(2024-09-01; PRE-ARXIV) EFFICIENT 1-BIT TENSOR APPROXIMATIONS

ALEX W. N. RIASANOVSKY AND SARAH EL KAZDADI

Abstract. We present a spatially efficient decomposition of matrices and arbitrary-order tensors as linear combinations of tensor products of $\{-1,1\}$-valued vectors. For any matrix $A \in \mathbb{R}^{m\times n}$,

$$A - R_w = S_w C_w T_w^\top = \sum_{j=1}^{w} c_j \cdot \boldsymbol{s}_j \boldsymbol{t}_j^\top$$

is a *w-width signed cut decomposition of A*. Here $C_w = \operatorname{diag}(\boldsymbol{c}_w)$ for some $\boldsymbol{c}_w \in \mathbb{R}^w$, and S_w, T_w, and the vectors $\boldsymbol{s}_j, \boldsymbol{t}_j$ are $\{-1,1\}$-valued.

To store (S_w, T_w, C_w), we may pack $w \cdot (m+n)$ bits, and require only w floating (or fixed-) point numbers. As a function of w, $\|R_w\|_F$ decays exponentially when applied to `f32` matrices with i.i.d. $\mathcal{N}(0,1)$ entries. Choosing w so that (S_w, T_w, C_w) has the same memory footprint as a `f16` or `bf16` matrix, the error is comparable.

As a first application, we approximate the weight matrices in the open `Mistral-7B-v0.1` Large Language Model to a 50% spatial compression. Remarkably, all 226 remainder matrices have a relative error $< 6\%$ and the expanded model closely matches `Mistral-7B-v0.1` on the `HuggingFace` leaderboard. Benchmark performance degrades slowly as we reduce the spatial compression from 50% to 25%.

Our algorithm yields efficient signed cut decompositions in 20 lines of pseudocode. It reflects a simple modification from a 1999 paper of Frieze and Kannan. This paper, touted as a foundation of the field of combinatorial limit theory, appears underutilized in randomized numerical linear algebra.

We optimize our open source `Rust` implementation with `SIMD` instructions on `avx2` and `avx512` architectures. We also extend our algorithm from matrices to tensors of arbitrary order and use it to compress a picture of the first author's cat Angus.

1. Introduction

In a recent survey [1], Murray et al offer RNLA as a treatment for large-scale problems in high performance computing (HPC) and machine learning (ML).

> *A dire situation.* While communities that rely on NLA now vary widely, they share one essential property: a ravenous appetite for solving larger and larger problems.
>
> — Murray et al [1]

Throughout the survey, the authors emphasize the emergence of structure at scale, which randomized algorithms are uniquely adept at exploiting.

The story in combinatorial limit theory is similar. Here, the *cut norm* $\|\cdot\|_\square$, first coined in 1999 in [2] by Frieze and Kannan, plays a vital role in exposing emergent structure in large networks. Using the cut norm, they approximate matrices with linear combinations of rank-1 matrices with entries in $\{0,1\}$:

$$\|A\|_{\square} := \max_{S,T}\left|\sum_{(i,j)\in S\times T} a_{ij}\right| \rightsquigarrow A \approx \sum_j d_j \mathbf{1}_{S_j\times T_j}.$$

Following [2], researchers at the Theory Group of Microsoft Research applied these so-called *Frieze-Kannan decompositions* to connect several central, but superficially different, topics in combinatorics.

Although Frieze-Kannan decompositions are spatially efficient, it is challenging to efficiently minimize their errors (c.f. Section 2.1). As a remedy, we base our methods around the signed cut norm $\|\cdot\|_{\blacksquare}$ instead of $\|\cdot\|_{\square}$, and instead approximate with outer products of $\{-1,1\}$-valued vectors:

$$\|A\|_{\blacksquare} := \max_{\boldsymbol{s},\boldsymbol{t}}\langle \boldsymbol{s}A, \boldsymbol{t}\rangle \rightsquigarrow A \approx \sum_j c_j \boldsymbol{s}_j \boldsymbol{t}_j^{\top}.$$

Summarizing Section 4, we argue that these *signed cut decompositions* produce state-of-the-art approximations. As noted in Section 3, the main algorithms we use to find them fit in 20 lines of pseudocode.

1.1. **Comparison to `bf16` quantization.** Our signed cut decompositions trade accuracy for space at a better exchange rate than `bf16` quantization offers for larger matrices, as shown in Figure 1. Moreover, we may make this tradeoff dynamically at runtime to suit the needs of our application.

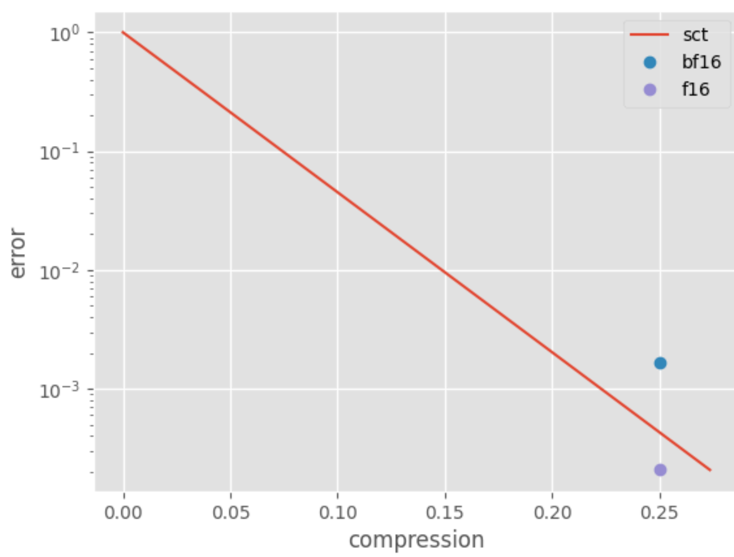

FIGURE 1. Relative error (log-scaled) and relative size of a signed cut decompositions of a 4096×4096 `f64` matrix with i.i.d. standard normal entries. For comparison, we note the tradeoff made with `f16` and `bf16`.

1.2. **Approximating `Mistral-7B-v0.1`.** When the original matrix is more structured, signed cut decompositions converge faster. In Figure 2, we show the effect of signed cut decompositions on the open `Mistral-7B-v0.1` model's benchmark performance.

