Matrix Math for Data Science

Part 1 - Addition and Subtraction

(GregSimpson 2018, December)

1 Introduction and Rationale

I recently returned to graduate school as a "seasoned computer professional". I was looking to update my skills and fill in the gaps in my data science knowledge. I applied and was accepted into the University of Denver's MS in Data Science program.

One of the fundamental topics in data science is math. So the university has a few "bridge" classes to refresh and sharpen your math skills. My first class this past semester was Discrete Math and Linear Algebra. I will save the discrete math topic for a different series on another day. For this series, I will discuss matrix math with a little linear algebra mixed in.

There are lots of articles available that discuss the reasons for using various mathematical algorithms. In an effort to not add to that pile of material, I will focus on the mechanics of how you go about applying some techniques. My examples will start out pretty simple and try not to get too complex. I intend this to be digestible by beginners in the field. We'll see how it goes.

Please feel free to leave feedback.

2 Definitions

We will start with a few definitions related to matrices. It helps if you think of them like database tables, so I will use database table imagry in my explanations.

columns the fields that define a table; they go left to right, horizontally

`example of columns`

$$\begin{array}{c} \text{columns} \\ \\ rows \end{array} \begin{array}{c} \begin{array}{cccccc} col_1 & col_2 & ... & col_n \end{array} \\ \begin{bmatrix} \vdots & \vdots & \ddots & \vdots \end{bmatrix} \end{array}$$

dimension numeric description of the number of rows and columns; in math problems, they usually say an "m by n" matrix or "m x n"

where m = rows and n = columns

`3x3 : m=3, n=3` $\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$

`2x3 : rows=2, cols=3` $\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}$

`2x4 : m=2, n=4` $\begin{bmatrix} a & b & c & d \\ e & f & g & h \end{bmatrix}$

matrix an array that has entries in two directions (or more): an "m by n" or "m x n" array

(see the examples for 'dimension')

multiplication multiplier x multiplicand = product

reduced row eschelon form (RREF) see reference link below

rows the data that populates the table; each row usually has an entry for each column

`example of rows`

$$\begin{array}{c} \\ row_1 \\ row_2 \\ row_m \end{array} \begin{array}{c} \begin{array}{cccc} col_1 & col_2 & ... & col_n \end{array} \\ \begin{bmatrix} \vdots & \vdots & \dots & \vdots \\ \vdots & \vdots & \dots & \vdots \\ \vdots & \vdots & \dots & \vdots \end{bmatrix} \end{array}$$

shape another term for dimension

vector an array that has entries in one direction:
1 row by n columns
or m rows by 1 column

`row vector`

$$\begin{array}{c} \text{columns} \\ 1row \end{array} \begin{array}{c} \begin{array}{cccc} col_1 & col_2 & ... & col_n \end{array} \\ \begin{bmatrix} a & b & \dots & d \end{bmatrix} \end{array}$$

`column vector`

$$\begin{array}{c} \text{1 column} \\ row_1 \\ row_2 \\ row_m \end{array} \begin{array}{c} \begin{array}{c} col_1 \end{array} \\ \begin{bmatrix} a \\ b \\ \vdots \end{bmatrix} \end{array}$$

3 Matrix Addition

In order to add any 2 matrices, they must be of the same shape. That means that you can add a 3x3 matrix to a 3x3 matrix, or a 2x2 to a 2x2. But you cannot add a 3x3 to a 2x3. I will explain the reason for that in a minute. When you add 2 matrices together, it is a cell by cell operation.
That means if are trying to add these 2 matrices to produce a new matrix C,

$$A + B = C$$

where:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} B = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

the resulting matrix , C =

$$\begin{bmatrix} a+e & b+f \\ c+g & d+h \end{bmatrix}$$

In general, when you write out a program to navigate an array, the common practice (or convention) is to use the subscript 'i' to iterate over the rows and 'j' to iterate over the columns. Math texts use the same conventions in their examples. So, the conventional way to describe what the example above shows is :

$$A + B = [a_{ij} + b_{ij}]$$

where i and j iterate over all of the rows and columns in A and B
and add the values in the cells of A and B and record each sum in matrix C.

So, to be clear, all of the cells in all 3 of the matrices involved need to be able to line up. If the matrix shapes were different, say you tried to add a (2x2) matrix to a (2x3) matrix, we have no way to process any of the cells that do not line up with the other matrix. That is why they need to be the same shape.

3.1 Practical Examples

We will do a few practical examples to make it a lot more clear.

