videos on linear algebra

The Essence of Linear Algebra series (3blue1brown) linked below is an excellent resource for reviewing (or learning) basic concepts of linear algebra. The most important concepts to review are vectors (Chapter 1-2), linear transformations and matrices (Chapter 3), matrix multiplication (Chapter 4), non-square matrices as transformations between dimensions (Chapter 8), and dot products (Chapter 9).

Please review these.

https://www.youtube.com/playlist?list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab

Homework 4

posted on Brightspace

due next Mon (Oct 3) at 11:59pm*

(pushed back so I'm not lecturing about topics for a homework due two days later)

(due late on Mon so people can ask last-minute questions that Mon after class)

`Homework4.pdf` (written description)

`Homework4.ipynb` (notebook to use for your solution)

`difdata.csv`

* most homeworks will continue to be due at class time

download from Brightspace

`FileIO.ipynb`
`LogicalIndexing.ipynb`

how to avoid overwriting files in Python

`FileIO.ipynb`

how to avoid overwriting files in Python

Option 1: use a unique file name for every file

easiest way to guarantee a unique is to use a time stamp
as part of the file name

e.g.,

`time.time()` returns the number of seconds since January 1, 1970

```
import time
fname = f'data.{time.time()}.csv'
```

example file name:

```
data.1664128556.5515451.csv
```

how to avoid overwriting files in Python

Option 2: check if the file exists already

```
import os.path

fname = 'data.csv'
if os.path.exists(fname):
    print(f'{fname} already exists!')
else:
    with open(fname, 'w', newline='') as fp:
        write to the file
        write to the file
        write to the file
```

use meaningful data file names

```
subj = 12
sess = 5
fname = f'Search2-{subj}-{sess}-{time.time()}.csv'
print(fname)
```

e.g.,

```
Search2-12-5-1664131357.440385.csv
```

missing data

In experiments, there are often cases where data is missing (or corrupted, or invalid) - how do you encode that in a numpy array and write/read that from a file?

Generally, missing data is coded with a "sentinel value"

- could be a value out of range (e.g., `-1`, `9999`, `np.inf`, `-np.inf`)
- could be `np.nan` (not a number) : *recommended*

`FileIO.ipynb`

Python Data Science Handbook, Handling Missing Data
https://jakevdp.github.io/PythonDataScienceHandbook/03.04-missing-values.html

missing data

Note, however, mathematical/statistical operations on numpy arrays containing `np.nan`

there are some special functions that ignore `nan`

see https://numpy.org/doc/stable/reference/routines.math.html

see https://numpy.org/doc/stable/reference/routines.statistics.html

e.g., `np.nanmin()`, `np.nansum()`, `np.nanmean()`, `np.nanstd()`, `np.nanmedian()`

`FileIO.ipynb`

Python Data Science Handbook, Handling Missing Data
https://jakevdp.github.io/PythonDataScienceHandbook/03.04-missing-values.html

an example of reading and analyzing some behavioral data using numpy arrays

`FileIO.ipynb`
`LogicalIndexing.ipynb`

operations on numpy arrays

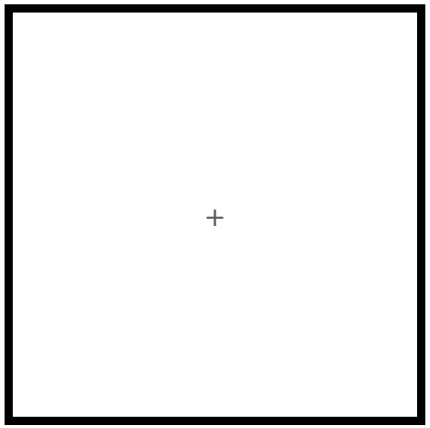
numpy arrays contain "data" (of the same type); in the context of psychology and neuroscience, these could be

- behavioral data (e.g., choices, response times, as a function of subjects, conditions, trials) from an experiment
- 1D signal (time-series) (e.g., intracellular, extracellular, scalp voltage as a function of time) from neural recordings
- 2D signal (time-series) (e.g., N electrical channels x T time steps), 3D signal (e.g., S subjects x N channels x T time)
- 3D eye movements with (x, y) gaze direction by time
- 2D B/W image with (x, y) intensity, 3D color images with (x,y) intensity for R, G, and B color channels
- 3D functional or anatomical brain scan (x, y, z)
- 4D functional or anatomical brain scan (x, y, z) by time steps

is there a T?

a simple experiment (visual search)

Trial 1

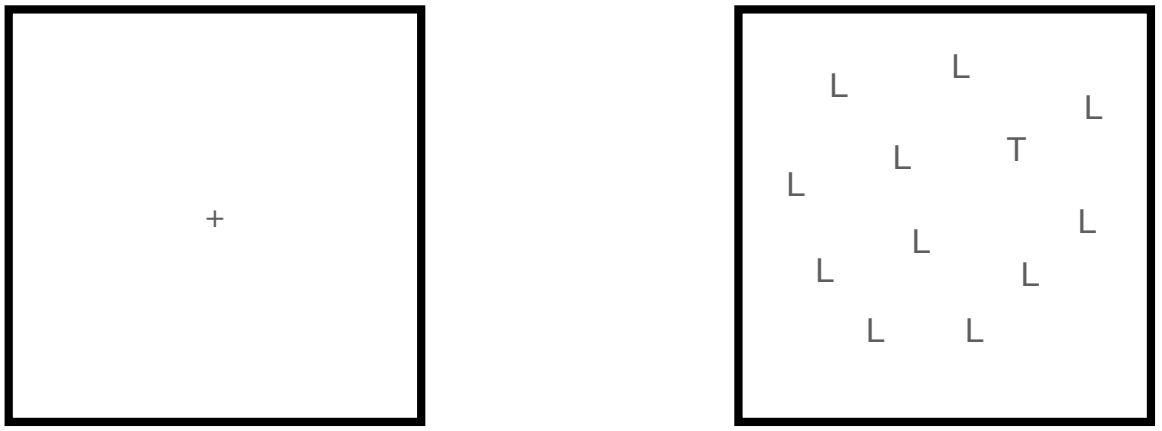


time

is there a T?

a simple experiment (visual search)

Trial 1



time

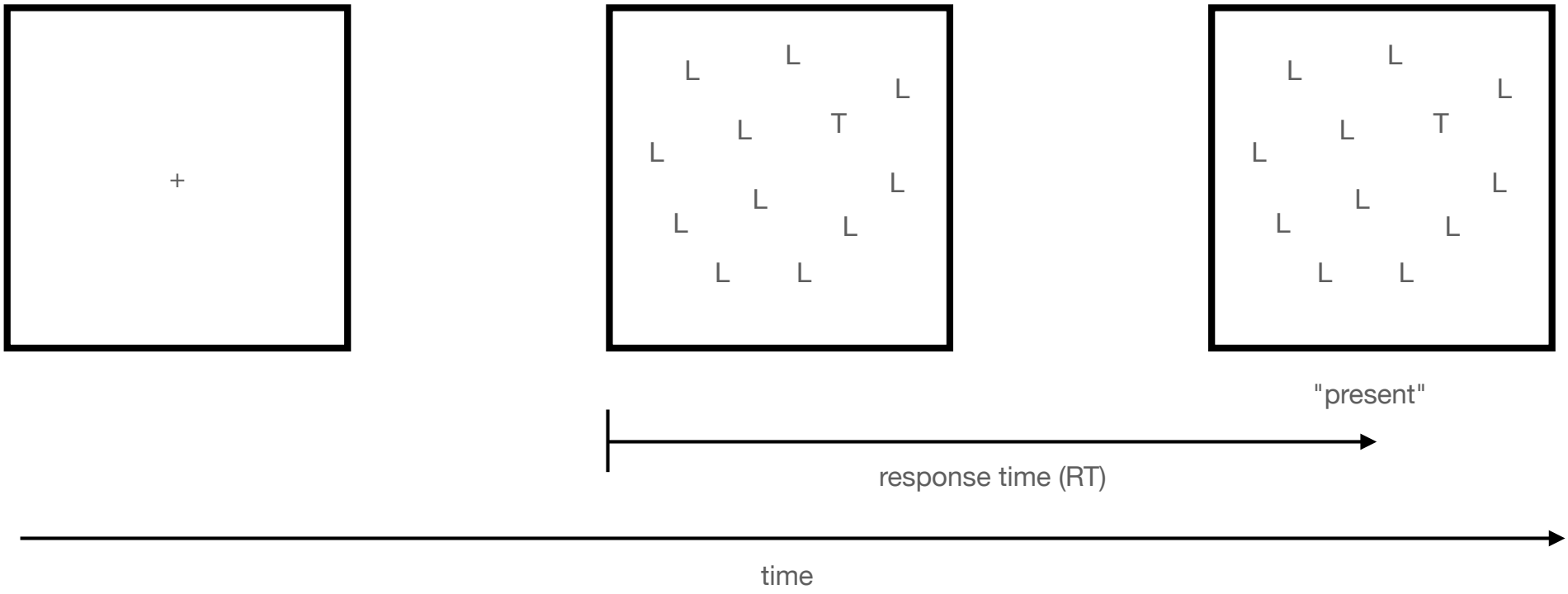
is there a T?

a simple experiment (visual search)

