{high}}/(1-\rho_{\text{high}})}$. We modify that slightly by setting the nonzero

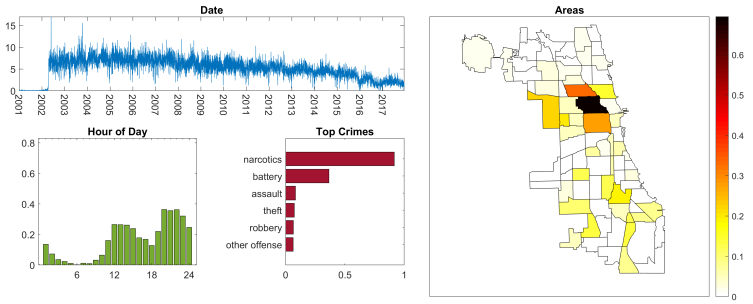


Figure F.1: Component 1 for Chicago crime tensor using semi-stratified sampling.

factor matrix entries to be drawn from a normal distribution with the mean as the target value and a standard deviation of 0.5. Since only a few entries are nonzero, we can identify all the possible nonzeros corresponding to the first four factors and then compute the exact probability computed by the model and then generate an observation.

For the final dense component, we want the probability of a one, denoted ρ_{low}, to be relatively low. This means that approximately ρ_{low} of the data tensor entries will correspond to this last "noise" column. The entries of the factor matrix are set to $\sqrt[d]{\rho_{\text{low}}/(1-\rho_{\text{low}})}$. We use this value exactly so that we can generate nonzero "noise" observations in bulk.

For the test problems in subsection 5.2, we use $\delta = 0.15$, $\rho_{\text{high}} = 0.9$, and $\rho_{\text{low}} = 0.0025$. For the test problems in subsection 5.5, we use $\delta = 0.15$, $\rho_{\text{high}} = 0.9$, and $\rho_{\text{low}} = 0.002$.

Appendix E. Cosine Similarity Score. If the true factor matrices are known, we can compute a cosine similarity score between the true and recovered solutions. If the true solution is denoted by $\mathbf{A}_k$ and the estimated solution is $\hat{\mathbf{A}}_k$, then the cosine similarity score is

$$\frac{1}{r}\sum_{j=1}^{r}\prod_{k=1}^{d}\cos(\mathbf{a}_k(:,j),\hat{\mathbf{a}}_k(:,\pi(j)))$$

where π is a permutation that should yield the highest possible similarity. Recall that the cosine of two vectors $\mathbf{a}$ and $\mathbf{b}$ is $\mathbf{a}^\intercal\mathbf{b}/(\|\mathbf{a}\|_2\|\mathbf{b}\|_2)$. We say that the true solution is recovered if the similarity score is at least 0.9. Assume that $\mathcal{M}$ holds $\hat{\mathbf{A}}_k$ and $\mathcal{M}_{\text{true}}$ holds $\mathbf{A}_k$, the cosine similarity is computed using the Tensor Toolbox for MATLAB [4] via the following command:

```
score(M,Mtrue,'lambda_penalty',false)
```

Appendix F. Individual components of Chicago crime tensor factorization. In this appendix, we show the remaining 10 components for the factorization of the Chicago crime tensor discussed in subsection 5.6 in Figures F.1 to F.10. Here we have scaled the date to show the overall weight of the component, and the other components are normalized.

Appendix G. Acknowledgments. We thank the referees for their helpful comments on an earlier draft of this paper, including bringing additional references to our attention. This paper was the result of a broader multi-year collaboration involving several institutions. We gratefully acknowledge Jed Duersch for preliminary investigations on handling large-scale tensors

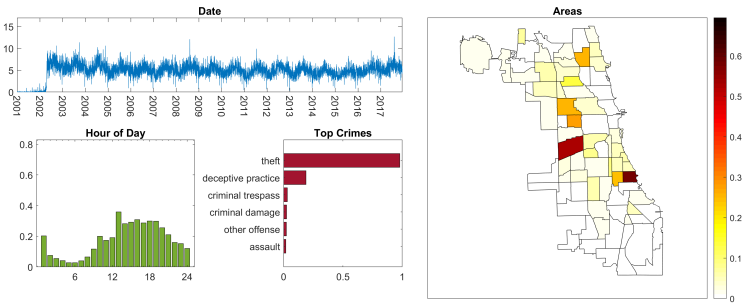


Figure F.2: Component 2 for Chicago crime tensor using semi-stratified sampling.