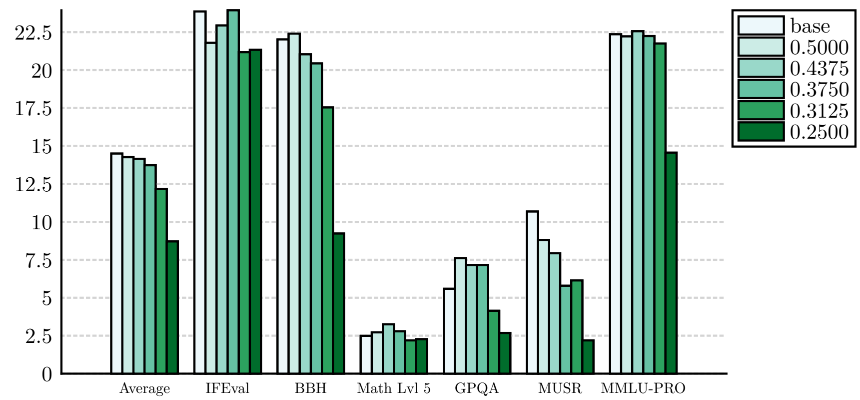

FIGURE 2. Benchmark results formed by approximating the weight matrices of `Mistral-7B-v0.1` with signed cut decompositions and expanding them at different widths. Here, 0.500 reflects a 2-fold spatial compression, and 0.2500 reflects a 4-fold spatial compression.

1.3. **Higher order tensors.** All of our techniques extend from matrices to tensors of arbitrary order, as show in Section 5. In Figure 3, we compress a picture the first author's cat by treating the implicit array of `RGB` values as an order-3 tensor.

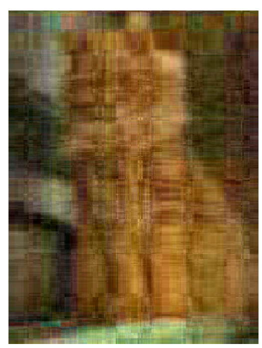

FIGURE 3. A picture of Angus and a very thin signed cut decomposition.

2. CUT NORMS AND DECOMPOSITIONS

2.1. **The unsigned cut norm and Frieze-Kannan decompositions.** We adapt techniques from combinatorial limit theory, a subfield of combinatorics cultivated by the Theory Group of Microsoft Research in the early 2000s. The germ of this theory is the *cut norm*, coined originally by Frieze and Kannan [2], which connects cut-set problems, matrix approximations, and the celebrated Regularity Lemma of Szemerédi. For any matrix $A = \left(a_{ij}\right) \in \mathbb{R}^{m\times n}$, the *cut norm* of A is

$$\|A\|_{\square} := \max_{S,T}\left|\sum_{(i,j)\in S\times T} a_{ij}\right| = \max_{S,T}|\langle \mathbf{1}_{S\times T}, A\rangle|,$$

where S (T) is any set of rows (columns) of A. In other words, $\|\cdot\|_\square$ measures the largest absolute sum over all submatrices of A. Although $\|A\|_\square$ is `MAX SNP hard`, it can theoretically be approximated efficiently in polynomial time (c.f. [3], [4]). As a motivating question, suppose we wish to optimize Frieze-Kannan decompositions on error, space, and time. As a function of the width w, we define the program

$$\mathrm{FK}_w(A) : \begin{cases} \min & \|A - \sum_{j=1}^{w} d_j \cdot \mathbf{1}_{S_j \times T_j}\|_F \\ \text{s.t.} & d_1, ..., d_w \in \mathbb{R} \\ & S_1, ..., S_w \subseteq \{1, ..., m\} \\ & T_1, ..., T_w \subseteq \{1, ..., n\} \end{cases}.$$

In [2], the authors approach $\mathrm{FK}_w(A)$ with randomized algorithms which estimate the cut norm of a residual matrix and use the result to perform a rank-1 update on the remainder matrix. Their algorithms are complicated by two unhelpful facts:

1. $\mathrm{FK}_1(A)$ is not in general solved by calculating $\|A\|_\square$. The optimal approximation $A \approx d \cdot \mathbf{1}_{S\times T}$ has $d = \langle \mathbf{1}_{S\times T}, A\rangle \cdot (|S| \cdot |T|)^{-1}$ for some other S, T.
2. Pairs of matrices of the form $\mathbf{1}_{S\times T}$ are typically far from orthogonal. If we reuse our solution to $\mathrm{FK}_w(A)$ to solve $\mathrm{FK}_{w+1}(A)$, then these correlations force us in general to apply large updates to the scalars $d_1, ..., d_w$.

2.2. **Signed cut decompositions and the signed cut norm.** By instead basing our algorithms around the *signed cut norm* $\|\cdot\|_{\blacksquare}$, we mitigate the issues from Section 2.1 in practice. For any matrix $A \in \mathbb{R}^{m\times n}$, let

$$\|A\|_{\blacksquare} := \max_{s,t} \langle s, At\rangle = \max_{x,y} \langle x, Ay\rangle = \|A\|_{\infty\to 1}$$

where s, t are $\{-1, 1\}$-valued and x, y are $[-1, 1]$-valued. As show in [3], calculating $\|\cdot\|_{\blacksquare}$ is also `MAX SNP hard`. We approximate it with a randomized greedy algorithm. We define a *signed cut decomposition of width* w as any expression

$$A - R_w = S_w C_w T_w^\top = \sum_{j=1}^{w} c_j \cdot \boldsymbol{s}_j \boldsymbol{t}_j^\top$$

where $R_w \in \mathbb{R}^{m\times n}$, $S_w \in \{-1, 1\}^{m\times w}$, $C_w = \mathrm{diag}(\boldsymbol{c}_w)$ for some $\boldsymbol{c}_w \in \mathbb{R}^w$, and $T_w \in \{-1, 1\}^{n\times w}$. As noted in Table 1, signed cut decompositions maintain the space efficiency of Frieze-Kannan decompositions while more closely emulating singular value decompositions (see Section 4).

