

# CSC 561: Neural Networks and Deep Learning

## Preliminaries

Marco Alvarez

Department of Computer Science and Statistics  
University of Rhode Island

Spring 2024



# Linear Algebra

Subset of most relevant concepts to Deep Learning

Credit: Deep Learning by Ian Goodfellow, Yoshua Bengio, Aaron Courville

## Scalars and vectors

- Scalars
  - ✓ integers, real numbers, rational numbers, etc.
  - ✓ usually denoted by lowercase letters
- Vectors
  - ✓ 1-D array of elements (scalars)

$$\mathbf{x} \in \mathbb{R}^n$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

3

## Matrices and tensors

- Matrices
  - ✓ 2-D array of elements
- Tensors
  - ✓ homogeneous arrays that may have **zero (scalar) or more dimensions**

$$\begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \quad A \in \mathbb{R}^{m \times n}$$

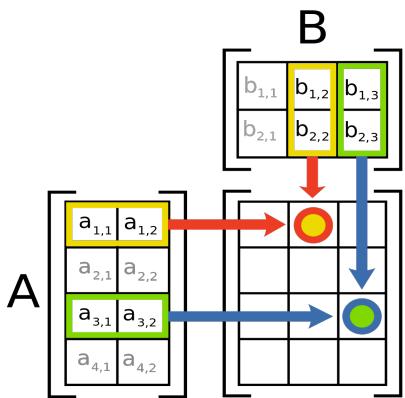
4

## Basics

### Matrix Multiplication

$$C = AB$$

$$C_{i,j} = \sum_k A_{i,k} B_{k,j}$$



number of columns in A must be equal to the number of rows in B

[https://en.wikipedia.org/wiki/Matrix\\_multiplication](https://en.wikipedia.org/wiki/Matrix_multiplication)

5

## Dot product

$$\begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

$$\mathbf{x}^T \mathbf{y} = \sum_{i=1}^n x_i y_i$$

$$\mathbf{x}^T \mathbf{y} = \|\mathbf{x}\| \|\mathbf{y}\| \cos \theta$$

$$\mathbf{x}^T \mathbf{y} \in \mathbb{R}$$

6

## Matrix transpose

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \\ A_{3,1} & A_{3,2} \end{bmatrix} \Rightarrow A^T = \begin{bmatrix} A_{1,1} & A_{2,1} & A_{3,1} \\ A_{1,2} & A_{2,2} & A_{3,2} \end{bmatrix}$$

$$(A^T)_{i,j} = A_{j,i}$$

$$(AB)^T = B^T A^T$$

7

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\forall x \in \mathbb{R}^n, I_n x = x$$

$$A^{-1} A = I_n$$

8

## Norms

- Functions that measure the **magnitude** (“length”) of a vector
  - ✓ **strictly positive**, except for the zero vector
  - ✓ think about the distance between zero and the point represented by the vector

$$f(\mathbf{x}) = 0 \Rightarrow \mathbf{x} = \mathbf{0}$$

$$f(\mathbf{x} + \mathbf{y}) \leq f(\mathbf{x}) + f(\mathbf{y}) \text{ (triangle inequality)}$$

$$\forall \alpha \in \mathbb{R}, f(\alpha \mathbf{x}) = |\alpha| f(\mathbf{x})$$

## Norms

$$\ell_1\text{-norm: } \|\mathbf{x}\|_1 = \sum_{i=1}^n |x_i|$$

$$\ell_2\text{-norm: } \|\mathbf{x}\|_2 = \left( \sum_{i=1}^n x_i^2 \right)^{\frac{1}{2}}$$

$$\text{max norm: } \|\mathbf{x}\|_\infty = \max_i |\mathbf{x}_i|$$

## Special matrices and vectors

- Unit vector

$$\|\mathbf{x}\|_2 = 1$$

- Symmetric matrix

$$A = A^T$$

- Orthogonal matrix

$$A^T A = A A^T = I$$

$$A^{-1} = A^T$$

11

## Programming with Tensors

10

# Non-negotiable (Python)

- Basic data types
  - ✓ booleans, integers, floating point values, strings
- Control flow
- Built-in data structures
  - ✓ lists, tuples, dictionaries, sets
  - ✓ iterators
- Functions
  - ✓ functions can be assigned, passed, returned, and stored
  - ✓ lambda functions (inline and anonymous)
- Classes