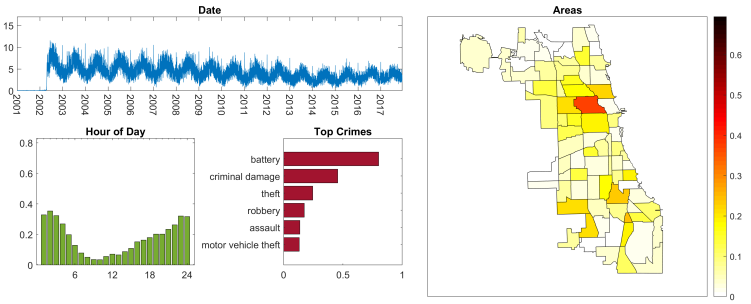


Figure F.3: Component 3 for Chicago crime tensor using semi-stratified sampling.

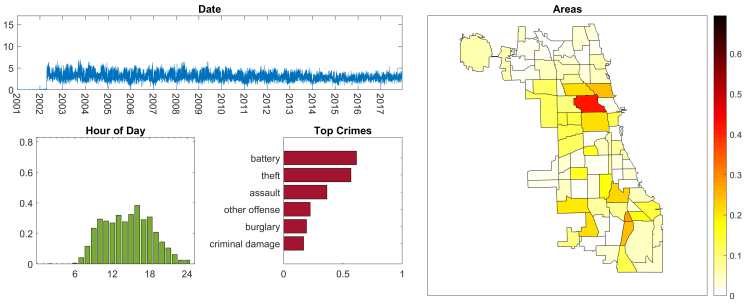


Figure F.4: Component 4 for Chicago crime tensor using semi-stratified sampling.

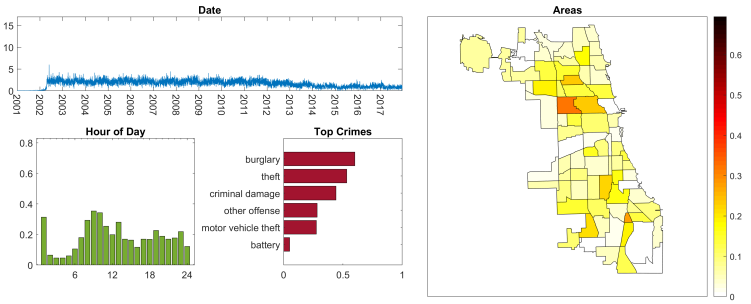


Figure F.5: Component 5 for Chicago crime tensor using semi-stratified sampling.

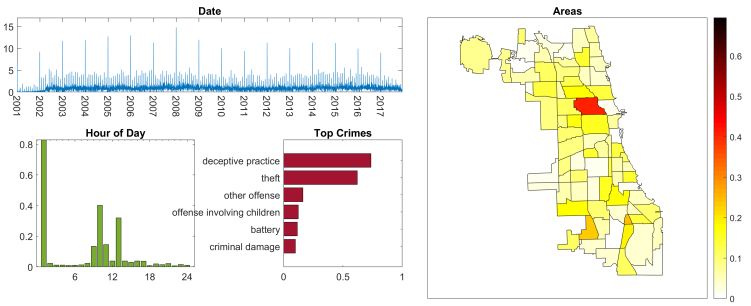


Figure F.6: Component 6 for Chicago crime tensor using semi-stratified sampling.

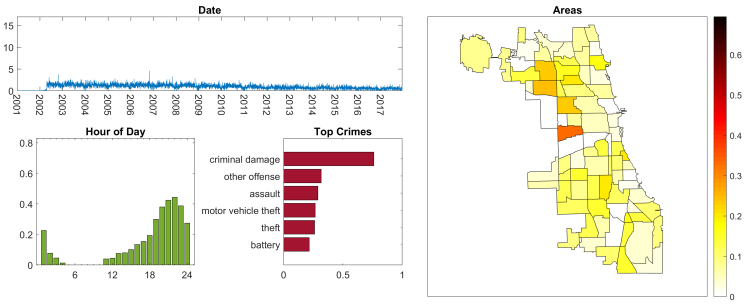


Figure F.7: Component 7 for Chicago crime tensor using semi-stratified sampling.