Trial 1

target present condition

correct



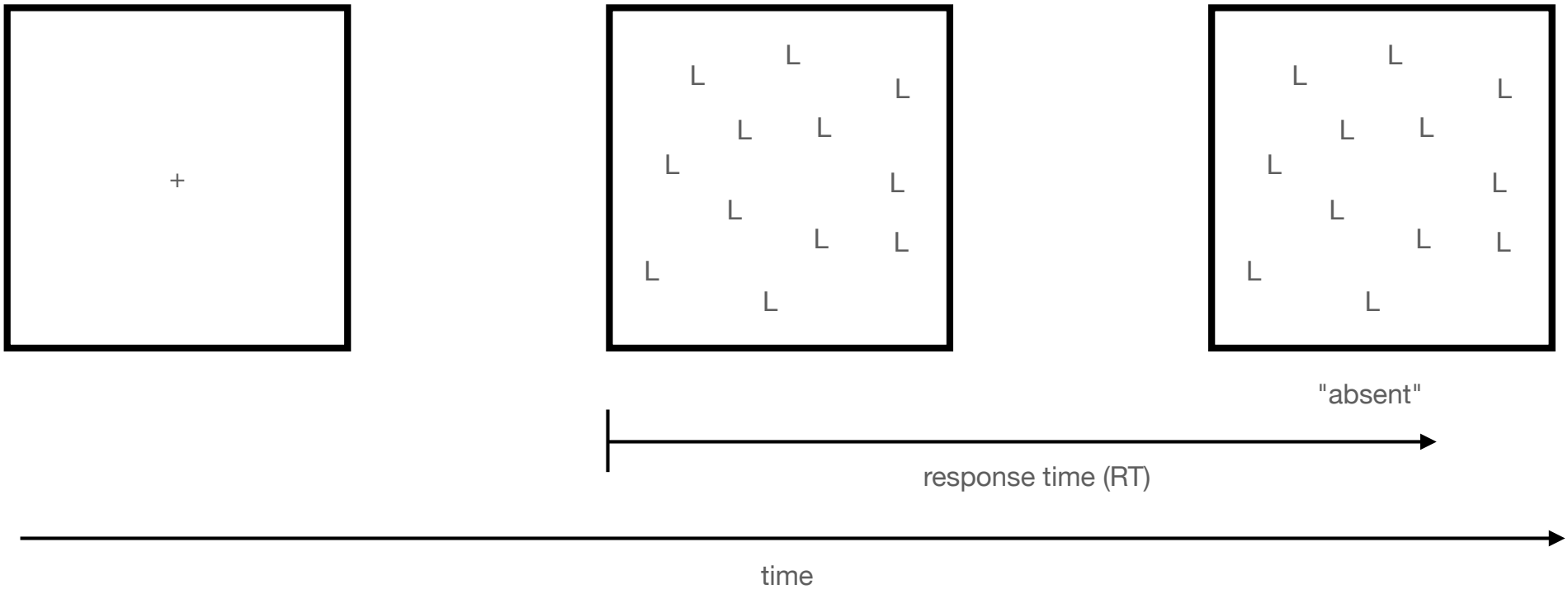
is there a T?

a simple experiment (visual search)

Trial 2

target absent condition

correct



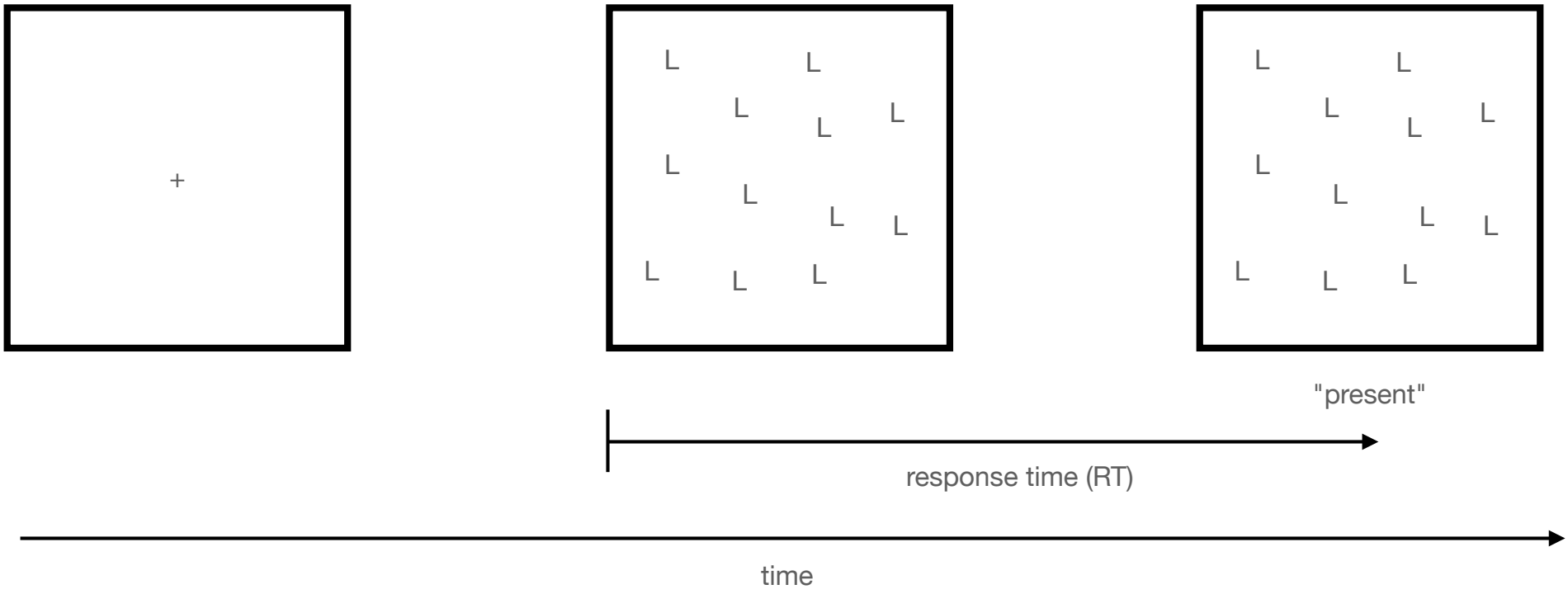
is there a T?

a simple experiment (visual search)

Trial 3

target absent condition

error

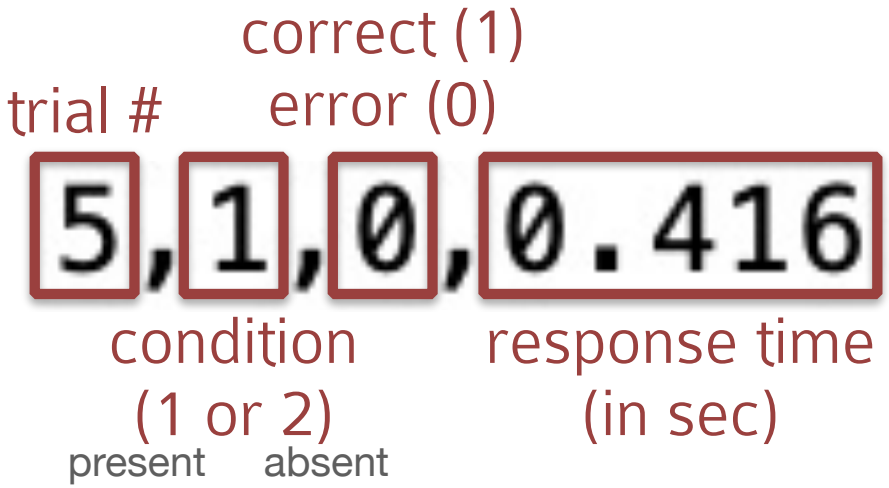


(simulated) data from an experiment with 2 conditions

`difdata.csv` (ASCII text file in CSV format)

one subject's data

each line (except first)



trials →

```
1000
0,1,0,0.386
1,1,1,0.520
2,1,0,0.388
3,2,1,0.419
4,1,1,0.530
5,1,0,0.416
6,1,1,0.393
7,2,1,0.661
8,2,1,1.095
9,1,1,0.514
```

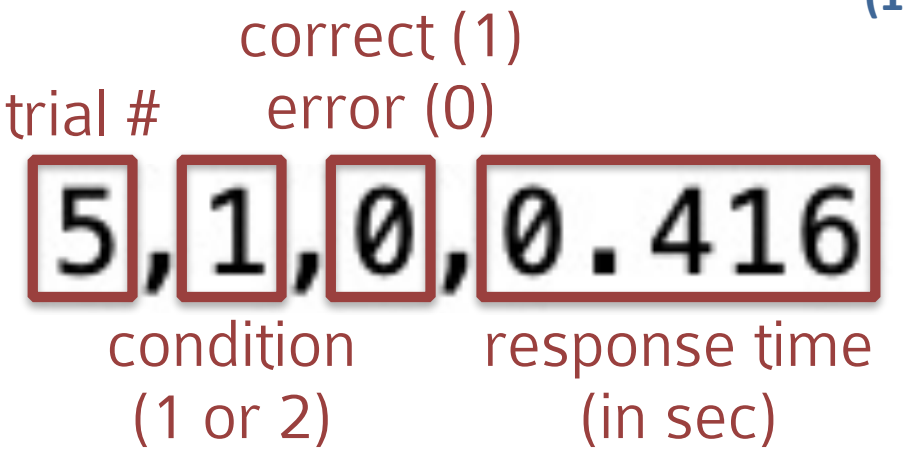
how to write code that processes this file

`difdata.csv` (ASCII text file)

(0) get csv

(1) open file for reading

each line (except first)



trials →

```
1000
0,1,0,0.386
1,1,1,0.520
2,1,0,0.388
3,2,1,0.419
4,1,1,0.530
5,1,0,0.416
6,1,1,0.393
7,2,1,0.661
8,2,1,1.095
9,1,1,0.514
```

