

# 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)

$$\mathbf{x} \in \mathbb{R}^n \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

## 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

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

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

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

5

## Matrices and tensors

### • Matrices

- ✓ 2-D array of elements

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

### • Tensors

- ✓ homogeneous arrays that may have **zero (scalar)** or **more** dimensions

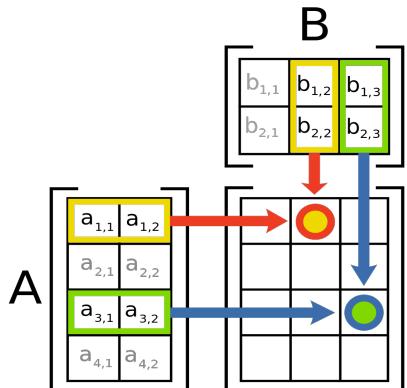
6

## 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)

7

## Dot product

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

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

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

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

8

## Matrix transpose

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

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

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

9

## Identity matrix

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

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

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

10

## 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

# 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
```

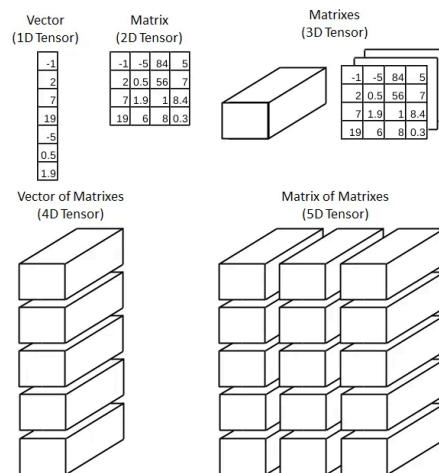
15

# 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

# Tensors



<https://towardsdatascience.com/deep-learning-introduction-to-tensors-tensorflow-36ce3663528f>

16

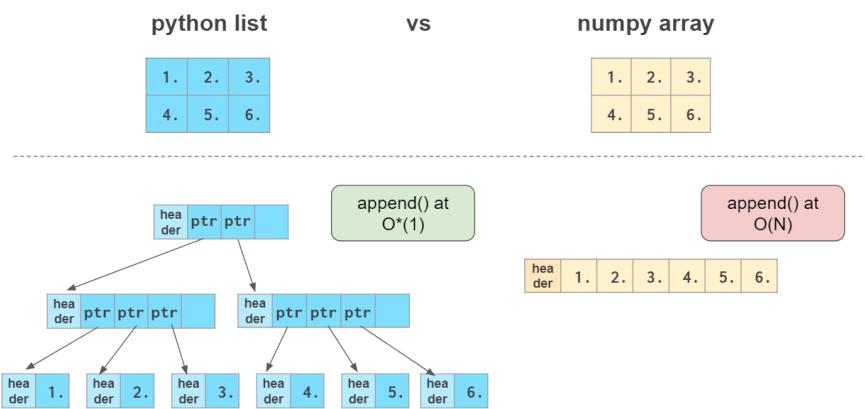
The following figures are from:

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

<https://betterprogramming.pub/numpy-illustrated-the-visual-guide-to-numpy-3b1d4976de1d>

18

## Homogeneous and fixed-length



19

## 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])`

20

## Creating numpy arrays

`a = np.array([1., 2., 3.])` →

`np.array([1, 2, 3])` →

`np.zeros(3)` →

`np.zeros_like(a)` →

`np.ones(3)` →

`np.ones_like(a)` →

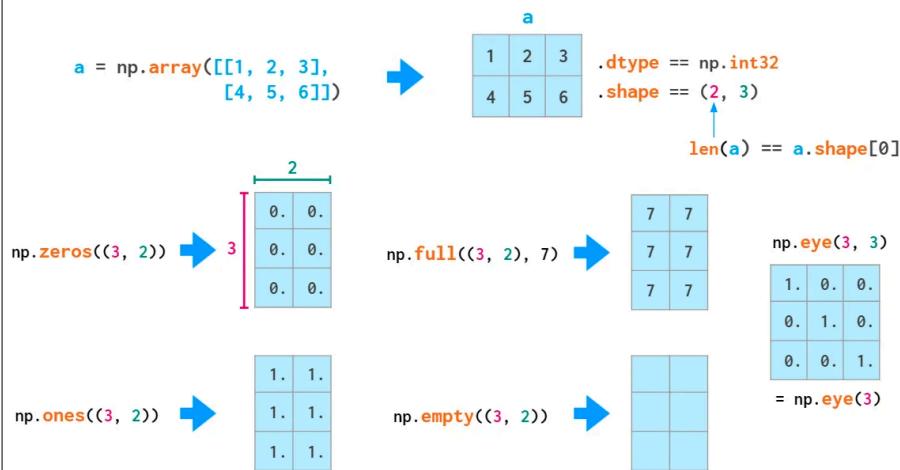
`np.empty(3)` →

`np.empty_like(a)` →

`np.full(3, 7.)` →

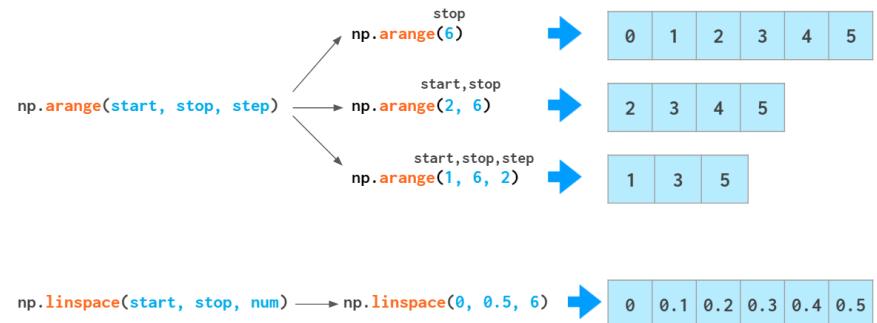
`np.full_like(a, 7)` →

## Creating numpy arrays



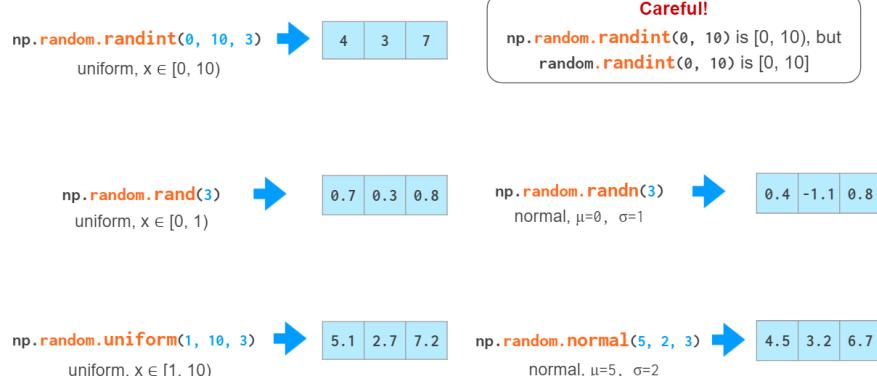
21

## Creating numpy arrays



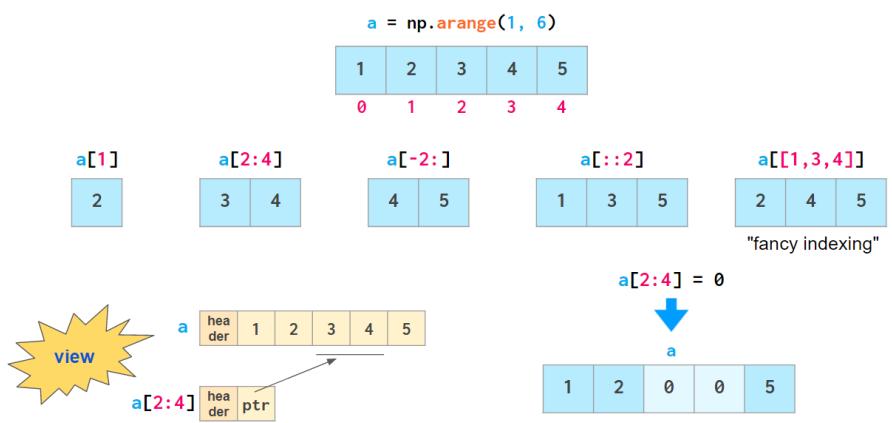
22

## Creating numpy arrays



23

## Indexing



24

# Careful when making copies

```
python list           vs          numpy array

a = [1, 2, 3]
b = a               # no copy
c = a[:]            # copy
d = a.copy()        # copy
```