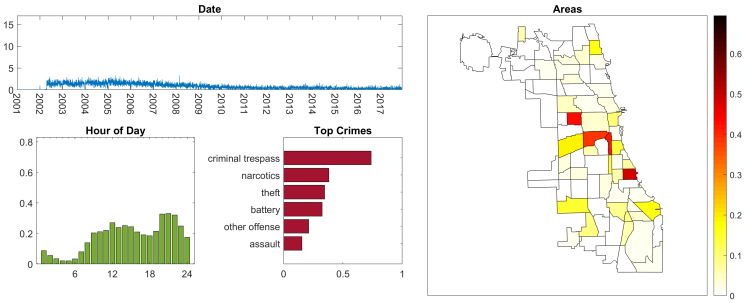


Figure F.8: Component 8 for Chicago crime tensor using semi-stratified sampling.

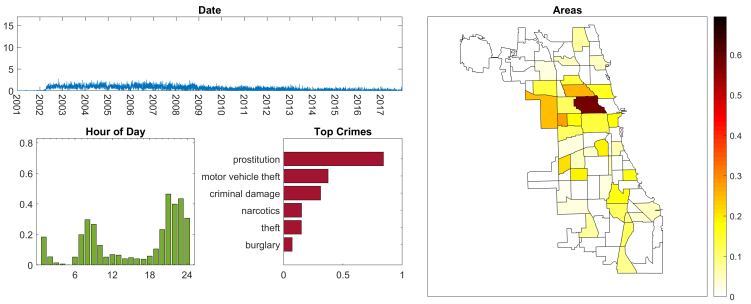


Figure F.9: Component 9 for Chicago crime tensor using semi-stratified sampling.

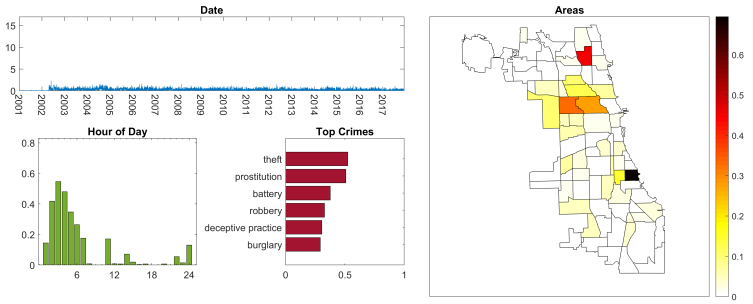


Figure F.10: Component 10 for Chicago crime tensor using semi-stratified sampling.

with GCP and many insightful discussions about the work presented in this paper. Thanks also to Cliff Anderson-Bergman for stimulating discussions about this project, including providing the information for Appendix B. We thank our colleague Eric Phipps for proposing an idea that ultimately led to semi-stratified sampling.

REFERENCES

[1] E. Acar, D. M. Dunlavy, and T. G. Kolda, *A scalable optimization approach for fitting canonical tensor decompositions*, Journal of Chemometrics, 25 (2011), pp. 67–86, doi:10.1002/cem.1335.

[2] E. Acar, D. M. Dunlavy, T. G. Kolda, and M. Mørup, *Scalable tensor factorizations for incomplete data*, Chemometrics and Intelligent Laboratory Systems, 106 (2011), pp. 41–56, doi:10.1016/j.chemolab.2010.08.004.

[3] B. W. Bader and T. G. Kolda, *Efficient MATLAB computations with sparse and factored tensors*, SIAM Journal on Scientific Computing, 30 (2007), pp. 205–231, doi:10.1137/060676489.

[4] B. W. Bader, T. G. Kolda, et al., *MATLAB Tensor Toolbox Version, Version 3.1*. Available online, June 2019, https://www.tensortoolbox.org.

[5] G. Ballard, N. Knight, and K. Rouse, *Communication lower bounds for matricized tensor times khatri-rao product*, in 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS), IEEE, May 2018, doi:10.1109/ipdps.2018.00065.

[6] C. Battaglino, G. Ballard, and T. G. Kolda, *A practical randomized CP tensor decomposition*, SIAM Journal on Matrix Analysis and Applications, 39 (2018), pp. 876–901, doi:10.1137/17M1112303, arXiv:1701.06600.

[7] A. Beutel, P. P. Talukdar, A. Kumar, C. Faloutsos, E. E. Papalexakis, and E. P. Xing, *FlexiFaCT: Scalable flexible factorization of coupled tensors on hadoop*, in SDM'14: Proceedings of the 2014 SIAM International Conference on Data Mining, Apr. 2014, pp. 109–117, doi:10.1137/1.9781611973440.13, http://dx.doi.org/10.1137/1.9781611973440.13.