Decomposition	Rank-1 terms	Vector entries	Size (bits)
Singular value	$\sigma_j \cdot \boldsymbol{u}_j \boldsymbol{v}_j^\top$	$\mathbb{R}$	$(m + n + 1) \cdot f$
Frieze-Kannan	$d_j \cdot \mathbf{1}_{S_j\times T_j}$	$\{0, 1\}$	$m + n + f$
Signed cut	$c_j \cdot \boldsymbol{s}_j \boldsymbol{t}_j^\top$	$\{-1, 1\}$	$m + n + f$

TABLE 1. A comparison of three matrix decompositions equivalent to finite sums of rank-1 matrices. Here, a floating point number occupies f bits in memory.

3. METHODS

For convenience, we let $\sigma(n) := \{-1, 1\}^n$. Our analog to $\mathrm{FK}_w(A)$ is the program

$$\mathrm{SC}_w(A) : \begin{cases} \min & \|A - \sum_{j=1}^{w} c_j \cdot \boldsymbol{s}_j \boldsymbol{t}_j^\top\|_F \\ \text{s.t.} & c_1, \ldots, c_w \in \mathbb{R} \\ & \boldsymbol{s}_1, \ldots, \boldsymbol{s}_w \in \sigma(m) \\ & \boldsymbol{t}_1, \ldots, \boldsymbol{t}_w \in \sigma(n) \end{cases}.$$

Our two main algorithms are straightforward and greedy. We include a few optimizations within each subsection and defer others (e.g., `SIMD`, bitset storage, and alignment) to Section 3.4. Algorithm 4 estimates $\|\cdot\|_{\blacksquare}$ iteratively. Algorithm 5 bootstraps these estimates to extend approximate solutions from $\mathrm{SC}_k(A)$ to $\mathrm{SC}_{k+1}(A)$. Finally, Algorithm 6 adjusts the coefficients to improve the regression. In Section 5, we generalize to tensors.

3.1. **Naive signed cut-sets.** Note that $\|A\|_{\blacksquare}$ is the solution to the program

$$\mathrm{SgnCut}(A) : \begin{cases} \max & \langle \boldsymbol{s}, A\boldsymbol{t} \rangle \\ \text{s.t.} & \boldsymbol{s} \in \sigma(m) \\ & \boldsymbol{t} \in \sigma(n) \end{cases}.$$

In [2], the authors explore the analogous program for the cut norm. By replacing $\boldsymbol{s}$ with $\mathrm{sgn}(A\boldsymbol{t})$, or $\boldsymbol{t}$ with $\mathrm{sgn}(A^\top \boldsymbol{s})$, we see that $\mathrm{Cut}(A)$ is equivalent to the programs

$$\mathrm{LeftSgnCut}(A) : \begin{cases} \max & \|A^\top \boldsymbol{s}\|_1 \\ \text{s.t.} & \boldsymbol{s} \in \sigma(m) \end{cases} \qquad \mathrm{RightSgnCut}(A) : \begin{cases} \max & \|A\boldsymbol{t}\|_1 \\ \text{s.t.} & \boldsymbol{t} \in \sigma(n) \end{cases}.$$

Algorithm 4 exploits this to approach a locally maximal cut by flipping signs.

<u>GREEDY SIGNED CUT</u>($A \in \mathbb{R}^{m \times n}$):

sample $\boldsymbol{s}_0 \in \sigma(m)$ uniformly
sample $\boldsymbol{t}_0 \in \sigma(n)$ uniformly
let $c_0 := -\infty$
for $j \in \{1, 2, \ldots\}$**:**
    let $\boldsymbol{s}_j := \mathrm{sgn}(A \cdot \boldsymbol{t}_{j-1})$
    let $\boldsymbol{t}_j := \mathrm{sgn}(A^\top \cdot \boldsymbol{s}_j)$
    let $c_j := \langle \boldsymbol{s}_j, A \cdot \boldsymbol{t}_j \rangle$
    if $c_j \leq c_{j-1}$**:**
        yeet $(c_j, \boldsymbol{s}_j, \boldsymbol{t}_j)$

ALGORITHM 4. A randomized greedy algorithm for $\mathrm{Cut}(A)$.

By caching the intermediate vectors $A \cdot \boldsymbol{t}_j, A^\top \boldsymbol{s}_j$, we may sparsify the matrix-vector products. To see this, note that at iteration $j \in \{2, 3, \ldots\}$,

$$\boldsymbol{s}_j = \mathrm{sgn}(A \cdot \boldsymbol{t}_{j-1}) = \mathrm{sgn}(A \cdot \boldsymbol{t}_{j-2} + A \cdot (\boldsymbol{t}_{j-1} - \boldsymbol{t}_{j-2})), \text{ and}$$
$$\boldsymbol{t}_j = \mathrm{sgn}(A^\top \cdot \boldsymbol{s}_{j-1}) = \mathrm{sgn}(A^\top \cdot \boldsymbol{s}_{j-2} + A^\top \cdot (\boldsymbol{s}_{j-1} - \boldsymbol{s}_{j-2})).$$

In practice, as j increases, the $\{-2,0,2\}$-valued vectors $\boldsymbol{s}_j - \boldsymbol{s}_{j-1}$ and $\boldsymbol{t}_j - \boldsymbol{t}_{j-1}$ become increasing sparse. We store the variables as follows:

- **sign bitsets**: the $\{-1,1\}$-valued vectors $\boldsymbol{s}_j$ and $\boldsymbol{t}_j$ are stored with unsigned integers. Here, each 1 bit indicating a -1. At iteration j, we also store $\boldsymbol{s}_{j-1}$ and $\boldsymbol{t}_{j-1}$.
- **contiguous, aligned vectors**: the real-valued vectors $A \cdot \boldsymbol{t}_{j-1}$ and $A^\top \cdot \boldsymbol{s}_j$ are stored with aligned slices of `f32` values. The matrices A and $A^\top$ are aligned, padded, and in column-major.

3.2. **Simple signed cut decompositions.** Algorithm 5 builds signed cut decompositions by repeatedly invoking Algorithm 4.