```
a = np.array([1, 2, 3])
b = a               # no copy
c = a[:]            # no copy!!!
d = a.copy()        # copy
```

25

# Boolean indexing

a	1	2	3	4	5	6	7	6	5	4	3	2	1
a > 5	False	False	False	False	False	True	True	True	False	False	False	False	False

np.any(a > 5)      a[a > 5]      np.all(a > 5)

a	1	2	3	4	5	0	0	0	5	4	3	2	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---

True

6 7 6

False

a[a &gt; 5] = 0

a	1	2	0	0	0	6	7	6	0	0	0	2	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---

a[(a &gt;= 3) &amp; (a &lt;= 5)] = 0

& and  
| or  
^ xor  
~ not

26

# Vector operations

Element-wise operations

$$\begin{array}{l} \begin{array}{rcl} \begin{matrix} 1 & 2 \end{matrix} + \begin{matrix} 4 & 8 \end{matrix} & = & \begin{matrix} 5 & 10 \end{matrix} \\ \begin{matrix} 1 & 2 \end{matrix} - \begin{matrix} 4 & 8 \end{matrix} & = & \begin{matrix} -3 & -6 \end{matrix} \\ \begin{matrix} 3 & 4 \end{matrix} ** \begin{matrix} 2 & 3 \end{matrix} & = & \begin{matrix} 9 & 16 \end{matrix} \end{array} & \begin{array}{rcl} \begin{matrix} 4 & 8 \end{matrix} * \begin{matrix} 2 & 5 \end{matrix} & = & \begin{matrix} 8 & 40 \end{matrix} \\ \begin{matrix} 4 & 8 \end{matrix} / \begin{matrix} 2 & 5 \end{matrix} & = & \begin{matrix} 2.8 & 1.6 \end{matrix} \text{ np.float64} \\ \begin{matrix} 4 & 8 \end{matrix} // \begin{matrix} 2 & 5 \end{matrix} & = & \begin{matrix} 2 & 1 \end{matrix} \text{ np.int32} \end{array} \end{array}$$

Broadcasting scalars

$$\begin{array}{l} \begin{array}{rcl} \begin{matrix} 1 & 2 \end{matrix} + 3 & = & \begin{matrix} 4 & 5 \end{matrix} \\ \begin{matrix} 1 & 2 \end{matrix} - 3 & = & \begin{matrix} -2 & -1 \end{matrix} \\ 3 \quad 4 \quad ** \quad 2 & = & \begin{matrix} 9 & 16 \end{matrix} \end{array} & \begin{array}{rcl} \begin{matrix} 1 & 2 \end{matrix} * \begin{matrix} 3 \end{matrix} & = & \begin{matrix} 3 & 6 \end{matrix} \\ \begin{matrix} 1 & 2 \end{matrix} / 3 & = & \begin{matrix} 0.33 & 0.67 \end{matrix} \text{ np.float64} \\ \begin{matrix} 1 & 2 \end{matrix} // 2 & = & \begin{matrix} 0 & 1 \end{matrix} \text{ np.int32} \end{array} \end{array}$$

# Math functions

$$\begin{array}{ll} a^2 & = \begin{matrix} 2 & 3 \end{matrix} ** 2 = \begin{matrix} 4 & 9 \end{matrix} \\ \sqrt{a} & = \text{np.sqrt}(\begin{matrix} 4 & 9 \end{matrix}) = \begin{matrix} 2. & 3. \end{matrix} \\ e^a & = \text{np.exp}(\begin{matrix} 1 & 2 \end{matrix}) = \begin{matrix} 2.72 & 7.39 \end{matrix} \\ \ln a & = \text{np.log}(\text{np.e} \text{ np.e**2}) = \begin{matrix} 1. & 2. \end{matrix} \end{array} \quad \begin{array}{ll} \vec{a} \cdot \vec{b} & = \text{np.dot}(\begin{matrix} 1 & 2 \end{matrix}, \begin{matrix} 3 & 4 \end{matrix}) \\ & = \begin{matrix} 1 & 2 \end{matrix} @ \begin{matrix} 3 & 4 \end{matrix} = \begin{matrix} 11 \end{matrix} \\ \vec{a} \times \vec{b} & = \text{np.cross}(\begin{matrix} 2 & 0 & 0 \end{matrix}, \begin{matrix} 0 & 3 & 0 \end{matrix}) = \begin{matrix} 0 & 0 & 6 \end{matrix} \end{array}$$