[8] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, *A limited memory algorithm for bound constrained optimization*, SIAM J. Sci. Comput., 16 (1995), pp. 1190–1208, doi:10.1137/0916069.

[9] J. D. Carroll and J. J. Chang, *Analysis of individual differences in multidimensional scaling via an N-way generalization of "Eckart-Young" decomposition*, Psychometrika, 35 (1970), pp. 283–319, doi:10.1007/BF02310791.

[10] Y. Chen, S. Bhojanapalli, S. Sanghavi, and R. Ward, *Completing any low-rank matrix, provably*, Journal of Machine Learning Research, 16 (2015), pp. 2999–3034, http://www.jmlr.org/papers/v16/chen15b.html.

[11] D. Cheng, R. Peng, I. Perros, and Y. Liu, *SPALS: Fast alternating least squares via implicit leverage scores sampling*, in NIPS'16, 2016, https://papers.nips.cc/paper/6436-spals-fast-alternating-least-squares-via-implicit-leverage-scores-sampling.pdf.

[12] E. C. Chi and T. G. Kolda, *On tensors, sparsity, and nonnegative factorizations*, SIAM Journal on

Matrix Analysis and Applications, 33 (2012), pp. 1272–1299, doi:10.1137/110859063.

[13] R. N. Cochran and F. H. Horne, *Statistically weighted principal component analysis of rapid scanning wavelength kinetics experiments*, Analytical Chemistry, 49 (1977), pp. 846–853, doi:10.1021/ac50014a045.

[14] R. Ge, F. Huang, C. Jin, and Y. Yuan, *Escaping from saddle points — online stochastic gradient for tensor decomposition*, in Conference on Learning Theory, 2015, pp. 797–842, http://proceedings.mlr.press/v40/Ge15.pdf.

[15] R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis, *Large-scale matrix factorization with distributed stochastic gradient descent*, in KDD'11: Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM Press, 2011, doi:10.1145/2020408.2020426.

[16] S. Gopal, *Adaptive sampling for SGD by exploiting side information*, in Proceedings of The 33rd International Conference on Machine Learning, M. F. Balcan and K. Q. Weinberger, eds., vol. 48 of Proceedings of Machine Learning Research, New York, New York, USA, 20–22 Jun 2016, PMLR, pp. 364–372, http://proceedings.mlr.press/v48/gopal16.html.

[17] E. Gujral, R. Pasricha, and E. E. Papalexakis, *SamBaTen: Sampling-based batch incremental tensor decomposition*, in Proceedings of the 2018 SIAM International Conference on Data Mining, 2018, pp. 387–395, doi:10.1137/1.9781611975321.44.

[18] S. Hansen, T. Plantenga, and T. G. Kolda, *Newton-based optimization for Kullback-Leibler nonnegative tensor factorizations*, Optimization Methods and Software, 30 (2015), pp. 1002–1029, doi:10.1080/10556788.2015.1009977, arXiv:1304.4964.

[19] R. A. Harshman, *Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis*, UCLA working papers in phonetics, 16 (1970), pp. 1–84. Available at http://www.psychology.uwo.ca/faculty/harshman/wpppfac0.pdf.

[20] K. Hayashi, G. Ballard, Y. Jiang, and M. J. Tobia, *Shared-memory parallelization of MTTKRP for dense tensors*, in Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming - PPoPP'18, ACM Press, 2018, doi:10.1145/3178487.3178522.

[21] D. Hong, J. A. Fessler, and L. Balzano, *Optimally weighted pca for high-dimensional heteroscedastic data*, arXiv:http://arxiv.org/abs/1810.12862v2 [math.ST].

[22] D. Hong, T. G. Kolda, and J. A. Duersch, *Generalized canonical polyadic tensor decomposition*, SIAM Review, 62 (2020), pp. 133–163, doi:10.1137/18M1203626, arXiv:1808.07452.

[23] J. J. Jansen, H. C. J. Hoefsloot, H. F. M. Boelens, J. van der Greef, and A. K. Smilde, *Analysis of longitudinal metabolomics data*, Bioinformatics, 20 (2004), pp. 2438–2446, doi:10.1093/bioinformatics/bth268.

[24] O. Kaya and B. Uçar, *Scalable sparse tensor decompositions in distributed memory systems*, in SC'15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2015, doi:10.1145/2807591.2807624.