(2) read # trials

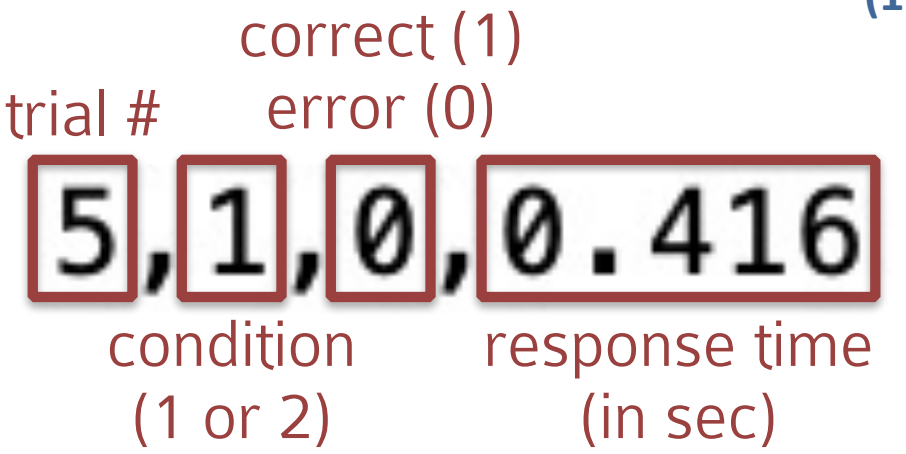
how to write code that processes this file

`difdata.csv` (ASCII text file)

(0) get csv

(1) open file for reading

each line (except first)



trials

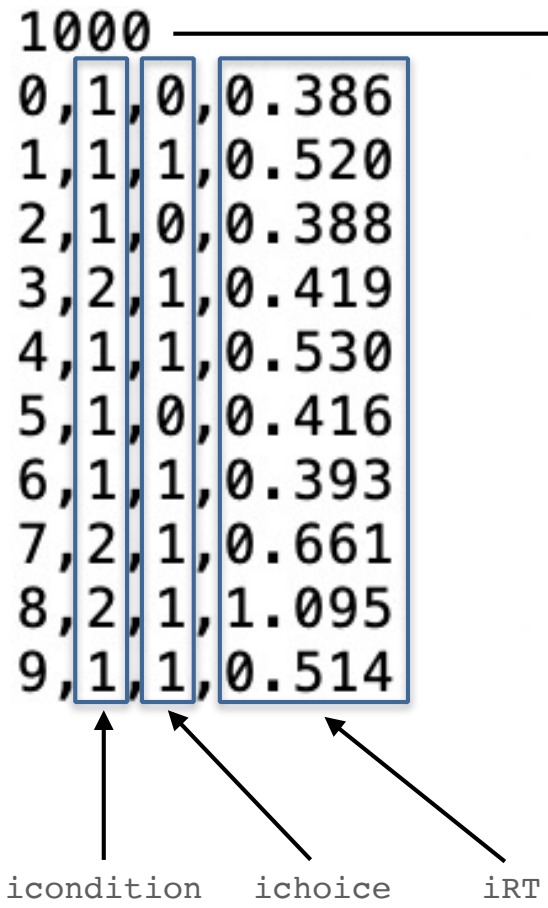
```
1000
0,1,0,0.386
1,1,1,0.520
2,1,0,0.388
3,2,1,0.419
4,1,1,0.530
5,1,0,0.416
6,1,1,0.393
7,2,1,0.661
8,2,1,1.095
9,1,1,0.514
```

(2) read # trials

(3) read each line, filling arrays with condition, response, RT

later, we'll do more

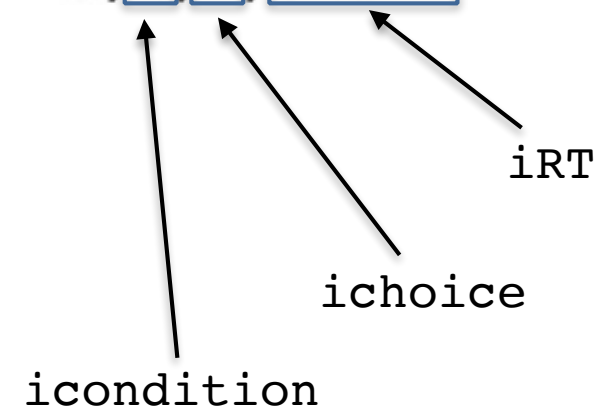
see `FileIO.ipynb`




```python
# read in the data

import csv
import numpy as np

with open('difdata.csv', 'r') as fp:
    # create the "reader" object
    csvreader = csv.reader(fp, delimiter=',')

    # get a line of the file
    row = next(csvreader)

    # that first line is the # trials
    Ntrials = int(row[0])

    # using # trials, preallocate np arrays to hold condition, choice, and RT
    icondition = np.zeros(Ntrials, dtype=int)
    ichoice    = np.zeros(Ntrials, dtype=int)
    iRT        = np.zeros(Ntrials, dtype=float)

    # loop over all trials (all remaining lines in the file)
    for i, row in enumerate(csvreader):
        icondition[i] = int(row[1])
        ichoice[i]    = int(row[2])
        iRT[i]        = float(row[3])
```

Homework 4

```
1000
0,1,0,0.386
1,1,1,0.520
2,1,0,0.388
3,2,1,0.419
4,1,1,0.530
5,1,0,0.416
6,1,1,0.393
7,2,1,0.661
8,2,1,1.095
9,1,1,0.514
```



(a) (6 points) Following the discussion from class, I want you to partition the data so that one two-dimensional array that holds the choices in condition 1 and the choices in condition 2 and another two-dimensional array that holds the RTs in condition 1 and the RTs in condition 2. **First**, do this using for loops. **Second**, do this using logical (Boolean) indexing.

one thing we might want to do is divide up the choice and RT data by condition, creating separate arrays for condition 1 and for condition 2 - how to do that?

logical (Boolean) indexing

```
a = np.array([1, 2, 3, 4, 5])
b = np.array([False, True, False, False, True])
print(a[b])
```

a and b need to be the same size

```
a = np.arange(100)
b = (a % 2) == 0
print(a[b])
```

what does this do?

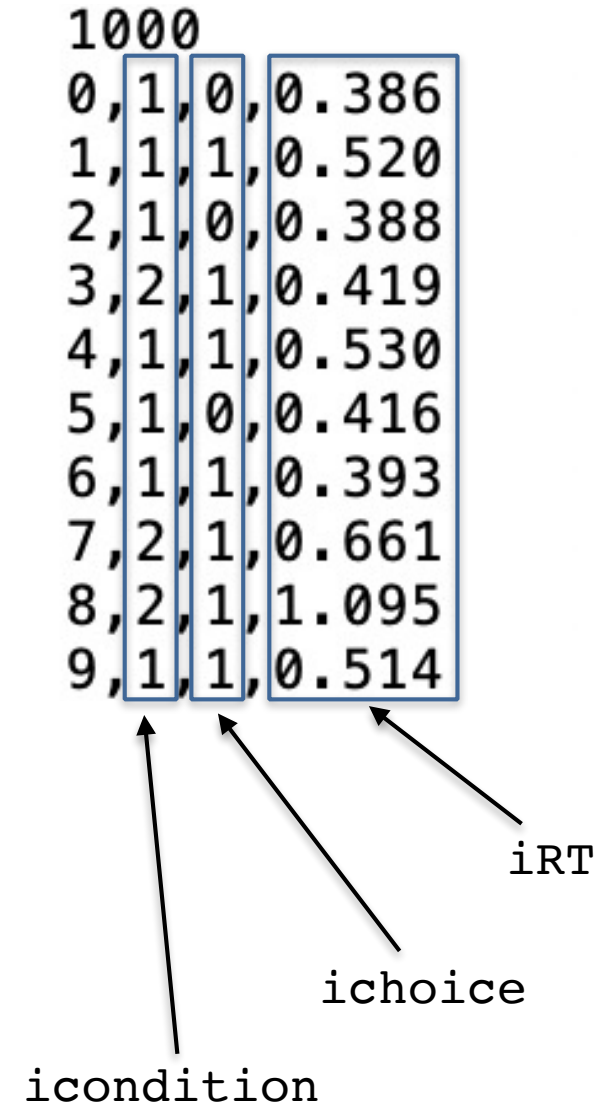
```
print(a[((a % 2) == 0)])
```

this is the same

https://numpy.org/doc/stable/user/basics.indexing.html#boolean-or-mask-index-arrays

see `LogicalIndexing.ipynb`

Homework 4



one thing we might want to do is divide up the choice and RT data by condition, creating separate arrays for condition 1 and for condition 2 - how to do that?

Homework 4

```
1000
0,1,0,0.386
1,1,1,0.520
2,1,0,0.388
3,2,1,0.419
4,1,1,0.530
5,1,0,0.416
6,1,1,0.393
7,2,1,0.661
8,2,1,1.095
9,1,1,0.514
```

iRT

ichoice

icondition

for **Homework 4**, write code that creates a 2x500 array for choice and RT data, separated by experimental condition

choice

0	1	0	1	0	1	1	1
1	1	1	0	0	1	1	0

. . .

RT

0.386	0.520	0.388	0.530
0.419	0.661	1.095	0.570

. . .

one thing we might want to do is divide up the choice and RT data by condition, creating separate arrays for condition 1 and for condition 2 - how to do that?

Homework 4

```
1000
0,1,0,0.386
1,1,1,0.520
2,1,0,0.388
3,2,1,0.419
4,1,1,0.530
5,1,0,0.416
6,1,1,0.393
7,2,1,0.661
8,2,1,1.095
9,1,1,0.514
```

icondition

ichoice

iRT

for **Homework 4**, write code that creates a 2x500 array for choice and RT data, separated by experimental condition

choice

0	1	0	1	0	1	1	1
1	1	1	0	0	1	1	0

. . .