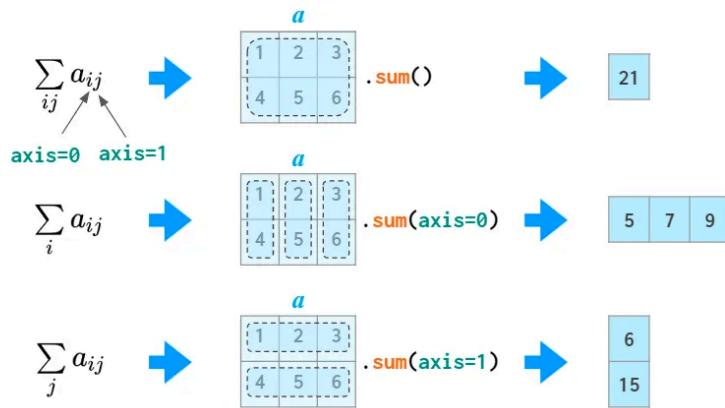
$$\begin{array}{ll} \text{np.max}(\begin{matrix} 1 & 2 & 3 \end{matrix}) & = \begin{matrix} 3 \end{matrix} \\ \text{np.floor}(\begin{matrix} 1.1 & 1.5 & 1.9 & 2.5 \end{matrix}) & = \begin{matrix} 1. & 1. & 1. & 2. \end{matrix} \\ \text{np.ceil}(\begin{matrix} 1.1 & 1.5 & 1.9 & 2.5 \end{matrix}) & = \begin{matrix} 2. & 2. & 2. & 3. \end{matrix} \\ \text{np.round}(\begin{matrix} 1.1 & 1.5 & 1.9 & 2.5 \end{matrix}) & = \begin{matrix} 1. & 2. & 2. & 2. \end{matrix} \end{array} \quad \begin{array}{ll} \begin{matrix} 1 & 2 & 3 \end{matrix} .\max() & = \begin{matrix} 3 \end{matrix} \quad \begin{matrix} 1 & 2 & 3 \end{matrix} .\argmax() = \begin{matrix} 2 \end{matrix} \\ \begin{matrix} 1 & 2 & 3 \end{matrix} .\min() & = \begin{matrix} 1 \end{matrix} \quad \begin{matrix} 1 & 2 & 3 \end{matrix} .\argmin() = \begin{matrix} 0 \end{matrix} \\ \begin{matrix} 1 & 2 & 3 \end{matrix} .\sum() & = \begin{matrix} 6 \end{matrix} \quad \begin{matrix} 1 & 2 & 3 \end{matrix} .\mean() = \begin{matrix} 2 \end{matrix} \\ \begin{matrix} 1 & 2 & 3 \end{matrix} .\var() & = \begin{matrix} 0.67 \end{matrix} \quad \begin{matrix} 1 & 2 & 3 \end{matrix} .\std() = \begin{matrix} 0.82 \end{matrix} \end{array}$$

$$S^2 = \frac{1}{n} \sum_{i=1}^n (X_i - \bar{X})^2, \quad \bar{X} = \frac{1}{n} \sum_{i=1}^n X_i$$

$$a = 2 \pm 0.82$$

28

## The axis



By specifying the **axis** parameter you can apply an operation along the specified axis of an array

29

## Matrix operations

$\begin{matrix} 1 & 2 \\ 3 & 4 \end{matrix} + \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix} = \begin{matrix} 2 & 2 \\ 3 & 5 \end{matrix}$	$\begin{matrix} 1 & 2 \\ 3 & 4 \end{matrix} @ \begin{matrix} 2 & 0 \\ 0 & 2 \end{matrix} = \begin{matrix} 2 & 0 \\ 0 & 8 \end{matrix}$
$\begin{matrix} 1 & 2 \\ 3 & 4 \end{matrix} - \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix} = \begin{matrix} 0 & 2 \\ 3 & 3 \end{matrix}$	$\begin{matrix} 1 & 2 \\ 3 & 4 \end{matrix} / \begin{matrix} 2 & 0 \\ 0 & 2 \end{matrix} = \begin{matrix} 2 & 4 \\ 6 & 8 \end{matrix}$
$\begin{matrix} 1 & 2 \\ 3 & 4 \end{matrix} * \begin{matrix} 2 & 1 \\ 1 & 2 \end{matrix} = \begin{matrix} 0.5 & 2 \\ 3. & 2. \end{matrix}$	
$\begin{matrix} 1 & 2 \\ 3 & 4 \end{matrix} ** \begin{matrix} 2 & 1 \\ 1 & 2 \end{matrix} = \begin{matrix} 1 & 2 \\ 3 & 16 \end{matrix}$	

Not every operator or operation in NumPy is element-wise

30

## Broadcasting

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