<u>GREEDY DECOMPOSITION</u>($A \in \mathbb{R}^{m\times n}$, $w \in \mathbb{N}$):

let $R_0 := A$
let $S_0 := 0 \in \{-1,1\}^{m\times 0}$
let $\boldsymbol{c}_0 := 0 \in \mathbb{R}^0$
let $T_0 := 0 \in \{-1,1\}^{n\times 0}$
for $k \in \{0, ..., w-1\}$**:**
  let $(c_{k+1}, \boldsymbol{s}_{k+1}, \boldsymbol{t}_{k+1}) =$ `greedy_signed_cut`(R_k)
  form $\boldsymbol{c}_{k+1}$ by extending $\boldsymbol{c}_k$ with c_{k+1}
  form S_{k+1} by extending S_k with $\boldsymbol{s}_{k+1}$
  form T_{k+1} by extending T_k with $\boldsymbol{t}_{k+1}$
  let $R_{k+1} := R_k - \frac{c_{k+1}}{mn} \cdot \boldsymbol{s}_{k+1}\boldsymbol{t}_{k+1}^\top$
yeet $(S_w, \mathrm{diag}(\boldsymbol{c}_w), T_w)$

ALGORITHM 5. A "greedy" least-squares approach to signed cut decompositions. The remainder R_{k+1} is the orthogonal complement of R_k with respect to the 1-dimensional subspace of $\mathbb{R}^{m\times n}$ spanned by $\boldsymbol{s}_{k+1}\boldsymbol{t}_{k+1}^\top$.

By inspection,

$$\|R_{k+1}\|_F^2 = \|R_k\|_F^2 - \frac{c_{k+1}^2}{mn}.$$

for all $0 \le k < w$. If c_k is proportional to $\|R_{k-1}\|_F$, then $\|R_k\|_F$ decreases exponentially. Empirically, results in Section 4 affirm this hypothesis.

To amortizes the cost of the rank-1 updates to R_k, we delay the `matmul` accumulation in Algorithm 5 by storing each R_k implicitly as (R', S, C, T) where $R_k = R' + SCT^\top$. When (S, C, T) reaches a carefully chosen width (32), we flush $SCT^\top$ into R' to form R_k. We store all real-valued vectors and matrices (except the diagonal matrices) as in Section 3.1. For the $\{-1,1\}$-valued matrices, we also use bitsets with the 1 bit indicating a -1 sign.

3.3. **Least square corrections.**

There is a simple improvement that can be made to Algorithm 5 based on the method of least squares. In the k-th iteration of Algorithm 6, we treat S_k and T_k as fixed and allow the diagonal matrix C_k to vary so as to minimize $\|A - S_k C_k T_k^\top\|_F$.

<u>Least Squares Decomposition</u>($A \in \mathbb{R}^{m\times n}$, $w \in \mathbb{N}$):

let $S_0 := 0 \in \{-1,1\}^{m\times 0}$
let $\boldsymbol{c}_0 := 0 \in \mathbb{R}^0$
let $T_0 := 0 \in \{-1,1\}^{n\times 0}$
for $k \in \{0, ..., w-1\}$**:**
    let $R_k := S_k \operatorname{diag}(\boldsymbol{c}_k) T_k^\top$
    let $(\_\_, \boldsymbol{s}_{k+1}, \boldsymbol{t}_{k+1}) =$ `greedy_signed_cut`(R_k)
    form S_{k+1} by extending S_k with $\boldsymbol{s}_{k+1}$
    form T_{k+1} by extending T_k with $\boldsymbol{t}_{k+1}$
    let $\boldsymbol{c}_{k+1}$ minimize $\|A - S_{k+1} \operatorname{diag}(\boldsymbol{c}_{k+1}) T_{k+1}^\top\|_F$
yeet $(S_w, \operatorname{diag}(\boldsymbol{c}_w), T_w)$

Algorithm 6. An improvement to Algorithm 5. Here, R_{k+1} is the orthogonal complement of A with respect to the span of $\boldsymbol{s}_1\boldsymbol{t}_1^\top, ..., \boldsymbol{s}_{k+1}\boldsymbol{t}_{k+1}^\top$.

To calculate $\boldsymbol{c}_{k+1}$ in Algorithm 6, we formulate the associated least-squares problem and solve its *normal equation*

$$X_k^\top X_{k+1} \cdot \boldsymbol{c}_{k+1} = X_{k+1}^\top \cdot A.$$

Here, X_{k+1} is the linear operator from $\mathbb{R}^k$ to $\mathbb{R}^{m\times n}$ defined by sending j-th standard basis vector to $\boldsymbol{s}_j\boldsymbol{t}_j^\top$.

In our experiments, the improvement from swapping Algorithm 5 for Algorithm 6 is not significant for many matrices. We offer two possible explanations:

1. If only c_{k+1} varies, then Algorithm 5 and Algorithm 6 are equivalent.
2. The more orthogonal the matrices $\boldsymbol{s}_k\boldsymbol{t}_k^\top$ are, the less Algorithm 6 helps.

A lengthier analysis goes beyond the scope of this paper.

3.4. **`SIMD` Optimizations.** We improve the implementation in [5] by leveraging `SIMD` intrinsics on both `avx2` and `avx512` architectures. Interestingly, neither Algorithm 4 nor Algorithm 5 makes a single call to `fused multiply add` instructions. To demonstrate this, suppose we wish to compute the inner product $\langle \boldsymbol{s}, \boldsymbol{x}\rangle$ where

$$\boldsymbol{s} = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{bmatrix} \in \sigma(n) \quad \text{and} \quad \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^n \quad .$$

Assuming $n = 8\ell$ for some ℓ, we apply the isometry $\mathbb{R}^n \cong \bigoplus_{j=1}^{\ell} \mathbb{R}^8$ and write

$$\begin{aligned}
\boldsymbol{s} &= \boldsymbol{s}_1 \oplus \cdots \oplus \boldsymbol{s}_\ell && \text{where each } \boldsymbol{s}_j \in \sigma(8), \text{ and} \\
\boldsymbol{x} &= \boldsymbol{x}_1 \oplus \cdots \oplus \boldsymbol{x}_\ell && \text{where each } \boldsymbol{x}_j \in \mathbb{R}^8, \text{ so that} \\
\langle \boldsymbol{s}, \boldsymbol{x} \rangle &= \langle \boldsymbol{s}_1, \boldsymbol{x}_1 \rangle + \cdots + \langle \boldsymbol{s}_\ell, \boldsymbol{x}_\ell \rangle.
\end{aligned}$$

Since our signed sets are stored as bitsets, we need only use the bits of each $\boldsymbol{s}_j$ to flip the sign of each $\boldsymbol{x}_j$ entry and then sum. On `avx512` instructions (and the upcoming `avx10` instructions), we may offload this task to `mask` instructions with 512-bit registers. On `avx2` and `aarch64`, we may instead broadcast 32 bits to smaller `f32` registers, then use shifts instructions to move bits of interest to flip sign bits.