RT

0.386	0.520	0.388	0.530
0.419	0.661	1.095	0.570

. . .

one thing we might want to do is divide up the choice and RT data by condition, creating separate arrays for condition 1 and for condition 2 - how to do that?

Homework 4

```
1000
0,1,0,0.386
1,1,1,0.520
2,1,0,0.388
3,2,1,0.419
4,1,1,0.530
5,1,0,0.416
6,1,1,0.393
7,2,1,0.661
8,2,1,1.095
9,1,1,0.514
```

iRT

ichoice

icondition

for **Homework 4**, write code that creates a 2x500 array for choice and RT data, separated by experimental condition

choice

0	1	0	1	0	1	1	1
1	1	1	0	0	1	1	0

. . .

RT

0.386	0.520	0.388	0.530
0.419	0.661	1.095	0.570

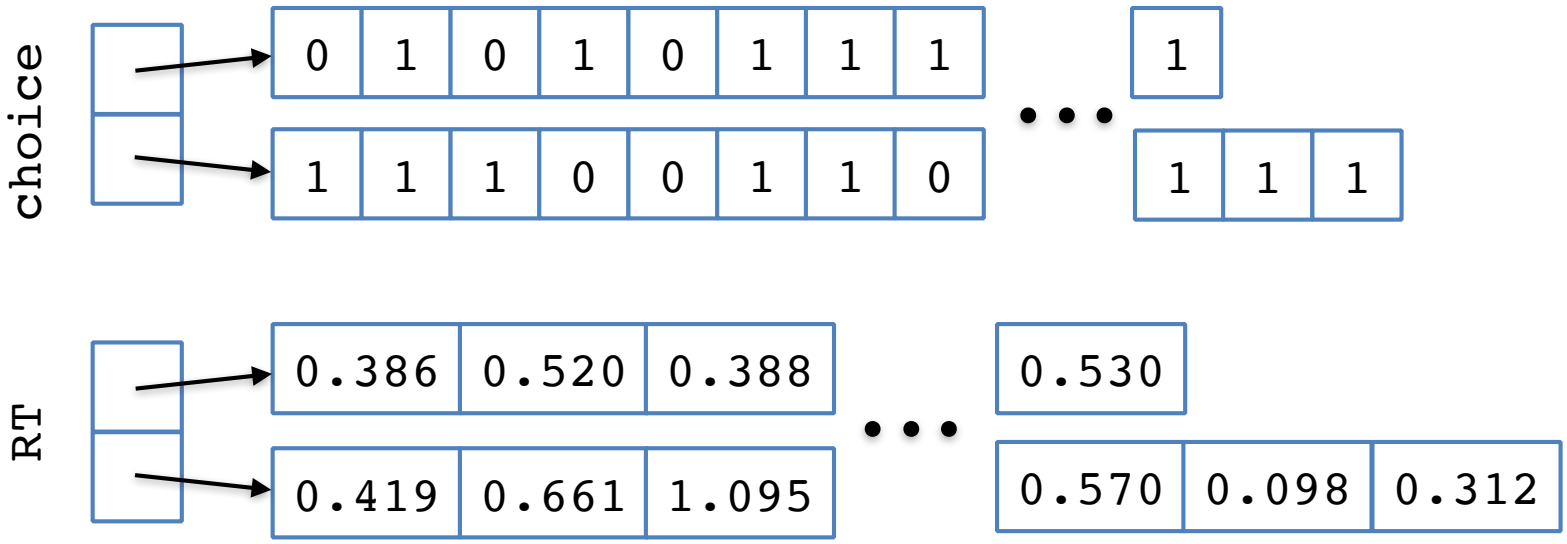
. . .

one thing we might want to do is divide up the choice and RT data by condition, creating separate arrays for condition 1 and for condition 2 - how to do that?

Homework 4

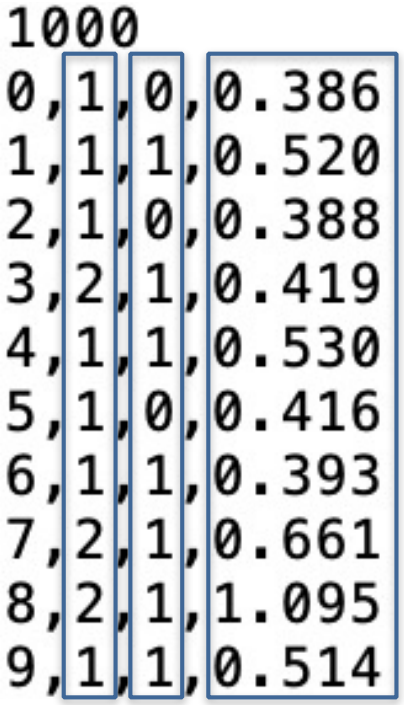
```
1000
0,1,0,0.386
1,1,1,0.520
2,1,0,0.388
3,2,1,0.419
4,1,1,0.530
5,1,0,0.416
6,1,1,0.393
7,2,1,0.661
8,2,1,1.095
9,1,1,0.514
```

how would you remove "outliers", for example trials where RT was outside some bound (e.g., RT<0.100 or RT>1.000)



one thing we might want to do is divide up the choice and RT data by condition, creating separate arrays for condition 1 and for condition 2 - how to do that?

Homework 4



(b) (6 points) Following the discussion from class, I want you to remove "outliers" based on RT, in this case trials where RT is outside some bound (RT<0.100 or RT>1.000). This will result in a list of arrays for the choices with outlier trials removed and a list of arrays for the RTs with the outlier trials removed. **First**, do this using for loops. **Second**, do this using logical (Boolean) indexing. Remember from discussion in class that here you will not be able to use a 2x500 numpy array because the number of resulting trials after removing outliers will be unequal (instead, use a list of numpy arrays).

logical indexing with NaN (missing data)

While there are some math/stats operations that can deal with np.nan entries (missing data) gracefully, there are many other analysis/modeling/ML methods that cannot. We need to remove the NaN vales from the numpy arrays.

`LogicalIndexing.ipynb`

logical indexing with NaN (missing data)

While there are some math/stats operations that can deal with np.nan entries (missing data) gracefully, there are many other analysis/modeling/ML methods that cannot. We need to remove the NaN vales from the numpy arrays.

We cannot do logical indexing like this:

```
data[data != np.nan]
```

(see `LogicalIndexing.ipynb`)

`np.nan` does not work with equality/inequality

need to instead use `np.isnan()`

Command-Line Input

command-line input

input a string

```
name = input('Enter Name :’)
```

entering a non-string requires a type conversion

```
SubjN = int(input('Enter the Subject Number: '))
SessN = int(input('Enter the Session Number: '))
```

Vectors, Matrices, and Linear Algebra

`VectorsMatricesLinearAlgebra.ipynb`

vectors, matrices, and linear algebra

Arrays are merely containers that hold numeric data in an organized way.

Vectors are mathematical entities represented in Python using 1D numpy arrays.

Matrices are mathematical entities represented in Python using 2D numpy arrays. (Tensors are more than 2D.)

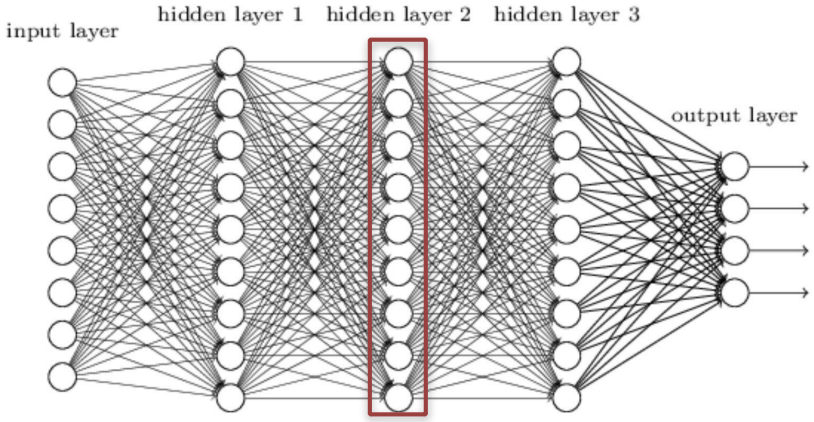
more complex mathematical definition

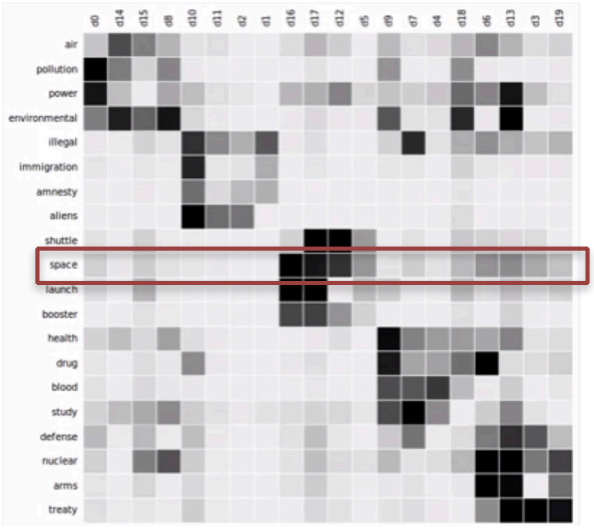
There are operations that apply to vectors and/or matrices that are different from operations that apply to simple arrays.