13

```
n = 100000000
array1 = np.random.rand(n)
array2 = np.random.rand(n)

def dot_python(x, y):
    sum = 0
    for i in range(len(x)):
        sum += x[i] * y[i]
    return sum

def dot_numpy(x, y):
    return np.dot(x, y)

time_taken = timeit.timeit(lambda: dot_numpy(array1, array2), number=1)
print(f'Numpy Time: {time_taken} seconds')
Numpy Time: 0.05994947799996453 seconds

array1 = array1.tolist()
array2 = array2.tolist()
time_taken = timeit.timeit(lambda: dot_python(array1, array2), number=1)
print(f'Python Time: {time_taken} seconds')
Python Time: 8.325287125999978 seconds
```

15

# Numpy

- Library for scientific computing
  - ✓ provides a **high-performance** multidimensional array object and routines for fast operations on these arrays
- The **ndarray** object encapsulates n-dimensional arrays of homogeneous data types
  - ✓ many operations performed in **compiled code** for higher performance
  - ✓ have a fixed size at creation
    - changing the size of an ndarray will create a new array and delete the original

14

The following figures are from:

“*NumPy Illustrated: The Visual Guide to NumPy*” by Lev Maximov

[https://betterprogramming.pub\(numpy-illustrated-the-visual-guide-to-numpy-3b1d4976de1d](https://betterprogramming.pub(numpy-illustrated-the-visual-guide-to-numpy-3b1d4976de1d)

## Elegant code

```
In [3]: a = [1, 2, 3]
[q*2 for q in a]

Out[3]: [2, 4, 6]
```

```
In [4]: a = np.array([1, 2, 3])
a * 2

Out[4]: array([2, 4, 6])
```

```
In [1]: a = [1, 2, 3]
b = [4, 5, 6]
[q+r for q, r in zip(a, b)]

Out[1]: [5, 7, 9]
```

```
In [2]: a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
a + b

Out[2]: array([5, 7, 9])
```

17

## Homogeneous and fixed-length

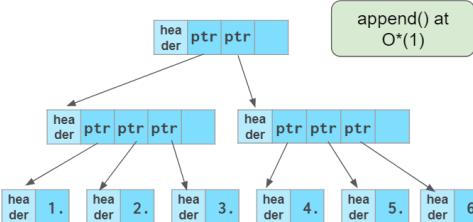
python list

1.	2.	3.
4.	5.	6.

vs

numpy array

1.	2.	3.
4.	5.	6.



18

## Creating numpy arrays

```
a = np.array([1., 2., 3.]) → a
1. 2. 3. .dtype == np.float64
.shape == (3,)
```

```
np.zeros(3) → 0. 0. 0.
np.array([1, 2, 3]) → 1 2 3
np.zeros_like(a) → 0 0 0
```

```
np.ones(3) → 1. 1. 1.
np.ones_like(a) → 1 1 1
```

```
np.empty(3) → 5e-296 7e-297 1e-296
np.empty_like(a) → 54087 1630433 2036429
6897 390 426
```

```
np.full(3, 7.) → 7. 7. 7.
np.full_like(a, 7) → 7 7 7
```

## Creating numpy arrays

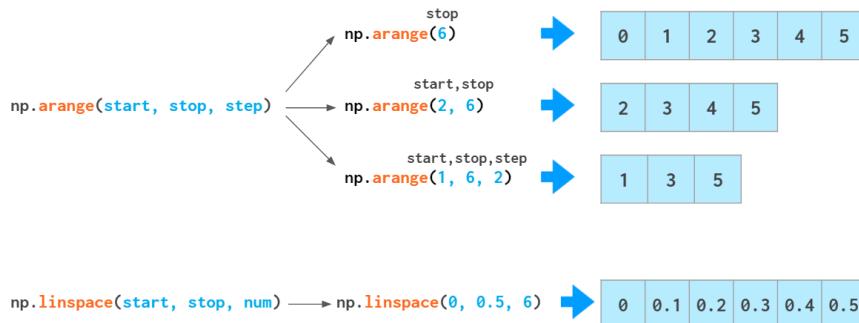
```
a = np.array([[1, 2, 3], [4, 5, 6]]) → a
1 2 3 .dtype == np.int32
4 5 6 .shape == (2, 3)
len(a) == a.shape[0]
```

```
np.zeros((3, 2)) → 3 2
0. 0. 0. 0. 0.
np.full((3, 2), 7) → 7 7
7 7 7
np.eye(3, 3) → 1. 0. 0.
0. 1. 0.
0. 0. 1.
= np.eye(3)
```