## General broadcasting rules

- Numpy follows a set of rules to determine how the smaller array is **broadcasted** to match the shape of the larger array
  - ✓ start with the rightmost dimension and work backwards
  - ✓ two dimensions are **compatible** if they are equal, or one of them is 1
    - otherwise throw a ValueError exception: “operands could not be broadcast together”
  - ✓ if the arrays have a different number of dimensions, pad the smaller array's shape with ones on its left
- 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

31

32

## Practice

Arrays	Result
$8 \times 1 \times 6 \times 1$ $7 \times 1 \times 5$	$8 \times 7 \times 6 \times 5$
$5 \times 4$ 1	$5 \times 4$
$5 \times 4$ 4	$5 \times 4$
$15 \times 3 \times 5$ $15 \times 1 \times 5$	$15 \times 3 \times 5$
$15 \times 3 \times 5$ $3 \times 5$	$15 \times 3 \times 5$
$15 \times 3 \times 5$ $3 \times 1$	$15 \times 3 \times 5$
3 4	mismatch
2 1 8 $\times 4 \times 3$	mismatch

Arrays	Result
	$4 \times 3$ 3
	$5 \times 4 \times 3$ 1 $\times$ 3
	$8 \times 1 \times 6$ 7 $\times$ 1
	$5 \times 2$ 5
	$3 \times 1$ 3 $\times$ 5
	$2 \times 4 \times 1$ 8
	$6 \times 3$ 3 $\times$ 1
	$3 \times 5 \times 2$ 3

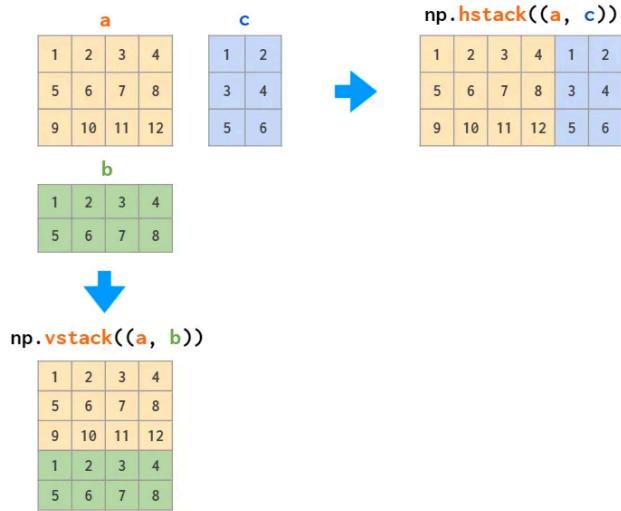
33

## Must know your shapes ...

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

34

## Concatenation



35

## Practice

- › Task
  - › **MSE loss** for a batch of predictions and corresponding ground truth labels
- › Input
  - › predictions (batch\_size, num\_outputs)
  - › ground\_truth (batch\_size, num\_outputs)
- › Output
  - › scalar value representing the average MSE across the batch
- › Hint
  - › element-wise subtraction followed by squaring and averaging

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

36

## Practice

- Task

- ✓ softmax for a batch of input vectors

- Input

- ✓ input\_vectors (batch\_size, num\_inputs)

- Output

- ✓ output\_vectors(batch\_size, num\_inputs)

- Hint

- ✓ consider numerical instability (behavior of the exponential with very small and very large numbers)
  - subtract the max of each input vector
  - ✓ make it efficient by using broadcasting
  - may need `keepdims=True`

$$\begin{matrix} & \frac{e^{x_i}}{\sum_{j=1}^d e^{x_j}} \\ \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix} & \Rightarrow \begin{bmatrix} 0.0117 \\ 0.0317 \\ 0.0861 \\ 0.2341 \\ 0.6364 \end{bmatrix} \end{matrix}$$

37

## Practice

- Task

- ✓ given N data points, find the nearest neighbor to a query data point

- Input:

- ✓ input vector (vector\_dim)
  - ✓ data (num\_vectors, vector\_dim)

- Output:

- ✓ nearest\_neighbor\_idx (scalar)

- Hint:

- ✓ calculate the euclidean distance
  - ✓ use vectorized and broadcasting operations

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^d (x_i - y_i)^2}$$

38