Vectors

vectors

- some number of psychological measures
- or psychological, social, and demographic measures
- or some number of neural measures





- activity of a units in a layer of a neural network model
- semantic representation of a word from latent semantic analysis

vectors

Terminology can be confusing in that a vector is one-dimensional (as a 1D array) in one sense

```
a = np.array([1, 2])
b = np.array([3, 2, 1])
c = np.array([1, 3, 3, 1])
print(a.shape, b.shape, c.shape)
```

vectors

But we'll also thinking about vectors themselves as residing in a multidimensional space

```
a = np.array([1, 2])
b = np.array([3, 2, 1])
c = np.array([1, 3, 3, 1])
print(a.shape, b.shape, c.shape)
```

2D

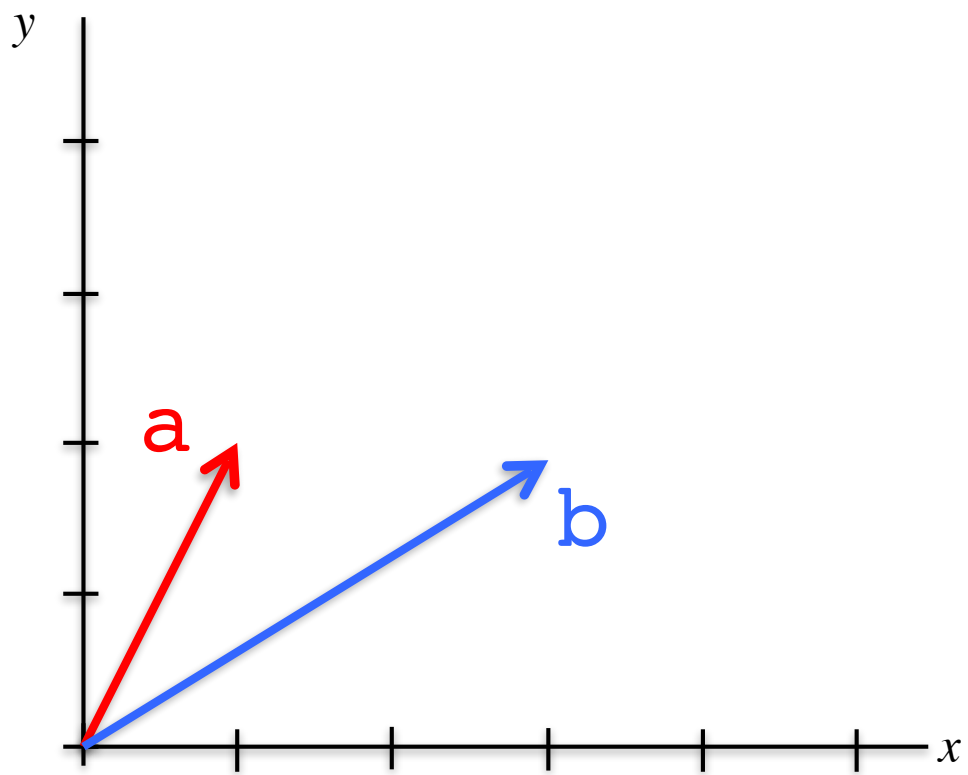
3D

4D

vectors have a magnitude and direction

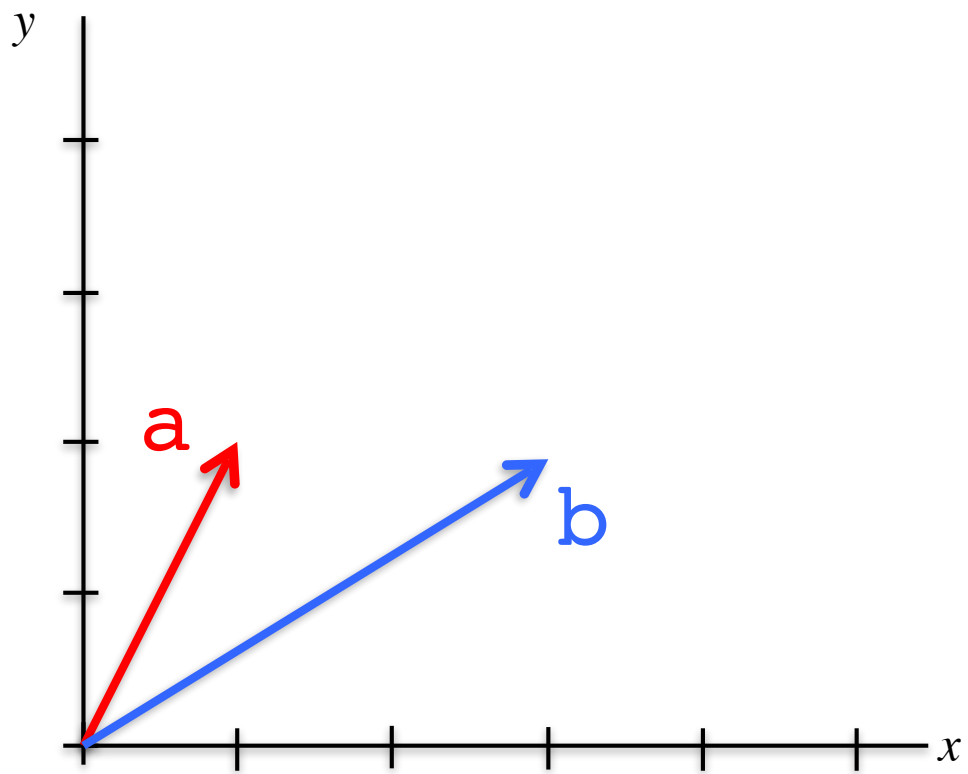
```
a = np.array([1, 2])
b = np.array([3, 2])
```

vectors in 2D space



adding vectors

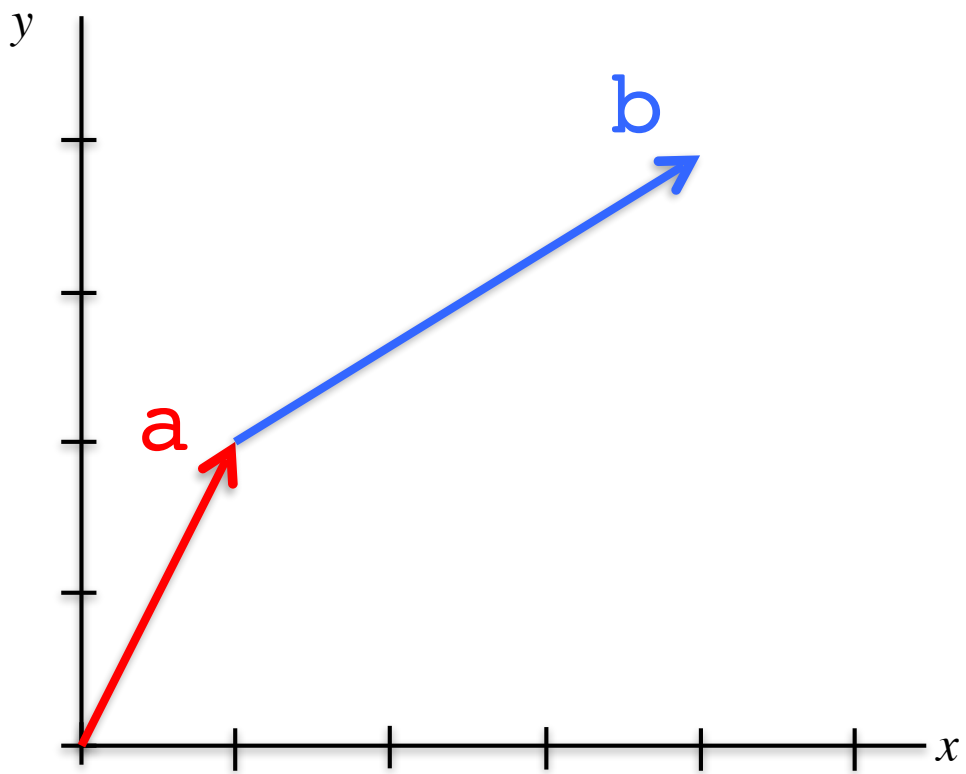
```
a = np.array([1, 2])
b = np.array([3, 2])
a + b
```



