Recitation 4 (9/26 - 9/30)

Overview

This week we will be reviewing the support page on matrix - vector multiplication
http://www-student.cse.buffalo.edu/~atri/cse331/support/matrix-vect/index.html

- Definition of matrices & vectors
- Example of a matrix - vector multiplication (time O(n^2))
- Inner product
- Structured matrices
- Distributive law

What is a Matrix?

In mathematics, a vector is simply an ordered list of values. A matrix represents a vector of vectors, or a 2-d vector. From the support pages:

A matrix A with m rows and n columns (also denoted as an $m \times n$ matrix) will in code be defined as `int [][] A = new int[m][n]` (assuming the matrix stores integers).

Note

For the programming question in hw3, you are dealing with square matrices. n=m

Matrix Multiplication

A matrix that has n **columns** can be multiplied by any matrix or vector that has n **rows**.

Inner Product

The inner product is a multiplication of two vectors. Each element is multiplied toget
(At this point it is easier to refer to the support page)

http://www-student.cse.buffalo.edu/~atri/cse331/support/matrix-vect/index.html
Review the “Inner Product” and “Matrix multiplication” notes

Time Complexity

The time complexity of a regular multiplication is O(n*m), or O(n^2) for square matrices. You can think of it as being two nested for loops. One for each row, and the inner one for each column. Here is a pseudocode example of multiplying matrix A * vector x:

```
Allocate an array of length n and call it y

for(i=0;i<n;i++)
    y[i]=0


for{(i=0;i<n;i++)
    for(j=0;j<n;j++)
        y[i]+= A[i][j]* x[j]

return y
```

Example

$$\begin{bmatrix} 3 & -4 & 8 \\ 0 & 7 & -9 \\ 2 & 2 & 4 \end{bmatrix} * \begin{bmatrix} -7 \\ 1 \\ 3 \end{bmatrix}$$

Answer:

$$\begin{bmatrix} (3)(-7)+(-4)(1)+(8)(3) \\ (0)(-7)+(7)(1)+(-9)(3) \\ (2)(-7)+(2)(1)+(4)(3) \end{bmatrix}$$

$$\begin{bmatrix} -1 \\ -20 \\ 0 \end{bmatrix}$$

Structured Matrices

If we know what the structure of a matrix is, we can often cut down on our algorithm time.

Outer Product

Outer Product

Given two vectors $\mathbf{s}$ and $\mathbf{t}$, their *outer product* is defined as the $n \times n$ matrix $\mathbf{A}$ such that for any entry (i, j), we have

$$A[i][j] = s[i] \cdot t[j].$$

Example

s = [4, 3, 2]
t = [-3, 2, -1]

$$\begin{bmatrix} 4 \\ 3 \\ 2 \end{bmatrix} \begin{bmatrix} -3, 2, -1 \end{bmatrix} \begin{bmatrix} & & \\ & & \\ & & \end{bmatrix}$$

Answer:

$$\begin{bmatrix} -12 & 8 & -4 \\ -9 & 6 & -3 \\ -6 & 4 & -2 \end{bmatrix}$$

Outer Product in multiplication

Multiplying outer product with a vector

Given three vectors $\mathbf{s}, \mathbf{t}$ and $\mathbf{x}$, compute the product of the outer product of $\mathbf{s}$ and $\mathbf{t}$ with the vector $\mathbf{x}$. If $\mathbf{y}$ is the product, then it is defined as (for every $0 \leq i < n$):

$$y_i = \sum_{j=0}^{n-1} s[i] \cdot t[j] \cdot x[j].$$

Algorithm Idea

Observe that

$$y_i = \sum_{j=0}^{n-1} s[i] \cdot t[j] \cdot x[j] = s[i] \cdot \sum_{j=0}^{n-1} t[j] \cdot x[j] = s[i] \left\langle \mathbf{t}, \mathbf{x} \right\rangle .$$

In other words, once we have computed $\langle \mathbf{t}, \mathbf{x} \rangle$, one can compute y_i by just multiplying the inner product with $s[i]$, which takes $O(1)$ time for each $0 \leq i < n$.

We can use a simple vector **x**, say [3, 2, -2], as an example, so our multiplication would be:

$$\begin{bmatrix} -12 & 8 & -4 \\ -9 & 6 & -3 \\ -6 & 4 & -2 \end{bmatrix} * \begin{bmatrix} 3 \\ 2 \\ -2 \end{bmatrix}$$

If we followed the algorithm idea, we have:

$\langle \mathbf{t}, \mathbf{x} \rangle = -3$

We then multiply this down the line for vector **s**:

$y = [-12, -9, -6]$

Compare this to the answer you get when using our original algorithm with the outer product matrix. The runtime for this algorithm is Θ(n)

WHY?
Calculating the inner product is Θ(n). Then we perform Θ(n) operations in total for the **s**[i]'s.

Distributive Law

Given three values *a, b,* and *c*:

$$a * b + a * c = a * (b + c)$$

Simple but useful, as you can see above with our outer product example. This implies that:

$$\sum_{i=0}^{n-1} a \cdot b_i = a \cdot \left(\sum_{i=0}^{n-1} b_i \right)$$

(From the support page)

With our above outer product example, this allowed us to store values and not have to recalculate them each time.

This will be very useful for problem 3 on hw3.

Computing Sums of Products

At this point, it is good again to simply follow the support pages on this. A good way would be to write the example sum of product full bit on the board, then simply show how to cut out the bits of each sums (on the board).