CSC 211: Object Oriented Programming
Multidimensional Arrays

Michael Conti

Department of Computer Science and Statistics
University of Rhode Island

Spring 2020

THINK BIG 🌎 WE DO℠

Original design and development by Dr. Marco Alvarez

Arrays, of any dimension, are **statically allocated** in memory with a size calculated at compile time. That is, their size is **fixed** and **cannot** be changed later.
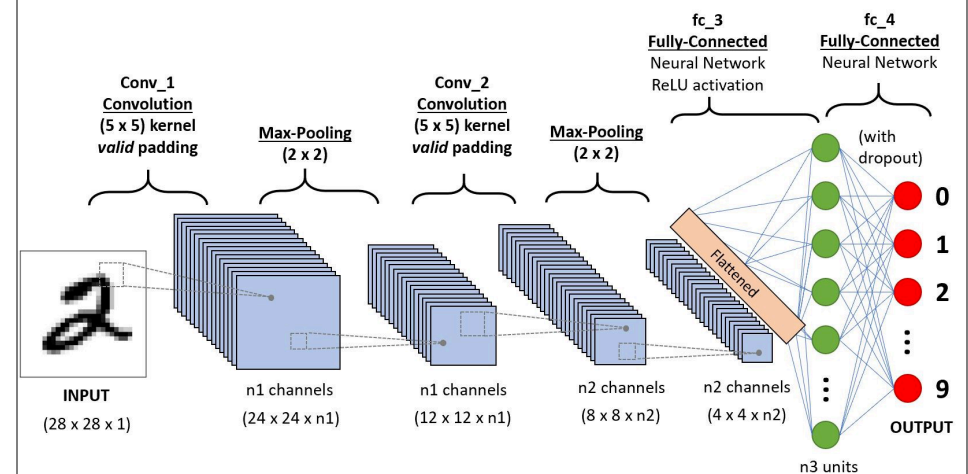

Multidimensional Arrays

‣ Generalization of **arrays** to multiple dimensions
  ✓ e.g. matrices, tensors

‣ Each element can be accessed using its corresponding **indices**

A[i]

A[i][j]   A[i][j][k]
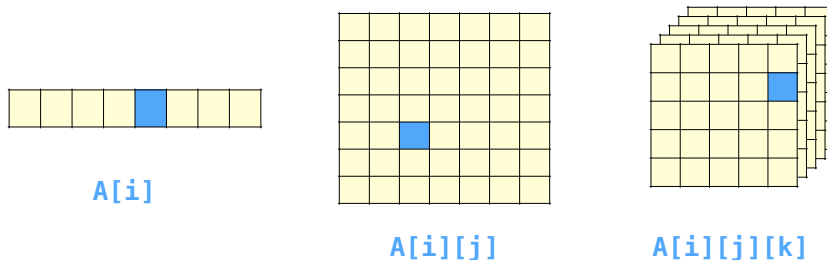
Modern machine learning

Declaration of 2D arrays

```
// array declaration by specifying size
int matrix1[10][10];

// can also declare an array of
// user specified size
int n = 8;
int matrix2[n][n];

// can declare and initialize elements
double matrix3[2][2];
matrix3 = { {10.0, 20.0}, {30.0, 40.0} };
```

Indexing 2D arrays

	0	1	2	3
0	A[0][0]			
1			A[1][2]	
2		A[2][1]		
3				

Indexing 2D arrays

· Individual elements can be accessed by using the **subscription operator [ ]**

1	2	3
4	5	6
7	8	9

```
int matrix2[3][3];

for (int i = 0 ; i < 3 ; i ++) {
    for (int j = 0 ; j < 3 ; j ++) {
        matrix[i][j] = (j + 1) + i * 3;
    }
}
```
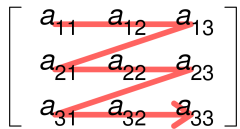
How are these arrays stored in memory?

· In computing, **row-major** order and **column-major** order are two methods for storing multidimensional arrays as contiguous blocks of memory

  ✓ row-major order is used in C, C++, Objective-C (for C-style arrays), PL/I, Pascal, Speakeasy, SAS, …

  ✓ column-major order is used in Fortran, MATLAB, GNU Octave, S-Plus, R, Julia, …

· Alternatively, neither row-major or column-major approaches are also used (non-contiguous blocks)
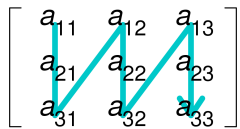
  ✓ Java, C#, CLI, .Net, Scala, Swift, Python, Lua, …

Row-major and column-major order

Row-major order

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Column-major order

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

	0	1	2	3
0	1	2	3	4
1	8	6	4	2
2	10	20	30	40
3	5	7	9	11

...

Question

· How many bytes are these arrays using in memory?

```
int array[100000];



int matrix[1000][1000];



double tensor[1000][1000][1000];
```

Question

· Write a program that reads in the value of n, and prints the identity matrix of size n x n?

Multidimensional arrays and functions

· The first array size need not be specified

· The second (and any subsequent) must be given

· Example:

```
int foo(int list[][100], int rows, int cols);
```

size is required so the compiler can calculate the memory addresses of individual elements

https://stackoverflow.com/questions/12813494/why-do-we-need-to-specify-the-column-size-when-passing-a-2d-array-as-a-parameter

Question

- Write a function that adds two 2D matrices?

Question

- Write a function that multiplies two 2D matrices?