adding vectors

same as element-wise array addition

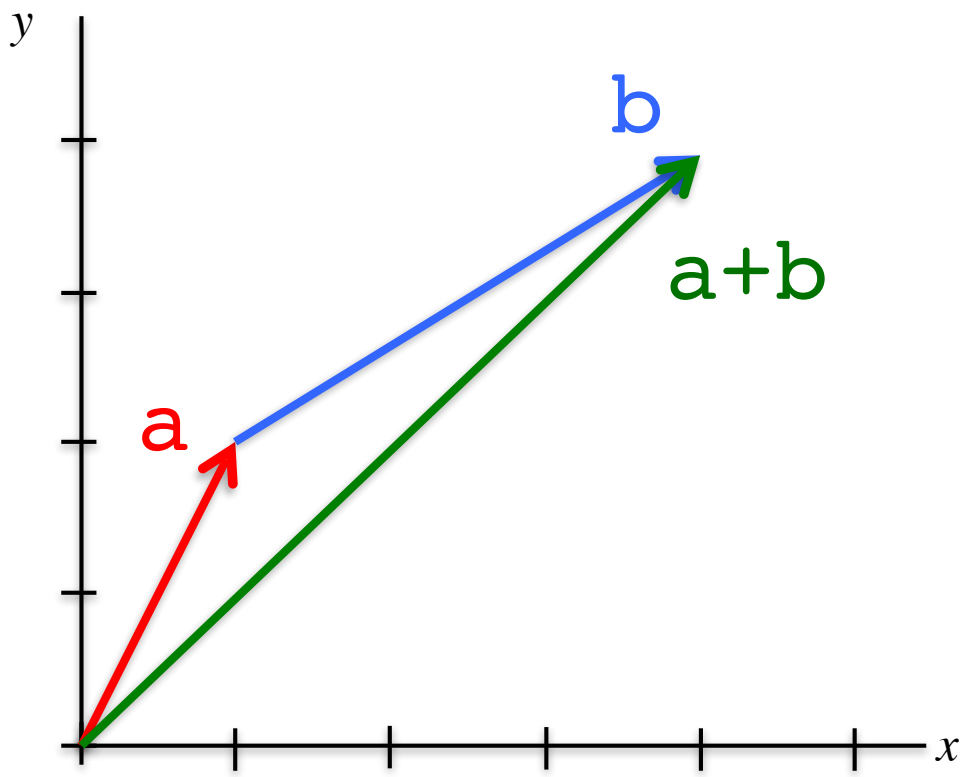
```
a + b
```



adding vectors

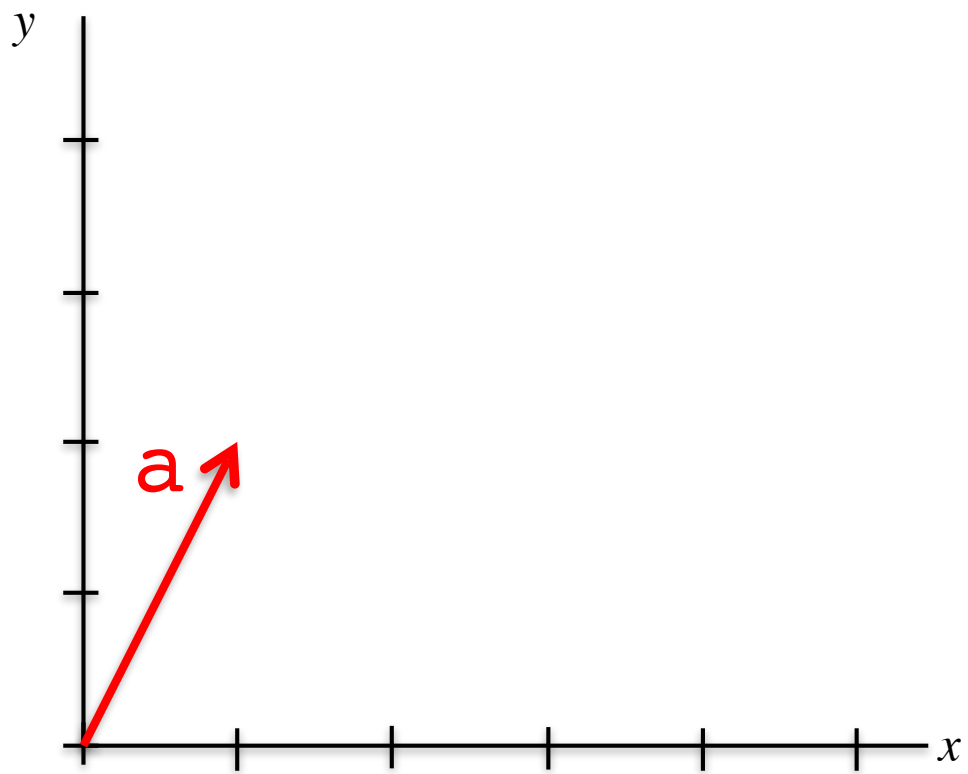
same as element-wise array addition

```
a + b
```



multiplying a vector by a scalar

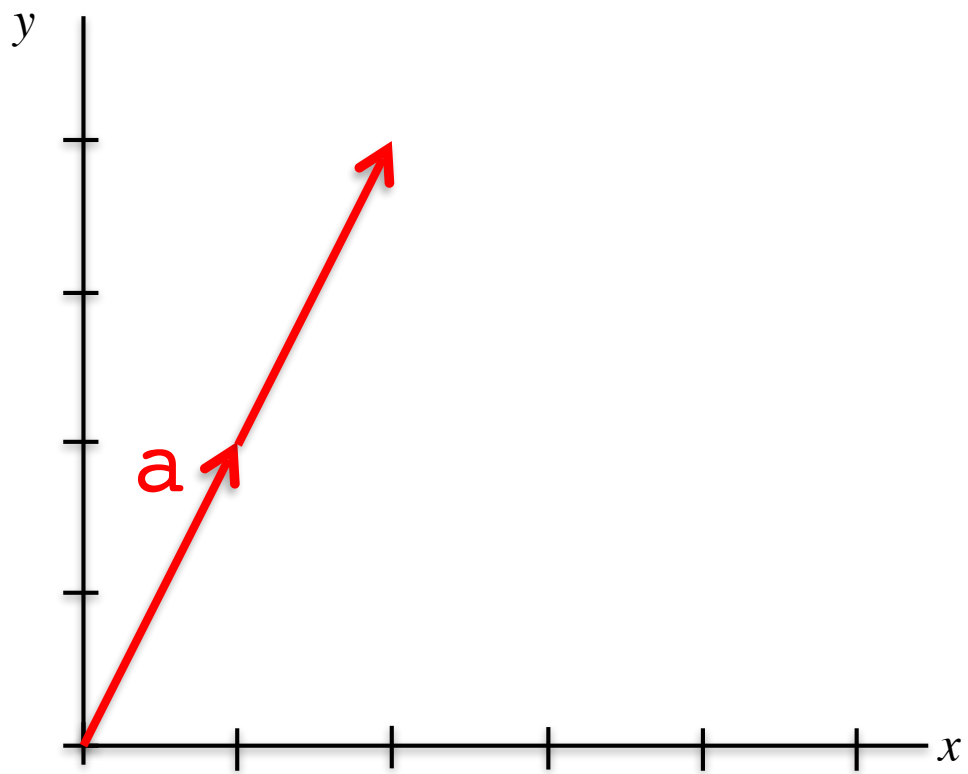
```
a = np.array([1, 2])
b = 2*a
```



multiplying a vector by a scalar

same as multiplying an array by a scalar value

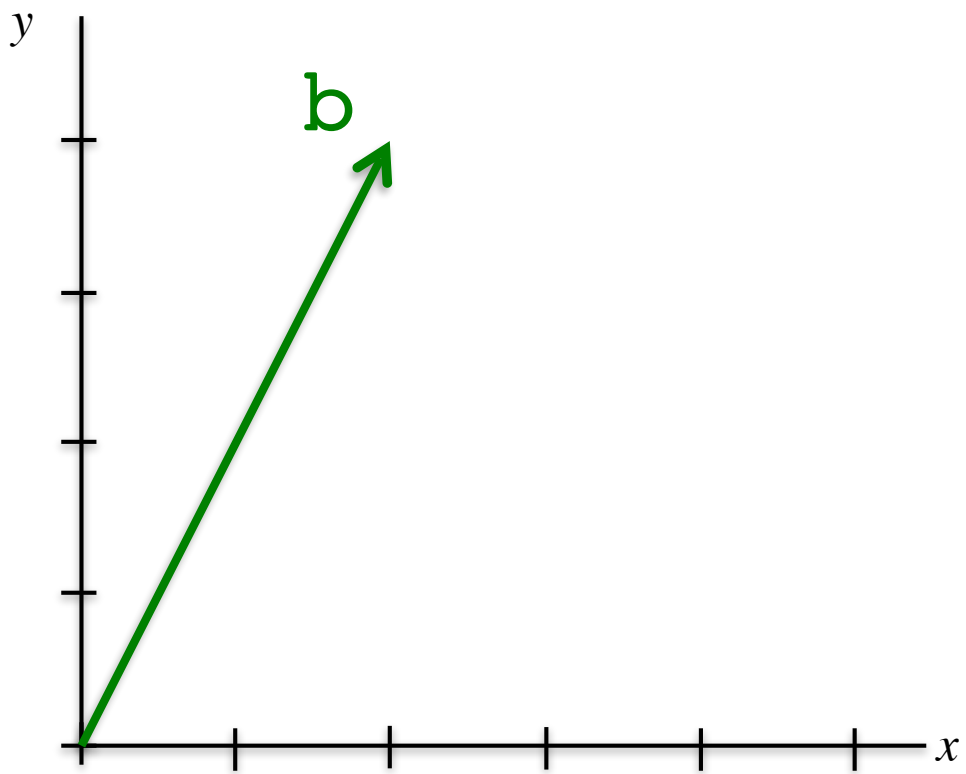
```
a = np.array([1, 2])
b = 2*a
```



multiplying a vector by a scalar

same as multiplying an array by a scalar value

```
a = np.array([1, 2])
b = 2*a
```