4. Results

4.1. **Approximating a random matrix.** In this section, we evaluate the quality of signed cut decompositions by approximating a random matrix $A \in \mathbb{R}^{4096\times 4096}$ with independent $\mathcal{N}(0,1)$ entries. As an array of `f64` numbers, this occupies $8 \cdot 4096 \cdot 4096$ bytes, or $64 \cdot 4096 \cdot 4096$ bits, in memory. In Figure 7, we consider the intermediate signed cut decompositions $A = R_k + \sum_{j=1}^{k} c_j \boldsymbol{s}_j \boldsymbol{t}_j^\top$. We store each triple $\left(c_j, \boldsymbol{s}_j, \boldsymbol{t}_j\right)$ so the scalar has `f64` precision and the signed vectors $\boldsymbol{s}_j, \boldsymbol{t}_j$ as a bitset, as in Section 3. This way, each triple can be stored with $64 + m + n$ bits in memory. For each k, we plot (p_k, r_k) where

$$p_k := \frac{k \cdot (64 + 4096 + 4096)}{64 \cdot 4096 \cdot 4096} \quad \text{and } r_k := \frac{\|R_k\|_F}{\|A\|_F}.$$

This way, p_k is the *compression rate* of the k-width approximation of A. For comparison, we plot $(0.25, r)$ where r is the relative error from downcasting A to `bf16` or `f16` precision.

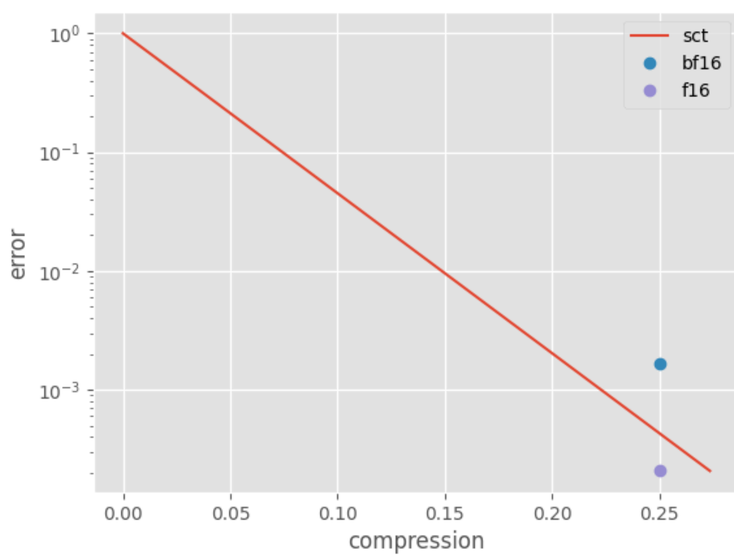

FIGURE 7. Relative error (log-scale) plotted against spatial compression rate for a standard normal 4096×4096 matrix with `f64` precision. Here, `bf16` has error ≈ 0.00166 and `f16` has error ≈ 0.000208. Signed cut decompositions breaks even on error at 0.2064 and 0.2734, respectively.

4.2. **Approximating `Mistral-7B-v0.1`.** In this section, we apply signed cut decompositions to approximate the open `Mistral-7B-v0.1` model.

4.2.1. *Frobenius norm error.* We repeat our experiment in Section 4.1 by forming signed cut decompositions of the 226 weight matrices from the open `Mistral-7B-v0.1` model. Since these matrices are stored with `bf16` precision, each such $A \in \mathbb{R}^{m\times n}$ occupies $16mn$ bits in the original `safetensors` file. In the approximation $A \approx \sum_{j=1}^{k} c_j \boldsymbol{s}_j \boldsymbol{t}_j^\top$, we store each scalar c_j with `f32` precision and store each vector sign vector $\boldsymbol{s}_j, \boldsymbol{t}_j$ as a bitset, as described in Section 3. So the summands together occupy $32 + m + n$ bits in memory.

Targeting a 2-fold spatial compression, our approximation $A \approx \sum_{j=1}^{w} c_j \boldsymbol{s}_j \boldsymbol{t}_j^\top$ has width

$$w \approx \frac{1}{2} \cdot \frac{16mn}{32 + m + n}.$$

In Figure 8, we plot the relative error as a function of the compression rate.

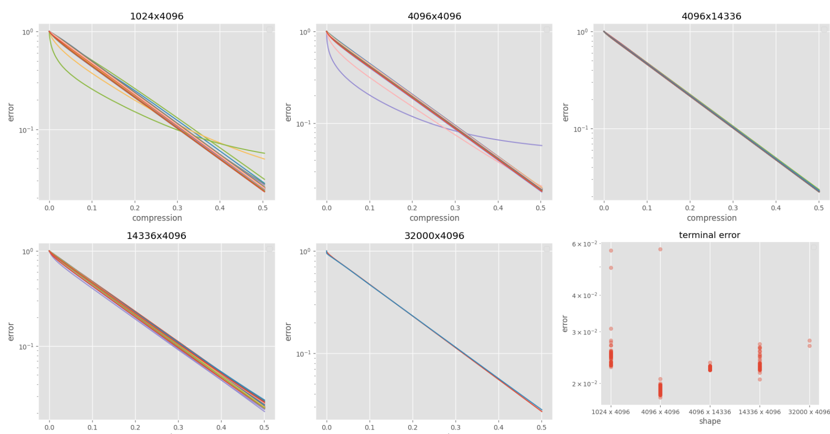

FIGURE 8. Relative error $\frac{\|R_k\|_F}{\|A\|_F}$ (log-scale) of signed cut decompositions of every matrix in `Mistral-7b-v0.1` after running Algorithm 5. Here, we group by matrix shape. The final width corresponds to a 50% spatial compression from the original `bf16` matrix.