3.1.1 2x2 matrix addition example

$$A + B = C$$

where:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

the resulting matrix

$$C = \begin{bmatrix} 1+5 & 2+6 \\ 3+7 & 4+8 \end{bmatrix} = \begin{bmatrix} 6 & 8 \\ 10 & 12 \end{bmatrix}$$

3.1.2 3x3 matrix addition example

$$A + B = C$$

where:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

the resulting matrix

$$C = \begin{bmatrix} 1+1 & 2+2 & 3+3 \\ 4+4 & 5+5 & 6+6 \\ 7+7 & 8+8 & 9+9 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 6 \\ 8 & 10 & 12 \\ 14 & 16 & 18 \end{bmatrix}$$

This really is not as hard as it sounds, at least for small matrices. But it can get pretty messy if you are doing this by hand. Here is a short and simple python program that implements our 3x3 matrix example.
If you are familiar with python and Pandas dataframes, you might notice that an m x n matrix looks a lot like a Pandas dataframe. This implementation is only done using simple python native elements.
It is fine this makes no sense to you at the moment. You do not need python to understand the math here. Explaining that in much more detail is beyond the scope of this paper, and could get confusing. So that will also be another topic for a different paper.

3.1.3 3x3 matrix addition example in python code

```
# Add 2, 3x3 matrices to produce a 3rd 3x3 matrix
#  A + B = C

# Define matrix A
A = [[1,2,3],
     [4,5,6],
     [7,8,9]]

# Define matrix B
B = [[1,2,3],
     [4,5,6],
     [7,8,9]]

# Define matrix C
C = [[0,0,0],
     [0,0,0],
     [0,0,0]]

# i is the row index
for i in range(len(A)):
    # j is the column index
    for j in range(len(A[0])):
        C[i][j] = A[i][j] + B[i][j]

for sum in C:
    print(sum)

# result:
# [ 2,  4,  6]
# [ 8, 10, 12]
# [14, 16, 18]
```

4 Matrix Subtraction

Matrix subtraction works in much the same way as addition, except that you subtract the values instead of adding them as you parse each cell in the matrices. The rules about only working for matrices of the same shape apply to subtraction just as they did to addition.
So, where this is the mathematical definition of matrix addition :

$$A + B = [a_{ij} + b_{ij}]$$

where i and j iterate over all of the rows and columns in A and B, add the values in the cells and record each sum in matrix C.

This is the mathematical definition of matrix subtraction:

$$A - B = [a_{ij} - b_{ij}]$$

where i and j iterate over all of the rows and columns in A and B, subtract the values in the cells and record each sum in matrix C.

4.1 Practical Example

4.1.1 3x3 matrix subtraction example

$$A - B = C$$

where:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

the resulting matrix

$$C = \begin{bmatrix} 1-1 & 2-2 & 3-3 \\ 4-4 & 5-5 & 6-6 \\ 7-7 & 8-8 & 9-9 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Here is a short and simple python program that implements our 3x3 matrix subtraction example.

4.1.2 3x3 matrix subtraction example in python code

```
# Subtract 2, 3x3 matrices to produce a 3rd 3x3 matrix
#  A - B = C

# Define matrix A
A = [[1,2,3],
     [4,5,6],
     [7,8,9]]

# Define matrix B
B = [[1,2,3],
     [4,5,6],
     [7,8,9]]

# Define matrix C
C = [[0,0,0],
     [0,0,0],
     [0,0,0]]

# i is the row index
for i in range(len(A)):
    # j is the column index
    for j in range(len(A[0])):
        C[i][j] = A[i][j] - B[i][j]

for sum in C:
    print(sum)

# result:
# [ 0,  0,  0]
# [ 0,  0,  0]
# [ 0,  0,  0]
```

5 reference links

https://du.edu

https://en.wikipedia.org/wiki/Row_echelon_form

https://ritchieschool.du.edu/datascience/

https://www.geeksforgeeks.org/python-program-multiply-two-matrices/

https://www.khanacademy.org/math/linear-algebra/alternate-bases/eigen-everything/v/linear-algebra-introduction-to-eigenvalues-and-eigenvectors

https://www.kdnuggets.com/2018/09/essential-math-data-science.html

http://stackoverflow.com/

https://stackoverflow.com/questions/31487264/numpy-eigenvalues-correct-but-eigenvectors-wro

Here is the book that we used for the class
https://www.springer.com/us/book/9783319747477