The Mailman algorithm for matrix vector multiplication

Edo Liberty* Steven Zucker†

Abstract

Given an $m \times n$ matrix A we are interested in applying it to a real vector $x \in \mathbb{R}^n$ in less then the trivial $O(mn)$ time. Since we are interested in a deterministic exact computation we must, at the very least, access all the entrees in A. This requires $O(mn)$ operations and matches the running time of naively applying A to x. However, we claim that if the matrix contains only a constant number of distinct values, reading the matrix once in $O(mn)$ steps is sufficient to preprocess it such that any subsequent application to vectors requires $O(mn/\log(max\{m,n\}))$. We show that a straight forward usage of our algorithm can improve on recent results for dimensionality reduction algorithms.

Introduction

General matrix vector multiplication is clearly a very common an basic operation. An enormous amount of effort was put into accelerating the application of special structured matrices. Such matrices include Fourier, Walsh Hadamard, Toeplitz, Vandermonde, and many others. Each of which can be applied to vectors in $O(npolylog(n))$. However, general matrix vector multiplication can be shown to require $\Omega(mn)$ operations ,Winograd [8], which matches the running time of the naive algorithm.

In light of this, some authors sought algorithms for other classes of limited matrices. Most of this effort concentrated on binary matrix operations. These results include the historical Four Russians Algorithm [6] and some improvements of it [5] which give a factor log improvement for matrix-matrix multiplication but do not extend naturally to matrix vector operations.

Regarding matrix-vector operations, we cannot avoid accessing each and every entree of our matrix, A. We therefore allow ourselves a preprocessing stage which takes $\Omega(mn)$ operations. After the preprocessing

*Yale University
†Yale University

stage we are given x and produce the product Ax as fast as possible. Under this framework Williams [7] showed that an $n \times n$ binary matrix can be preprocessed in time $O(n^{2+\epsilon})$ and subsequently applied to $b$inary vectors in $O(n^2/\epsilon \log^2 n)$. Williams also extends his result to matrix operations over finite semirings. To the best of the authors understanding his work cannot be extended to real vectors.

In this manuscript we claim that any $m \times n$ matrix over a *finite alphabet*[1] can be preprocessed in time $O(mn)$ such that it can be applied to any *real* vector $x \in \mathbb{R}^n$ in $O(mn/\log(max\{m,n\}))$ operations. Such operations are common, for example, in spectral algorithms for unweighted graphs, nearest neighbor searches, dimensionality reduction, and compressed sensing. Our algorithm also includes all previous results (excluding that of Williams) while using a strikingly simple approach.

The Mailman algorithm

Intuitively, our algorithm multiplies A by x not unlike the local mailman distributes letters. Notice that $Ax = \sum_{i=1}^{n} A^{(i)}x(i)$. Each column $A^{(i)}$ can, metaphorically, indicate one address or house and each $x(i)$ a letter addressed to it. We imagine computing and adding the term $A^{(i)}x(i)$ to the sum as the effort of walking to house $A^{(i)}$ and delivering $x(i)$. The naive algorithm delivers each letter individually to it's address regardless to how far the walk is or if other letters are addressed to the same house. Whereas actual mailmen know much better. First, they arrange their letters according to the shortest route (which includes all houses) while staying put. Then, they walk the route, visiting each house regardless of how many letters should be delivered there (possibly non). Our algorithm decomposes A into two matrices P and U such that $A = UP$. P is the "address-letter" correspondence matrix. Applying P to x is like arranging the letters. U is matrix containing all possible columns in A. Applying U to (Px) is like walking the route. Hance, we name our algorithm after the age old wisdom of the men and women of the postal service.

Our idea can be easily explained for an $m \times n$ matrix A where $m = \log_2(n)$ (w.l.o.g assume $\log_2(n)$ integer) and $A(i,j) \in \{0,1\}$. Later we modify it to include other sizes and possible entrees. Consider the different columns of the matrix A. There are only $2^m = n$ such possible columns since each of the m column entrees can only be either 0 or 1. Let us define the matrix U_n as the matrix containing *all* possible columns over $\{0,1\}$ of length $\log(n)$. By definition U_n contains all the columns that appear in A. More precisely, for any column $A^{(j)}$ there exists an index $1 \leq i \leq n$ such that $A^{(j)} = U_n^{(i)}$. We can therefore define a $n \times n$ matrix, P, such that $A = UP$. In particular the matrix P contains one 1 entree per column such that

[1] $A(i,j) \in \Sigma$, and $|\Sigma|$ is a finite constant.