19

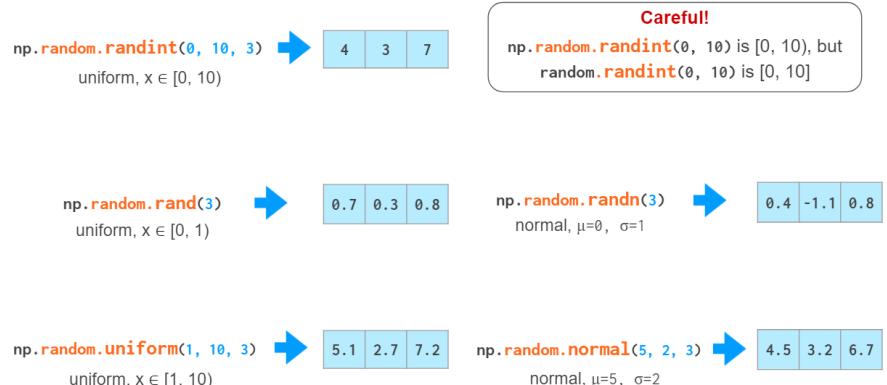
20

# Creating numpy arrays



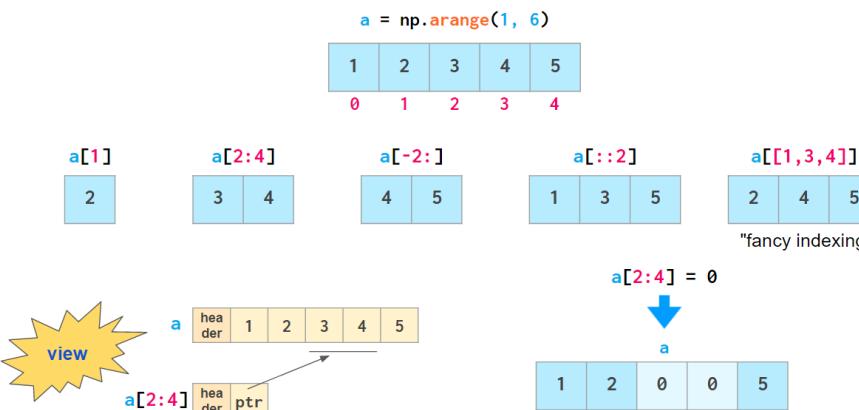
21

# Creating numpy arrays



22

# Indexing



23

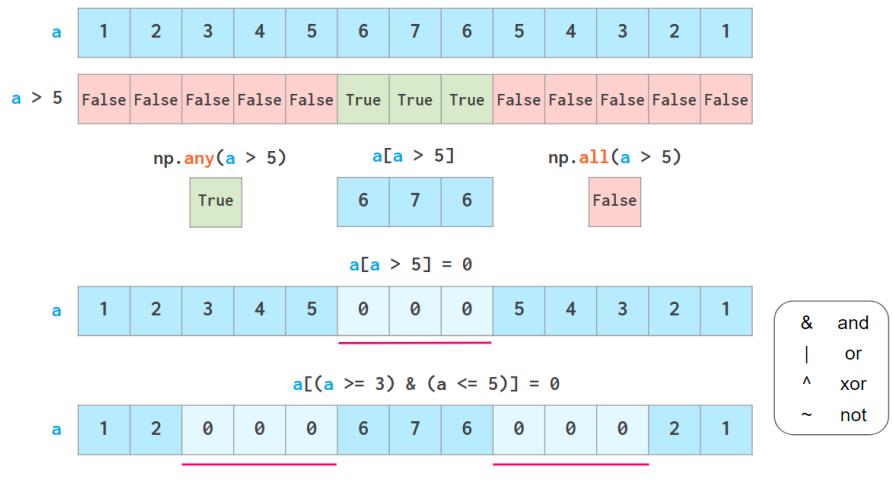
# Careful when making copies

python list vs numpy array

<code>a = [1, 2, 3]</code>	vs	<code>a = np.array([1, 2, 3])</code>
<code>b = a</code> # no copy		<code>b = a</code> # no copy
<code>c = a[:]</code> # copy		<code>c = a[:]</code> # no copy!!!
<code>d = a.copy()</code> # copy		<code>d = a.copy()</code> # copy