4.2.2. *Large language model benchmarks.* Using $\frac{j}{16}$ for $j \in \{4, 5, 6, 7, 8\}$ as our target compression rate, we select appropriate widths based on the matrix dimensions m, n as in Table 2.

Shape	**Count**	**Width by compression**				
		0.5000	0.4375	0.3750	0.3125	0.2500
1024×4096	64	6512	5698	4884	4070	3256

Shape	Count	Width by compression				
		0.5000	0.4375	0.3750	0.3125	0.2500
4096×4096	64	16320	14280	12240	10200	8160
4096×14336	32	25442	22261	19081	15901	12721
14336×4096	64	25442	22261	19081	15901	12721
32000×4096	2	29023	25395	21767	18139	14511

TABLE 2. Widths matching each shape to each desired compression rate.

We expand the corresponding truncated signed cut approximations and write new `safetensors` files, reusing all other values. In order to measure the quality of signed cut decompositions as a vehicle for model quantization, we truncate the weight matrix approximations $A \approx \sum_{j=1}^{w} c_j \boldsymbol{s}_j \boldsymbol{t}_j^\top$ at different weights corresponding to our target compression rates. With m, n fixed, a w-width signed cut approximation of at `bf16` $m \times n$ matrix has compression rate

$$p_{m,n}(w) = \frac{w \cdot (32 + m + n)}{16mn}.$$

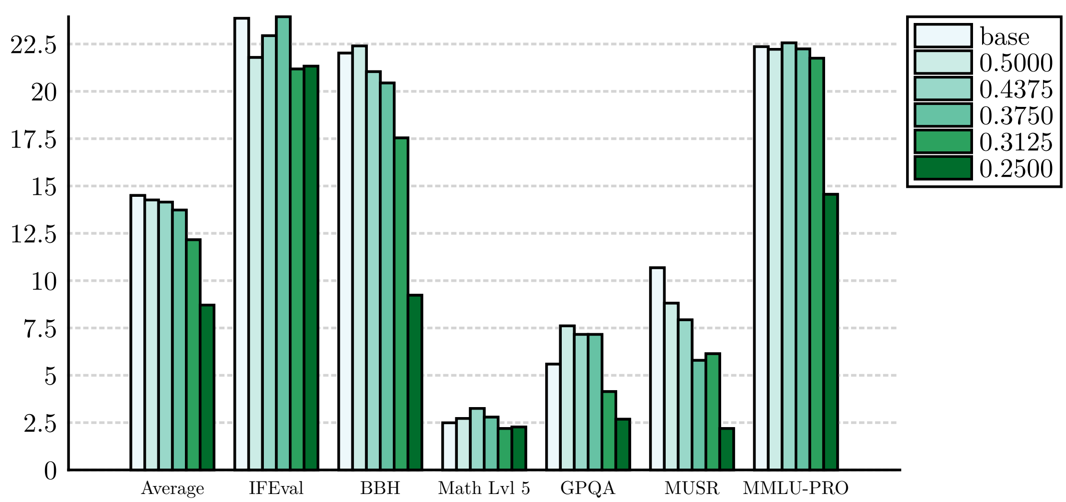

FIGURE 9. Benchmark performance of signed cut decompositions of the open `Mistral-7B-v0.1` model at compression rates between 0.5 and 0.25.

5. GENERALIZATIONS

As mentioned in Section 1, we motivate the tensor signed cut decomposition by approximating a picture of the first author's cat Angus, as in Figure 10.

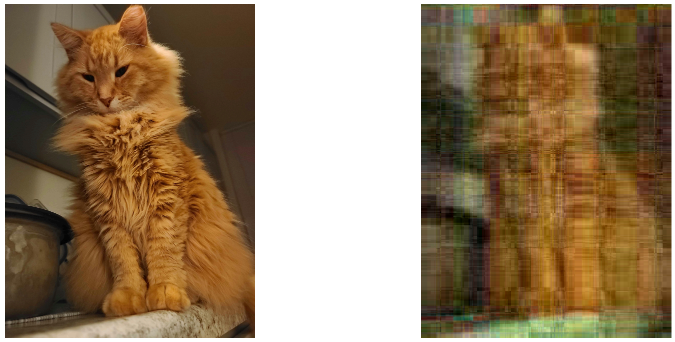

FIGURE 10. A picture of Angus and a very thin signed cut decomposition.

To begin, we convert this `jpg` image to a 4000×3000 array of `RGB` triples, with each value stored as an unsigned 8-bit integer. As a tensor, we may write $\boldsymbol{a} \in \mathbb{R}^{4000\times3000\times3}$, with axes corresponding to rows, columns, and colors, respectively. Our goal is to optimize an approximation of the form

$$\boldsymbol{a} \approx \boldsymbol{a}_w = \sum_{j=1}^{w} c_j \cdot \boldsymbol{x}_j \otimes \boldsymbol{y}_j \otimes \boldsymbol{k}_j.$$

By using `f32` precision for the scalars, and bitsets for the sign vectors, each quadruple $\left(c_j, \boldsymbol{x}_j, \boldsymbol{y}_j, \boldsymbol{k}_j\right)$, occupies $4 + 500 + 375 + 0.375 = 879.375$ bytes of memory.