$P(i,j) = 1$ if $A^{(j)} = U_m^{(i)}$.

We claim that the decomposition of any matrix A into U and P as described above can be achieved in $O(mn)$ steps. Moreover, we show below that applying both P and U requires only $O(n)$ operation. Note that although A is of size $\log(n) \times n$ it can be applied in linear time ($O(n)$), and thus a $\log(n)$ factor is gained. If the number of rows, m, in A is more then $\log(n)$ we can section A into at most $\lceil m/\log(n) \rceil$ matrices each of size at most $\log(n) \times n$. Since each of the sections can be applied in $O(n)$ operations, the entire matrix can be applied in $O(mn/\log(n))$ operations.

Applying U requires linear time

Let us denote by U_n the $\log_2(n) \times n$ matrix containing all possible strings over $\{0,1\}$ of length $\log(n)$. Notice the simple fact that U_n can be constructed using $U_{n/2}$ as such

$$U_1 = (\,0 \quad 1\,), \quad U_n = \begin{pmatrix} 0,\ldots,0 & 1,\ldots,1 \\ U_{/2} & U_{n/2} \end{pmatrix}$$

Applying U_n to any vector z requires less then $2n$ operations. This can be seen by dividing z into its first and second halves z_1 and z_2 and computing the product $U_n z$ recursively.

$$U_n z = \begin{pmatrix} 0,\ldots,0 & 1,\ldots,1 \\ U_{n/2} & U_{n/2} \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} \sum z_2 \\ U_{n/2}(z_1 + z_2) \end{pmatrix}$$

Since computing $z_1 + z_2$ and $\sum z_2$ requires n operations (the length of z is n) we get the recurrence relation

$$T(2) = 2, \;\; T(n) = T(n/2) + n \Rightarrow T(n) \leq 2n$$

Constructing P efficiently

Since U is applied implicitly (not stored) we are only concerned with constructing P. We set an entree $P(i,j)$ to 1 if $A^{(j)} = U_n^{(i)}$. Notice that by our construction of U_n its i'th column encodes the binary value of the index i. Thus, if a column of A contains the binary representation of the value i it is equal to $U^{(i)}$. We therefore construct P by reading A and setting $P(i,j)$ to 1 if $A^{(j)}$ represents the value i. This, clearly can be done in time $O(mn)$, the size of A. Since P contains only n nonzeros (one for each column of A) it can trivially be applied in n steps.

Generalizations

"Tall and skinny" matrices

Notice that if $n \leq m$ we can section A horizontally to sections of size $m \times \lceil \log(m) \rceil$ and argue very similarly that $P^T U^T$ can be applied in linear time. Thus a $\log(m)$ saving factor in running time is achieved.

Constant sized alphabet

The definition of U_m can be changed such that it encodes all possible strings over a larger alphabet $\Sigma = \{\sigma_1, \ldots, \sigma_{|\Sigma|}\}$. Under the mapping $\sigma_\ell \to \ell - 1$ there is a one to one mapping between strings and the integers they represent in base $|\Sigma|$.

$$U_m = \begin{pmatrix} \sigma_1, \ldots, \sigma_1 & \ldots & \sigma_\ell, \ldots, \sigma_\ell \\ U_{m-1} & \ldots & U_{m-1} \end{pmatrix}$$

The matrix U_m of size $m \times |\Sigma|^m$ encodes all possible strings of length m over Σ and can be applied in linear time if $|\Sigma|^m = O(n)$. In this case we must divide our matrix into sections of size $\log_{|\Sigma|}(n) \times n$. The total running time therefore becomes $O(mn \log(|\Sigma|)/\log(\max\{m, n\})$.

Matrix operations over semirings

Suppose we supply Σ with and associative and commutative addition operation $(+)$, and a multiplication operation $(\cdot)$ that distributes over $(+)$. If both A and x are chosen from Σ the matrix vector multiplication over the semiring $\{\Sigma, +, \cdot\}$ can be performed using the exact same algorithm. This is, of course, not a new result. However, it includes all other log-factor-saving results.