24

## Boolean indexing



## Vector operations

$1 \begin{smallmatrix} 2 \\ 1 \end{smallmatrix} + 4 \begin{smallmatrix} 8 \\ 4 \end{smallmatrix} = \begin{smallmatrix} 5 \\ -3 \end{smallmatrix}$	$8 \begin{smallmatrix} 40 \\ 1.6 \end{smallmatrix} = \begin{smallmatrix} 8 \\ 40 \end{smallmatrix}$
$4 \begin{smallmatrix} 8 \\ 4 \end{smallmatrix} / 2 \begin{smallmatrix} 5 \\ 1 \end{smallmatrix} = \begin{smallmatrix} 2.0 \\ 1 \end{smallmatrix}$	$\text{np.float64}$
$4 \begin{smallmatrix} 8 \\ 4 \end{smallmatrix} // 2 \begin{smallmatrix} 5 \\ 1 \end{smallmatrix} = \begin{smallmatrix} 2 \\ 1 \end{smallmatrix}$	$\text{np.int32}$
$3 \begin{smallmatrix} 4 \\ 1 \end{smallmatrix} ** 2 \begin{smallmatrix} 3 \\ 2 \end{smallmatrix} = \begin{smallmatrix} 9 \\ 64 \end{smallmatrix}$	
$1 \begin{smallmatrix} 2 \\ 1 \end{smallmatrix} * 3 \begin{smallmatrix} 3 \\ 1 \end{smallmatrix} = \begin{smallmatrix} 3 \\ 6 \end{smallmatrix}$	
$1 \begin{smallmatrix} 2 \\ 1 \end{smallmatrix} / 3 \begin{smallmatrix} 3 \\ 1 \end{smallmatrix} = \begin{smallmatrix} 0.33 \\ 0.67 \end{smallmatrix}$	$\text{np.float64}$
$1 \begin{smallmatrix} 2 \\ 1 \end{smallmatrix} // 2 \begin{smallmatrix} 3 \\ 1 \end{smallmatrix} = \begin{smallmatrix} 0 \\ 1 \end{smallmatrix}$	$\text{np.int32}$
$3 \begin{smallmatrix} 4 \\ 1 \end{smallmatrix} ** 2 \begin{smallmatrix} 3 \\ 2 \end{smallmatrix} = \begin{smallmatrix} 9 \\ 16 \end{smallmatrix}$	