When evaluating this approximation, it is natural to first round the entries of $\boldsymbol{a}_w$ to the nearest value in $\{0, ..., 255\}$. To account for this, we define

$$R(i) := \left\{ x \in \mathbb{R} : |x - i| = \min_{j \in \{0,...,255\}} |x - j| \right\} \text{ for all } i \in \{0, ..., 255\}.$$

In particular, $R(0) = (-\infty, 0.5]$, $R(255) = [254.5, \infty)$, and for all $j \in \{1, ..., 254\}$, $R(j) = \left[j - \frac{1}{2}, j + \frac{1}{2}\right]$. In general, for all `RGB` image arrays $\boldsymbol{a} \in \{0, ..., 255\}^{m\times n\times 3}$, the set of tensors which round to $\boldsymbol{a}$ is

$$\mathcal{R}(\boldsymbol{a}) := \left\{ \boldsymbol{b} \in \mathbb{R}^{m\times n\times 3} : \text{ for all } (i, j, k) \in [m] \times [n] \times [3], \ (\boldsymbol{b})_{ijk} \in R\Big((\boldsymbol{a})_{ijk}\Big) \right\}.$$

Then for all w, we let

$$\mathrm{RGBSgnCut}_w(\boldsymbol{a}) : \begin{cases} \min & \left\| \boldsymbol{b} - \sum_{j=1}^{w} c_j \cdot \boldsymbol{x}_j \otimes \boldsymbol{y}_j \otimes \boldsymbol{k}_j \right\|_2 \\ \text{s.t.} & \boldsymbol{b} \in \mathcal{R}(\boldsymbol{a}) \\ & c_1, ..., c_w \in \mathbb{R} \\ & \boldsymbol{x}_1, ..., \boldsymbol{x}_w \in \sigma(m) \\ & \boldsymbol{y}_1, ..., \boldsymbol{y}_w \in \sigma(n) \\ & \boldsymbol{k}_1, ..., \boldsymbol{k}_w \in \sigma(3) \end{cases}.$$

Pixel-wise, this means that estimating $i \in \{0, ..., 255\}$ with $z \in \mathbb{R}$ contributes error

$$\rho(i,x) = \begin{cases} \max\bigl(0, x - \frac{1}{2}\bigr) & \text{if } i = 0 \\ \max\bigl(0, 255 - x - \frac{1}{2}\bigr) & \text{if } i = 255 \\ \max\bigl(0, |x - i| - \frac{1}{2}\bigr) & \text{else} \end{cases}.$$

5.1. **Signed cuts.** For our purposes, an order-k tensor is an array $\boldsymbol{a} \in \mathbb{R}^{\otimes \boldsymbol{n}}$ where $\boldsymbol{n} = (n_1, ..., n_k) \in \mathbb{N}^k$. We define the signed cut norm over $\mathbb{R}^{\otimes \boldsymbol{n}}$ as follows. First, let

$$\operatorname{sgn}(\boldsymbol{n}) := \{-1, 1\}^{\otimes \boldsymbol{n}} = \bigotimes_{i=1}^{k} \operatorname{sgn}(n_i).$$

Here, $\operatorname{sgn}(\boldsymbol{n})$ is a $2^{n_1 + \cdots + n_k}$-element frame spanning $\mathbb{R}^{\otimes \boldsymbol{n}}$. Then for all $\boldsymbol{a} \in \mathbb{R}^{\otimes \boldsymbol{n}}$,

$$\|\boldsymbol{a}\|_{\blacksquare} := \max_{\boldsymbol{s} \in \operatorname{sgn}(\boldsymbol{n})} \langle \boldsymbol{s}, \boldsymbol{a} \rangle.$$

Generalizing from Section 3.1, we note that for all $\boldsymbol{n} \in \mathbb{N}^k$ and all $1 \leq j \leq k$, the programs

$$\operatorname{TensorSgnCut}(A) : \begin{cases} \max & \langle \boldsymbol{s}, \boldsymbol{a} \rangle \\ \text{s.t.} & \boldsymbol{s} \in \sigma(\boldsymbol{n}) \end{cases} \quad \text{and}$$

$$\operatorname{AxialCut}_i(A) : \begin{cases} \max & \|\boldsymbol{a} \times_i \boldsymbol{s}\|_1 \\ \text{s.t.} & \boldsymbol{s} \in \sigma\Bigl(\bigl(n_j\bigr)_{j \neq i}\Bigr) \end{cases}$$

both solve for $\|\boldsymbol{a}\|_{\blacksquare}$. Here, $\boldsymbol{a} \times_i \boldsymbol{s} \in \mathbb{R}^{n_j}$ is the product of $\boldsymbol{a}$ and $\boldsymbol{s}$ along the i-th axis. We extend Algorithm 4 from Section 3.1 to Algorithm 11.

<u>Axial greedy cut</u>($\boldsymbol{a} \in \mathbb{R}^{\otimes \boldsymbol{n}}$):

sample $\boldsymbol{s}_0 = \bigotimes_{i \in \mathcal{A}} \boldsymbol{s}_{0i} \in \sigma(\boldsymbol{n})$ uniformly
let $c_0 := -\infty$
for $j \in \{1, 2, ...\}$**:**
    for $i \in \{1, ..., k\}$**:**
        let $\boldsymbol{s}_{j\downarrow i} := \bigotimes_{i' < i} \boldsymbol{s}_{ji}$
        let $\boldsymbol{s}_{j\uparrow i} := \bigotimes_{i' > i} \boldsymbol{s}_{j-1,i}$
        let $\boldsymbol{s}_{j\updownarrow i} := \boldsymbol{s}_{j\downarrow i} \otimes \boldsymbol{s}_{j\uparrow i}$
        let $\boldsymbol{s}_{ji} := \operatorname{sgn}\bigl(\boldsymbol{a} \times_i \boldsymbol{s}_{j\updownarrow i}\bigr)$
    let $\boldsymbol{s}_j := \bigotimes_{i \in \mathcal{A}} \boldsymbol{s}_{ji}$
    let $c_j := \langle \boldsymbol{s}_j, \boldsymbol{a} \rangle$
    if $c_j \leq c_{j-1}$**:**
        yeet $\bigl(c_j, \boldsymbol{s}_j\bigr)$

Algorithm 11. Algorithm 4 for tensors.

5.2. **Signed cut decompositions.** Here, for any $\boldsymbol{a} \in \mathbb{R}^{\otimes \boldsymbol{n}}$, a *signed cut decomposition of width w* is any solution to

$$\boldsymbol{a} - \boldsymbol{r}_w = S_w \cdot \boldsymbol{c}_w = \sum_{j=1}^{w} c_j \cdot \boldsymbol{s}_j$$

where each $\boldsymbol{s}_j \in \sigma(\boldsymbol{n})$, S_w is a linear operator from $\mathbb{R}^w$ to $\mathbb{R}^{\otimes \boldsymbol{n}}$ which sends the j-th standard basis vector to $\boldsymbol{s}_j$, $\boldsymbol{c}_w = (c_j) \in \mathbb{R}^w$, and $\boldsymbol{r}_w \in \mathbb{R}^{\otimes \boldsymbol{n}}$. Continuing, we extend Algorithm 5 from Section 3.2 to Algorithm 12.