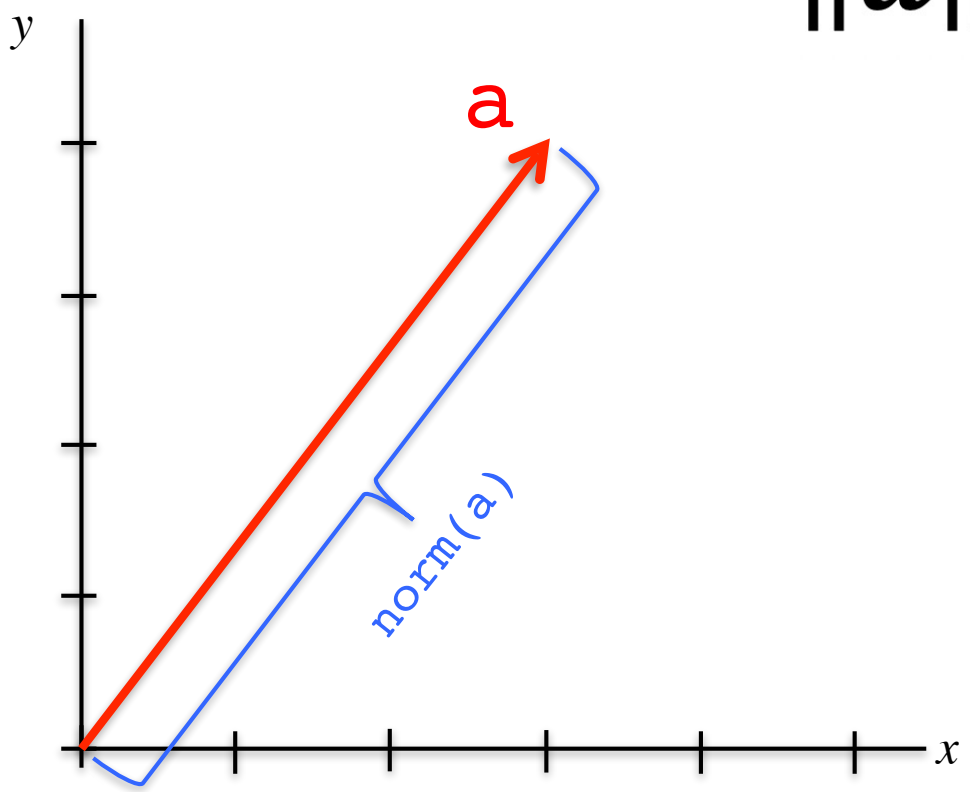


(norm) length of a vector

```
a = np.array([3, 4])
np.linalg.norm(a)
```

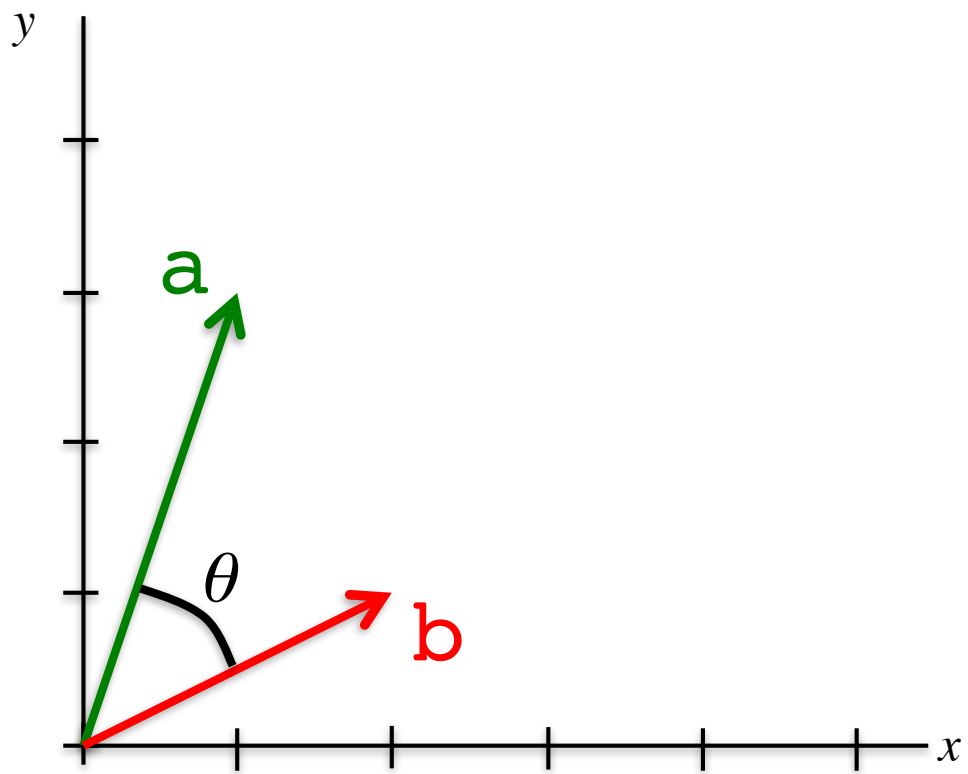
Euclidian norm is found using the Pythagorean Theorem
(also called 2-norm or an L_2 norm)

$\|\boldsymbol{a}\|$ often depicted mathematically



angle between two vectors

```
a = np.array([1, 3])
b = np.array([2, 1])
```

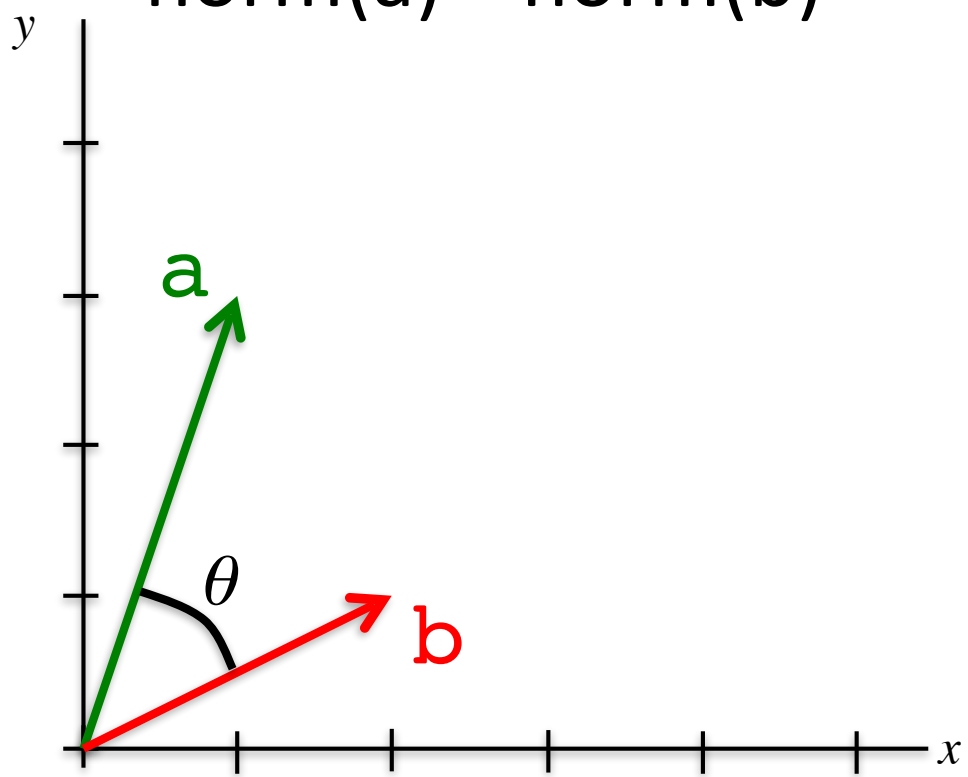


angle between two vectors

$$\cos(\theta) = \frac{\boldsymbol{a} \cdot \boldsymbol{b}}{\|\boldsymbol{a}\|\|\boldsymbol{b}\|}$$

"dot product"

norm(a) norm(b)



angle between two vectors

$$\cos(\theta) = \frac{\boldsymbol{a} \cdot \boldsymbol{b}}{\|\boldsymbol{a}\|\|\boldsymbol{b}\|}$$

$$\theta = \operatorname{acos}\left(\frac{\boldsymbol{a} \cdot \boldsymbol{b}}{\|\boldsymbol{a}\|\|\boldsymbol{b}\|}\right)$$