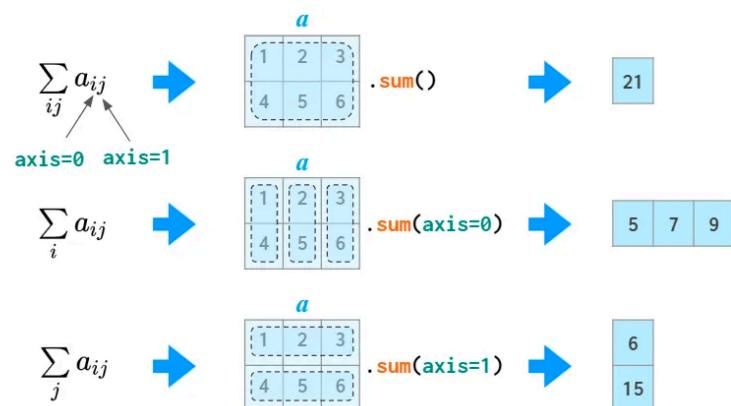
26

## Math functions

$a^2$	$= \begin{smallmatrix} 2 & 3 \\ 4 & 9 \end{smallmatrix} ** 2 = \begin{smallmatrix} 4 & 9 \end{smallmatrix}$	$\vec{a} \cdot \vec{b} = \text{np.dot}(\begin{smallmatrix} 1 & 2 \\ 3 & 4 \end{smallmatrix}, \begin{smallmatrix} 3 & 4 \\ 1 & 2 \end{smallmatrix}) = 11$
$\sqrt{a}$	$= \text{np.sqrt}(\begin{smallmatrix} 4 & 9 \end{smallmatrix}) = \begin{smallmatrix} 2. \\ 3. \end{smallmatrix}$	
$e^a$	$= \text{np.exp}(\begin{smallmatrix} 1 & 2 \end{smallmatrix}) = \begin{smallmatrix} 2.72 \\ 7.39 \end{smallmatrix}$	$\vec{a} \times \vec{b} = \text{np.cross}(\begin{smallmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \end{smallmatrix}, \begin{smallmatrix} 0 & 3 & 0 \end{smallmatrix}) = \begin{smallmatrix} 0 & 0 & 6 \end{smallmatrix}$
$\ln a$	$= \text{np.log}(\text{np.e} \cdot \text{np.e}+2) = \begin{smallmatrix} 1. \\ 2. \end{smallmatrix}$	
$\text{np.floor}(\begin{smallmatrix} 1.1 & 1.5 & 1.9 & 2.5 \end{smallmatrix})$	$= \begin{smallmatrix} 1. \\ 1. \\ 1. \\ 2. \end{smallmatrix}$	$\text{np.max}(\begin{smallmatrix} 1 & 2 & 3 \end{smallmatrix}) = 3$
$\text{np.ceil}(\begin{smallmatrix} 1.1 & 1.5 & 1.9 & 2.5 \end{smallmatrix})$	$= \begin{smallmatrix} 2. \\ 2. \\ 2. \\ 3. \end{smallmatrix}$	$\begin{smallmatrix} 1 & 2 & 3 \end{smallmatrix}.max() = 3 \quad \begin{smallmatrix} 1 & 2 & 3 \end{smallmatrix}.argmax() = 2$
$\text{np.round}(\begin{smallmatrix} 1.1 & 1.5 & 1.9 & 2.5 \end{smallmatrix})$	$= \begin{smallmatrix} 1. \\ 2. \\ 2. \\ 2. \end{smallmatrix}$	$\begin{smallmatrix} 1 & 2 & 3 \end{smallmatrix}.min() = 1 \quad \begin{smallmatrix} 1 & 2 & 3 \end{smallmatrix}.argmin() = 0$
		$\begin{smallmatrix} 1 & 2 & 3 \end{smallmatrix}.sum() = 6 \quad \begin{smallmatrix} 1 & 2 & 3 \end{smallmatrix}.mean() = 2$
		$\begin{smallmatrix} 1 & 2 & 3 \end{smallmatrix}.var() = 0.67 \quad \begin{smallmatrix} 1 & 2 & 3 \end{smallmatrix}.std() = 0.82$
		$S^2 = \frac{1}{n} \sum_{i=1}^n (X_i - \bar{X})^2, \quad \bar{X} = \frac{1}{n} \sum_{i=1}^n X_i$
		$a = 2 \pm 0.82$

27

## The axis



28

# Matrix operations

$$\begin{array}{c} \begin{array}{cc|cc} 1 & 2 & + & 1 & 0 \\ 3 & 4 & & 0 & 1 \end{array} = \begin{array}{cc} 2 & 2 \\ 3 & 5 \end{array} \\ \begin{array}{cc|cc} 1 & 2 & - & 1 & 0 \\ 3 & 4 & & 0 & 1 \end{array} = \begin{array}{cc} 0 & 2 \\ 3 & 3 \end{array} \end{array}$$

$$\begin{array}{c} \begin{array}{cc|cc} 1 & 2 & * & 2 & 0 \\ 3 & 4 & & 0 & 2 \end{array} = \begin{array}{cc} 2 & 0 \\ 0 & 8 \end{array} \\ \begin{array}{cc|cc} 1 & 2 & @ & 2 & 0 \\ 3 & 4 & & 0 & 2 \end{array} = \begin{array}{cc} 2 & 4 \\ 6 & 8 \end{array} \\ \begin{array}{cc|cc} 1 & 2 & / & 2 & 1 \\ 3 & 4 & & 1 & 2 \end{array} = \begin{array}{cc} 0.5 & 2. \\ 3. & 2. \end{array} \end{array}$$

$$\begin{array}{cc|cc} 1 & 2 & ** & 2 & 1 \\ 3 & 4 & & 1 & 2 \end{array} = \begin{array}{cc} 1 & 2 \\ 3 & 16 \end{array}$$