Applications

We claim that multiplying constant sized alphabet matrices to real vectors is not uncommon. We do not claim to cover even a small fraction of possible applications but rather give a few examples.

Graph Laplacian Eigensystems

The graph Laplacian L of an unweighted graph $G(V, E)$ is an $n \times n$ matrix ($|V| = n$) such that $L(i, j) = -1$ if $\{i, j\} \in E$ and zero else. Also, $L(i, i) = -\sum_{j \neq i} L(i, j)$. The simplest method for finding the largest

eigenvector of the Graph Laplacian matrix is the *power method*. A more complicated but extremely common method is the *Lanczos method*. Both methods require multiplying L by a (possibly large) number of real vectors.

Assuming the method of choice requires s steps to converge[2] the trivial algorithm requires $O(sn^2)$ operations. Clearly, the matrix L can be decomposed into $L = D - A$ where D is diagonal (and therefore can be applied in $O(n)$ operations) and A is a $\{0, 1\}$ matrix. Applying the mailman algorithm for applying A reduces the running time to $O(n^2 + sn^2/log(n))$ (The n^2 term is requires for preprocessing A). This beats the naive method for any $s > 1$.

Dimensionality reduction

Assume we are given p points $\{x_1, \ldots, x_p\}$ in $\mathbb{R}^n$ and are asked to embed them into $\mathbb{R}^m$ such all distances between points are preserved up to distortion ε. A classic result by Johnson and Lindenstrauss [4] shows that this is possible for $m = \Theta(\log(p)/\varepsilon^2)$. The algorithm first chooses a random $m \times n$ matrix, A, from a special distribution. Then, each of the vectors (points) $\{x_1, \ldots, x_p\} \in \mathbb{R}^n$ are multiplied by A. The embedding $x_i \to Ax_i$ can be shown to have the desired property with some constant probability.

Clearly a naive application of A to each vector requires $O(mn)$ operations. Recently, by Ailon and Liberty [3] showed that a simple modification to the work of Ailon and Chazelle [2] allows A to be applied in $O(n \log(m))$ operations. We claim however that in some situations an older result by Achlioptas can by trivially modified to out preform this result. Achlioptas [1] showed that the entries of A can be chosen *i.i.d* from $\{-1, 0, 1\}$ with some constant probabilities. Using our method and Achlioptas's matrix, one can apply A to each vector in $O(n \log(p)/\log(n)\varepsilon^2)$ operations (remainder $m = O(\log(p)/\varepsilon^2)$). Therefore, for a constant ε, if p is polynomial in n, applying A to x_i requires only $O(n)$ operation which outperforms the best known $O(n \log(m))$ and matches the lower bound.

References

[1] Dimitris Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *J. Comput. Syst. Sci.*, 66(4):671–687, 2003.

[2] Convergence of iterative methods is a well studied topic which we conveniently sidestep at the moment.

[2] Nir Ailon and Bernard Chazelle. Approximate nearest neighbors and the fast Johnson-Lindenstrauss transform. In *Proceedings of the 38st Annual Symposium on the Theory of Compututing (STOC)*, pages 557–563, Seattle, WA, 2006.

[3] Nir Ailon and Edo Liberty. Fast dimension reduction using rademacher series on dual bch codes. In *Symposium on Discrete Algorithms (SODA), accepted*, 2008.

[4] W.B. Johnson and J. Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. *Contemp. Math.*, 26:189–206, 1984.

[5] N Santoro and J Urrutia. An improved algorithm for boolean matrix multiplication. *Computing*, 36(4):375–382, 1986.

[6] M.A. Kronrod V.L. Arlazarov, E.A. Dinic and I.A. Faradzev. On economic construction of the transitive closure of a direct graph. *Soviet Mathematics, Doklady*, (11):1209–1210, 1970.

[7] Ryan Williams. Matrix-vector multiplication in sub-quadratic time: (some preprocessing required). In *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 995–1001, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[8] Shmuel Winograd. On the number of multiplications necessary to compute certain functions. *Communications on Pure and Applied Mathematics*, (2):165–179, 1970.