<u>Tensor greedy decomposition</u>($\boldsymbol{a} \in \mathbb{R}^{\otimes \boldsymbol{n}}$, $w \in \mathbb{N}$):

let $\boldsymbol{r}_0 := \boldsymbol{a}$
let $S_0 := 0 \in \sigma(\boldsymbol{n}) \otimes \mathbb{R}^0$
let $\boldsymbol{c}_0 := 0 \in \mathbb{R}^0$
for $k \in \{0, ..., w-1\}$**:**
    let $(c_{k+1}, \boldsymbol{s}_{k+1}) = \texttt{axial\_greedy\_cut}(\boldsymbol{r}_k)$
    form $\boldsymbol{c}_{k+1}$ by extending $\boldsymbol{c}_k$ with c_{k+1}
    form S_{k+1} by extending S_k with $\boldsymbol{s}_{k+1}$
    let $\boldsymbol{r}_{k+1} := \boldsymbol{r}_k - \frac{c_{k+1}}{\prod_i n_i} \cdot \boldsymbol{s}_{k+1}$
yeet $(\boldsymbol{c}_w, S_w)$

Algorithm 12. Algorithm 5 for tensors.

5.3. **Least square corrections.** Finally, we extend Algorithm 6 from Section 3.3 to Algorithm 13.

<u>Tensor least Squares</u>($\boldsymbol{a} \in \mathbb{R}^{\otimes \boldsymbol{n}}$, $w \in \mathbb{N}$):

let $S_0 := 0 \in \sigma(\boldsymbol{n}) \otimes \mathbb{R}^0$
let $\boldsymbol{c}_0 := 0 \in \mathbb{R}^0$
for $k \in \{0, ..., w-1\}$**:**
    let $\boldsymbol{r}_k := \boldsymbol{a}_k - S_k \cdot \boldsymbol{c}_k$
    let $(\_\_, \boldsymbol{s}_{k+1}) = \texttt{axial\_greedy\_cut}(\boldsymbol{r}_k)$
    form S_{k+1} by extending S_k with $\boldsymbol{s}_{k+1}$
    let $\boldsymbol{c}_{k+1}$ minimize $\|\boldsymbol{a} - S_{k+1} \cdot \boldsymbol{c}_{k+1}\|_2$
yeet $(\boldsymbol{c}_w, S_w)$

Algorithm 13. Algorithm 6 for tensors.

6. Discussion and Future Work

We discuss some potential applications of signed cut decompositions for future work.

6.1. **High Performance Computing.** In Section 4.1, we showed that signed cut decompositions can approximate large random matrices roughly as well as `bf16` and `f16` quantization while matching the memory footprint. In Section 4.2, we also showed that they approximate highly structured low-precision matrices even after driving down the memory footprint by a factor of 2 or higher. How do these gains in space complexity yield better runtimes? More precisely, we ask:

Question A: How does the runtime of `matvec` (calculating $A \cdot x$) for matrix A and vector x compare when A is replaced by a w-width signed cut decomposition?

At this time, we lack the infrastructure and resources to fully answer this question. As noted in Section 3.4, our `matmul` implementation manipulate bits en lieu of making a call to `FMA` instructions.

6.2. **Machine Learning.** Recall from Section 4.2 that our experiments replace all 226 weight matrices from the open `Mistral-7B-v0.1` model with the result of expanding signed cut decompositions $\sum_{j=1}^{w} c_j \boldsymbol{s}_j \boldsymbol{t}_j^\top$ where the width w is chosen so as to emulate a predetermined compression rate. Based on Table 2, we see that our 2-fold compression consists of

$$64 \cdot 6{,}512 + 64 \cdot 16{,}320 + 64 \cdot 25{,}442 + 32 \cdot 25{,}442 + 2 \cdot 29023 = 3{,}961{,}726$$

scalars c_j (and the same number of row and column vectors $\boldsymbol{s}_j$ and $\boldsymbol{t}_j$). In other words:

Question B: If the $\boldsymbol{s}_j$ and $\boldsymbol{t}_j$ are fixed, can the `Mistral-7B-v0.1` model be efficiently retrained treating only the 3.96 million scalars as variables?

More broadly, we ask:

Question C: What is the best way to build machine learning architecture around signed vectors?

In a feed-forward neural network, some input vector $x \in \mathbb{R}^n$ is passed through hidden layers which pair affine linear transformations with activation functions:

$$x \mapsto Ax + b$$
$$y \mapsto \sigma(y).$$

We may instead calculate $YCZ \cdot x$ where Y, Z have $\{-1, 1\}$-valued entries and C is a diagonal matrix of scalars. To justify universal approximation (provided that Z is sufficiently tall), it is sufficient to note that $\mathbb{R}^k$ is spanned by $\sigma(k)$. Can we get away with only the "$CZ \cdot x$" steps, while maintaining universal approximation?

7. Acknowledgements

We thank the reviewers Alice, Bob, and Charlie for their helpful comments. Additionally, we thank Bernard Lidický for lending the TODO computing cluster to run our first prototype.

References

1. Murray, R., Demmel, J., Mahoney, M., Erichson, N., Melnichenko, M., Malik, O., Grigori, L., Luszczek, P., Dereziński, M., Lopes, M., others: Randomized Numerical Linear Algebra: A

Perspective on the Field With an Eye to Software (2023). DOI: https://doi. org/10.48550/ arXiv. 2302,

2. Frieze, A., Kannan, R.: Quick approximation to matrices and applications. Combinatorica. 19, 175–220 (1999). https://doi.org/10.1007/s004930050052
3. Alon, N., Naor, A.: Approximating the cut-norm via Grothendieck's inequality. In: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing. pp. 72–80 (2004)
4. Lovász, L., Szegedy, B.: Szemerédi's lemma for the analyst. GAFA Geometric And Functional Analysis. 17, 252–270 (2007)
5. Neal Riasanovsky, A.: cuts, https://github.com/ariasanovsky/cuts

Philadelphia, PA, USA
Email address: a.riasanovsky@gmail.com
URL: https://riasanovsky.me/