29

# Broadcasting

$$\begin{array}{c} \begin{array}{ccc|ccc} 1 & 2 & 3 & 9 & 9 & 9 & / \\ 4 & 5 & 6 & 9 & 9 & 9 & \\ 7 & 8 & 9 & 9 & 9 & 9 & \end{array} = \begin{array}{ccc} .1 & .2 & .3 \\ .4 & .5 & .7 \\ .8 & .9 & 1. \end{array} \\ \text{normalization} \\ \begin{array}{ccc|ccc} 1 & 2 & 3 & -1 & 0 & 1 & * \\ 4 & 5 & 6 & -1 & 0 & 1 & \\ 7 & 8 & 9 & -1 & 0 & 1 & \end{array} = \begin{array}{ccc} -1 & 0 & 3 \\ -4 & 0 & 6 \\ -7 & 0 & 9 \end{array} \\ \text{multiplying several columns at once} \\ \begin{array}{ccc|ccc} 1 & 2 & 3 & 3 & 3 & 3 & / \\ 4 & 5 & 6 & 6 & 6 & 6 & \\ 7 & 8 & 9 & 9 & 9 & 9 & \end{array} = \begin{array}{ccc} .3 & .7 & 1. \\ .6 & .8 & 1. \\ .8 & .9 & 1. \end{array} \\ \text{row-wise normalization} \\ \begin{array}{ccc|ccc} 1 & 2 & 3 & 1 & 1 & 1 & * \\ 1 & 2 & 3 & 2 & 2 & 2 & \\ 1 & 2 & 3 & 3 & 3 & 3 & \end{array} = \begin{array}{ccc} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{array} \\ \text{outer product} \end{array}$$

30

# General broadcasting rules

- When operating on two arrays, NumPy compares their shapes element-wise
  - starts with the rightmost dimension and works backwards
  - two dimensions are compatible when:
    - they are equal, or
    - one of them is 1
  - otherwise a ValueError: operands could not be broadcast together exception is thrown
- Input arrays do not need to have the same number of dimensions
  - resulting array will have the same number of dimensions as the input array with the greatest number of dimensions
  - the size of each dimension is the largest size of the corresponding dimension among the input arrays
  - missing dimensions are assumed to have size one

A (4d array): 8 x 1 x 6 x 1  
 B (3d array): 7 x 1 x 5  
 Result (4d array): 8 x 7 x 6 x 5

A (2d array): 5 x 4  
 B (1d array): 1  
 Result (2d array): 5 x 4

A (2d array): 5 x 4  
 B (1d array): 4  
 Result (2d array): 5 x 4

A (3d array): 15 x 3 x 5  
 B (3d array): 15 x 1 x 5  
 Result (3d array): 15 x 3 x 5

A (3d array): 15 x 3 x 5  
 B (2d array): 3 x 5  
 Result (3d array): 15 x 3 x 5

A (3d array): 15 x 3 x 5  
 B (2d array): 3 x 1  
 Result (3d array): 15 x 3 x 5

A (1d array): 3  
 B (1d array): 4 # mismatch

A (2d array): 2 x 1  
 B (3d array): 8 x 4 x 3 # mismatch

31

32

## Must know your shapes ...

$$\begin{array}{c} \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} \\ @ \\ \begin{matrix} 1 & 2 & 3 \end{matrix} \end{array} = \begin{matrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{matrix} \quad \text{outer product}$$
$$\begin{matrix} 1 & 2 & 3 \end{matrix} \\ @ \\ \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} = 14 \quad \text{inner (or dot) product}$$

## Concatenation

$$\begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \text{a} \\ \begin{matrix} 5 & 6 & 7 & 8 \end{matrix} \\ \begin{matrix} 9 & 10 & 11 & 12 \end{matrix} \end{array} \rightarrow \begin{array}{c} \begin{matrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{matrix} \\ \text{c} \end{array} \quad \text{np.hstack((a, c))}$$
$$\begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \text{b} \\ \begin{matrix} 5 & 6 & 7 & 8 \end{matrix} \end{array} \quad \downarrow \quad \text{np.vstack((a, b))}$$
$$\begin{matrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{matrix}$$

33

34