sumarry

Mohamed Hamlil

July 2024

1 Introduction

The theme of this work relates to matrix sparsification.

Matrix Sparsification Problem (MS):
Let $A \in \mathbb{R}^{n \times m}$. The objective is to find an invertible matrix U such that

$$\text{argmin}(\text{nnz}(AU))$$

(the number of non-zero elements in the product AU is minimized).

To address this problem, we will first review the main project (algorithms learned from the articles), then examine secondary projects, and finally discuss the distribution of work during the project's implementation.

2 Main Project

This section covers the mandatory work.

2.0.1 Description

In the article Sparsifying the Operators of Fast Matrix Multiplication Algorithms, the algorithm that solves the matrix sparsification problem uses an oracle algorithm (which solves the Sparsest Independent Vector problem).

Sparsest Independent Vector:
Let $U \in \mathbb{R}^{n \times m}$ (with $n \leq m$) and let $\Omega = \{\omega_1, \ldots, \omega_k\} \subseteq [m]$.

- v lies in the row space of U,
- v is not in the subspace spanned by $\{U_{\omega_1}, \ldots, U_{\omega_k}\}$,
- v has a minimal number of non-zero components.

Oracle algorithms:

Exact oracle algorithms:

The first algorithm explores all possibilities of vectors orthogonal to the combination set of A's (n-1) linearly independent row vectors(we can't choose n cause the only vector as a solution is the all zero vector so we go to n-1) until finding the one that best sparsifies the product AU, then moves to a second vector linearly independent of the first to construct the matrix U which consists of vectors that have sparsified A.

The second is an improvement on the first, adopting a top-down approach (starting from the largest set of row vectors), where the first vector that sparsifies A is necessarily the best, thus reducing computation time.

Heuristic methods:

The first is based on the observation that using rows from the original matrix for sparsification ensures that the sparsified matrix contains n rows, each with a single non-zero entry. Although this method is inefficient, requiring $\binom{m}{n}$ passes, it finds sparsities that significantly enhance dominant coefficients in several algorithms.

The second algorithm, not yet proven optimal, iterates over each row of matrix A, attempting to minimize the number of non-zeros and non-singletons in the product with a vector v linearly independent from previous solution vectors. The minimization problem is transformed into maxsat and a Z3 Solver is used to find the vector.

In another article Faster Matrix Multiplication via Sparse Decomposition, the problem is as follows:

Let $Q \in \mathbb{R}^{t \times n}$ and $r \in [t - n]$. We seek a decomposition of Q into $Q_\phi \in \mathbb{R}^{t \times (t-r)}$, $\phi \in \mathbb{R}^{(t-r) \times n}$, satisfying:

$$\begin{aligned} &\text{minimize :} \\ &\text{nnz}(Q_\phi) + \text{nns}(Q_\phi) \\ &\text{subject to :} \\ &\qquad Q = Q_\phi \cdot \phi \end{aligned}$$

Let P_π be the permutation matrix that permutes the rows of Q_ϕ so that the first $t - r$ rows contain the identity matrix.

$$P_\pi \cdot Q = P_\pi \cdot Q_\phi \cdot \phi = (\text{Identity Matrix } |:::) \cdot \phi$$

Thus, for all $i \in [t - r]$, $\phi_i = (P_\pi \cdot Q)_i$, and therefore ϕ is unique. The sparsity process works as follows:

1. Choose the positions of the identity matrix rows in Q_ϕ.
2. Compute ϕ based on this selection.

3. For each remaining row v_i of Q_ϕ, solve:

$$v_i = \arg\min_{\mathbf{x}\in\mathbb{R}^{t-r}} \left\{ \text{nnz}(\mathbf{x}) : \phi^T \cdot \mathbf{x}^T = (Q_i)^T \right\}$$

The problem has been transformed into a Compressed sensing problem, and algorithms such as Basis Pursuit and Orthogonal Matching Pursuit have been used, with their results approximating solutions to the Compressed Sensing problem. However, these algorithms are designed to solve other problems, and their approximated solutions do not guarantee optimality.

2.0.2 Implementation

Both exact methods rely on the existence of a vector λ (Ω validator).
Ω validator :
Let

$$S \subset [m]$$

$$\lambda \in \mathbb{R}^n \text{ with } \text{supp}(\lambda) \notin \Omega \text{ such that } \lambda^T \cdot U_{:,S} = 0$$

(where $\text{supp}(\lambda) = \{i : \lambda_i \neq 0\}$).

This vector λ serves as a validator for a set (Ω valid) and aids in finding the sparse vector.
The Ω valid set :

$S \subset [m]$ is Ω-valid if $\exists i \notin \Omega$ such that $U_{i,S}$ is in the span of rows $U_{[n]\setminus\{i\},S}$

Thus, the equivalence between the existence of λ and the valid Omega set is given by:

$$\lambda^T_{[n]/i} \cdot U_{[n]/i,S} = -\lambda_i \cdot U_{i,S}$$

This equivalence is used to compute λ while validating the chosen set, reducing computational cost. In the first oracle, multiple iterations are performed on the same set (the combination set of A's column vectors), checking if the chosen combination will yield a matrix of a defined rank, thereby eliminating non-conforming combinations from the outset.

For the final algorithm in the first article, an attempt was made to implement minimization in cases of infinite sets, but the program took too long for a simple problem.

For the algorithm in the second article, using exact methods to approximate their solutions to non-exact solutions is not an optimal method for solving the problem, especially with the existence of better methods as mentioned above.

3 Conclusion

The implemented algorithms are implementations of algorithms from the articles mentioned in this paper with small improvements.