[25] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2015, arXiv:1412.6980v9 [cs.LG]. Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[26] T. G. Kolda and B. W. Bader, *Tensor decompositions and applications*, SIAM Review, 51 (2009), pp. 455–500, doi:10.1137/07070111X.

[27] T. G. Kolda, A. Pinar, T. Plantenga, and C. Seshadhri, *A scalable generative graph model with community structure*, SIAM Journal on Scientific Computing, 36 (2014), pp. C424–C452, doi:10.1137/130914218.

[28] Y. Koren, R. Bell, and C. Volinsky, *Matrix factorization techniques for recommender systems*, Computer, 42 (2009), pp. 30–37, doi:10.1109/MC.2009.263.

[29] J. Li, J. Choi, I. Perros, J. Sun, and R. Vuduc, *Model-driven sparse CP decomposition for higher-order tensors*, in 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS), IEEE, 2017, pp. 1048–1057, doi:10.1109/ipdps.2017.80.

[30] I. Loshchilov and F. Hutter, *Fixing weight decay regularization in Adam*, 2017, arXiv:1711.05101v2 [cs.LG].

[31] C. Ma, X. Yang, and H. Wang, *Randomized online CP decomposition*, in Proc. Tenth Int. Conf. Advanced Computational Intelligence (ICACI), Mar. 2018, pp. 414–419, doi:10.1109/ICACI.2018.8377495.

[32] T. Maehara, K. Hayashi, and K.-i. Kawarabayashi, *Expected tensor decomposition with stochastic gradient descent*, in AAAI, 2016, pp. 1919–1925.
[33] M. W. Mahoney, *Randomized algorithms for matrices and data*, arXiv:1104.5557v3 [cs.DS].
[34] M. Mardani, G. Mateos, and G. B. Giannakis, *Subspace learning and imputation for streaming big data matrices and tensors*, IEEE Transactions on Signal Processing, 63 (2015), pp. 2663–2677, doi:10.1109/tsp.2015.2417491.
[35] B. Marlin, R. S. Zemel, S. Roweis, and M. Slaney, *Collaborative filtering and the missing at random assumption*, in Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence (UAI'07), 2007, pp. 267–275, arXiv:1206.5267v1.
[36] D. Needell, N. Srebro, and R. Ward, *Stochastic gradient descent, weighted sampling, and the randomized Kaczmarz algorithm*, Mathematical Programming, 155 (2015), pp. 549–573, doi:10.1007/s10107-015-0864-7.
[37] D. Nion and N. D. Sidiropoulos, *Adaptive algorithms to track the PARAFAC decomposition of a third-order tensor*, IEEE Transactions on Signal Processing, 57 (2009), pp. 2299–2310, doi:10.1109/TSP.2009.2016885.
[38] E. E. Papalexakis, C. Faloutsos, and N. D. Sidiropoulos, *ParCube: Sparse parallelizable tensor decompositions.*, in Machine Learning and Knowledge Discovery in Databases (European Conference, ECML PKDD 2012), vol. 7523 of Lecture Notes in Computer Science, Springer, 2012, pp. 521–536, doi:10.1007/978-3-642-33460-3_39.
[39] A.-H. Phan, P. Tichavsky, and A. Cichocki, *Fast alternating LS algorithms for high order CANDECOMP/PARAFAC tensor factorizations*, IEEE Transactions on Signal Processing, 61 (2013), pp. 4834–4846, doi:10.1109/TSP.2013.2269903.
[40] E. Phipps and T. G. Kolda, *Software for sparse tensor decomposition on emerging computing architectures*, SIAM Journal on Scientific Computing, 41 (2019), pp. C269–C290, doi:10.1137/18M1210691, arXiv:1809.09175.
[41] N. D. Sidiropoulos, E. E. Papalexakis, and C. Faloutsos, *A parallel algorithm for big tensor decomposition using randomly compressed cubes (PARACOMP)*, in 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, May 2014, doi:10.1109/icassp.2014.6853546.
[42] S. Smith, J. W. Choi, J. Li, R. Vuduc, J. Park, X. Liu, and G. Karypis, *FROSTT: The formidable repository of open sparse tensors and tools*, 2017, http://frostt.io/.
[43] S. Smith, J. Park, and G. Karypis, *An exploration of optimization algorithms for high performance tensor completion*, in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '16, IEEE Press, 2016, pp. 31:1–31:13, doi:10.1109/sc.2016.30, http://dl.acm.org/citation.cfm?id=3014904.3014946.
[44] S. Smith, N. Ravindran, N. D. Sidiropoulos, and G. Karypis, *SPLATT: Efficient and parallel sparse tensor-matrix multiplication*, in IPDPS 2015: IEEE International Parallel and Distributed Processing Symposium, May 2015, pp. 61–70, doi:10.1109/ipdps.2015.27.
[45] Z. Song, D. P. Woodruff, and H. Zhang, *Sublinear time orthogonal tensor decomposition*, in Advances in Neural Information Processing Systems (NIPS) 30, 2016, https://papers.nips.cc/paper/6495-sublinear-time-orthogonal-tensor-decomposition.pdf.
[46] N. Srebro and T. Jaakkola, *Weighted low-rank approximations*, in IMCL-2003: Proceedings of the Twentieth International Conference on Machine Learning, 2003, pp. 720–727, https://www.aaai.org/Papers/ICML/2003/ICML03-094.pdf.
[47] O. Tamuz, T. Mazeh, and S. Zucker, *Correcting systematic effects in a large set of photometric light curves*, Monthly Notices of the Royal Astronomical Society, 356 (2005), pp. 1466–1470, doi:10.1111/j.1365-2966.2004.08585.x.
[48] M. Udell, C. Horn, R. Zadeh, and S. Boyd, *Generalized low rank models*, FNT in Machine Learning, 9 (2016), pp. 1–118, doi:10.1561/2200000055, http://dx.doi.org/10.1561/2200000055.
[49] M. Vandecappelle, N. Vervliet, and L. D. Lathauwer, *Nonlinear least squares updating of the canonical polyadic decomposition*, in 2017 25th European Signal Processing Conference (EUSIPCO), IEEE, aug 2017, pp. 663–667, doi:10.23919/EUSIPCO.2017.8081290.
[50] A. Vergara, J. Fonollosa, J. Mahiques, M. Trincavelli, N. Rulkov, and R. Huerta, *On the performance of gas sensor arrays in open sampling systems using inhibitory support vector machines*,

Sensors and Actuators B: Chemical, 185 (2013), pp. 462 – 477, doi:http://dx.doi.org/10.1016/j.snb.2013.05.027.

[51] N. Vervliet and L. De Lathauwer, *A randomized block sampling approach to canonical polyadic decomposition of large-scale tensors*, IEEE J. Sel. Top. Signal Process., 10 (2016), pp. 284–295, doi:10.1109/JSTSP.2015.2503260.

[52] N. Vervliet, O. Debals, and L. De Lathauwer, *Tensorlab 3.0 — numerical optimization strategies for large-scale constrained and coupled matrix/tensor factorization*, in 2016 50th Asilomar Conference on Signals, Systems and Computers, Nov 2016, pp. 1733–1738, doi:10.1109/ACSSC.2016.7869679.

[53] Y. Wang, H.-Y. Tung, A. J. Smola, and A. Anandkumar, *Fast and guaranteed tensor decomposition via sketching*, in Advances in Neural Information Processing Systems (NIPS) 28, 2015, pp. 991–999, http://papers.nips.cc/paper/5944-fast-and-guaranteed-tensor-decomposition-via-sketching.pdf.

[54] M. Welling and M. Weber, *Positive tensor factorization*, Pattern Recognition Letters, 22 (2001), pp. 1255–1261, doi:10.1016/S0167-8655(01)00070-8.

[55] D. P. Woodruff, *Sketching as a tool for numerical linear algebra*, FNT in Theoretical Computer Science, 10 (2014), pp. 1–157, doi:10.1561/0400000060, arXiv:1411.4357.

[56] H. H. Yue and M. Tomoyasu, *Weighted principal component analysis and its applications to improve FDC performance*, in 2004 43rd IEEE Conference on Decision and Control (CDC) (IEEE Cat. No.04CH37601), IEEE, 2004, doi:10.1109/cdc.2004.1429421.

[57] P. Zhao and T. Zhang, *Accelerating minibatch stochastic gradient descent using stratified sampling*, 2014, arXiv:1405.3080v1 [stat.ML].

[58] P. Zhao and T. Zhang, *Stochastic optimization with importance sampling for regularized loss minimization*, in Proceedings of the 32nd International Conference on Machine Learning, 2015, pp. 1–9, http://proceedings.mlr.press/v37/zhaoa15.html.

[59] G. Zhou, A. Cichocki, and S. Xie, *Decomposition of big tensors with low multilinear rank*, Dec. 2014, arXiv:1412.1885.