$$\boldsymbol{a} \cdot \boldsymbol{b} = \sum_i a_i \, b_i$$

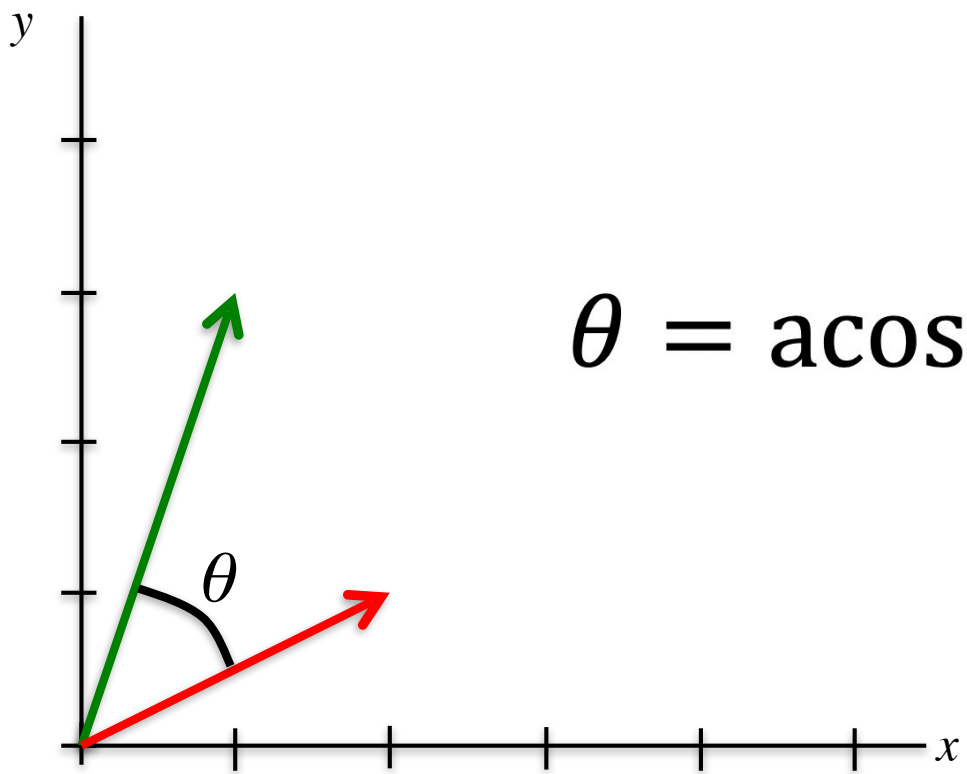
sum of element-wise multiplication

$$\|\boldsymbol{a}\| = \sqrt{\boldsymbol{a} \cdot \boldsymbol{a}}$$

Pythagorean theorem

angle between two vectors

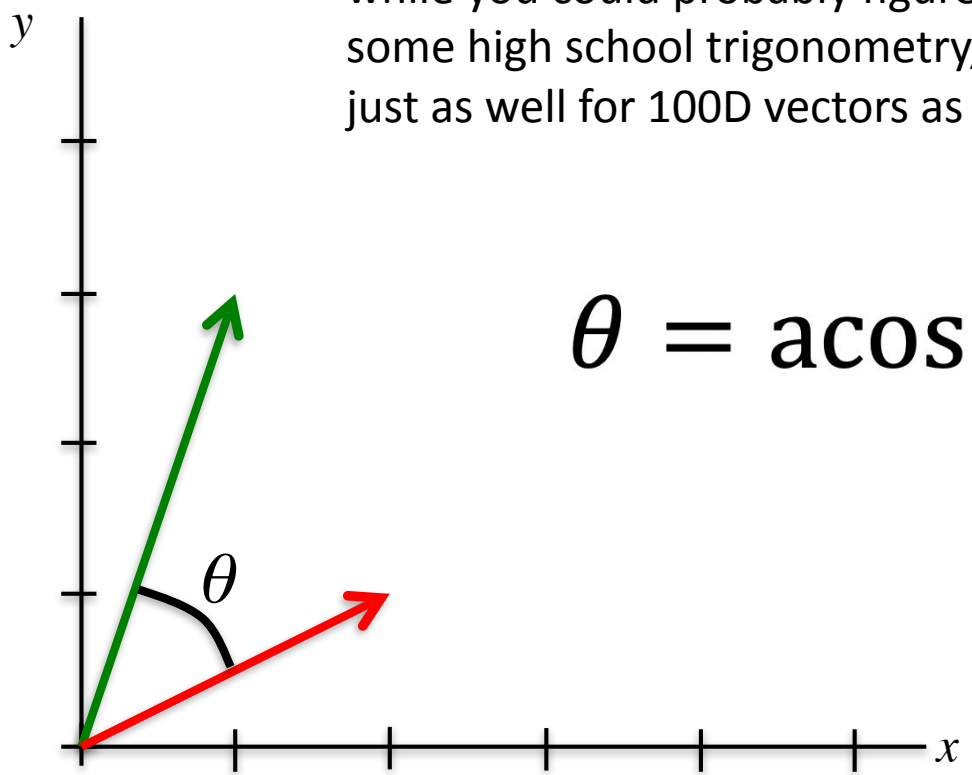
```
an = np.linalg.norm(a)
bn = np.linalg.norm(b)
theta = math.acos(np.dot(a,b)/(an*bn))
print(np.rad2deg(theta))
```



$$\theta = \text{acos}\left(\frac{\boldsymbol{a} \cdot \boldsymbol{b}}{\|\boldsymbol{a}\|\|\boldsymbol{b}\|}\right)$$

angle between two vectors

```
an = np.linalg.norm(a)
bn = np.linalg.norm(b)
theta = math.acos(np.dot(a,b)/(an*bn))
print(np.rad2deg(theta))
```



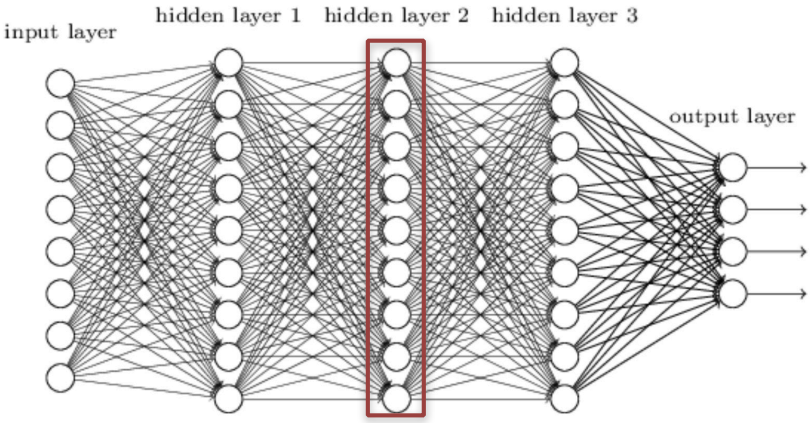
while you could probably figure this angle out using some high school trigonometry, this method works just as well for 100D vectors as for 2D vectors

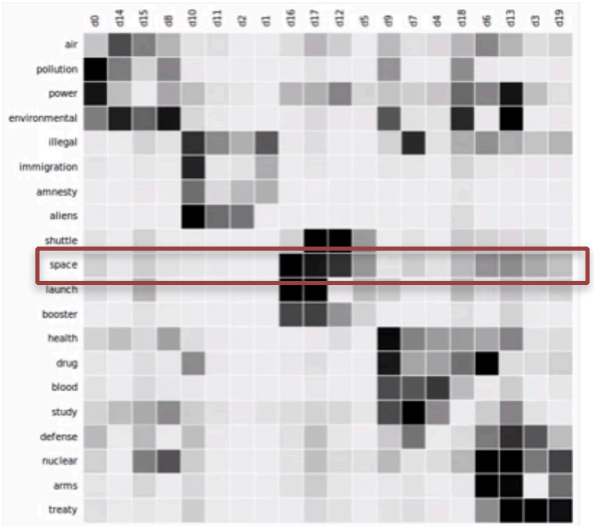
$$\theta = \text{acos}\left(\frac{\boldsymbol{a} \cdot \boldsymbol{b}}{\|\boldsymbol{a}\|\|\boldsymbol{b}\|}\right)$$

$$\theta = \text{acos}\left(\frac{\boldsymbol{a} \cdot \boldsymbol{b}}{\|\boldsymbol{a}\|\|\boldsymbol{b}\|}\right)$$

cosine angle is sometimes used to measure the similarity between two vector representations

- some number of psychological measures
- or psychological, social, and demographic measures
- or some number of neural measures

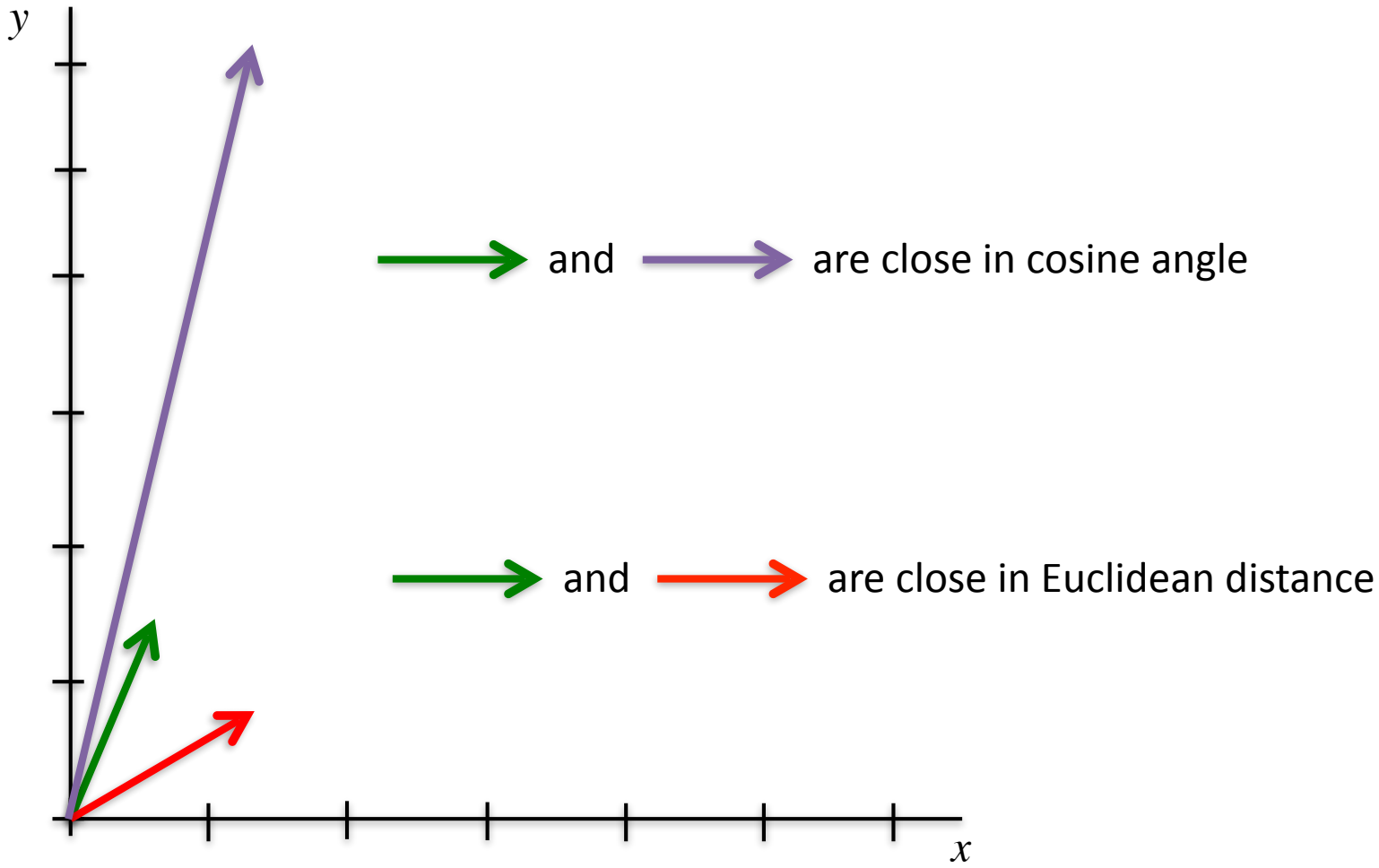




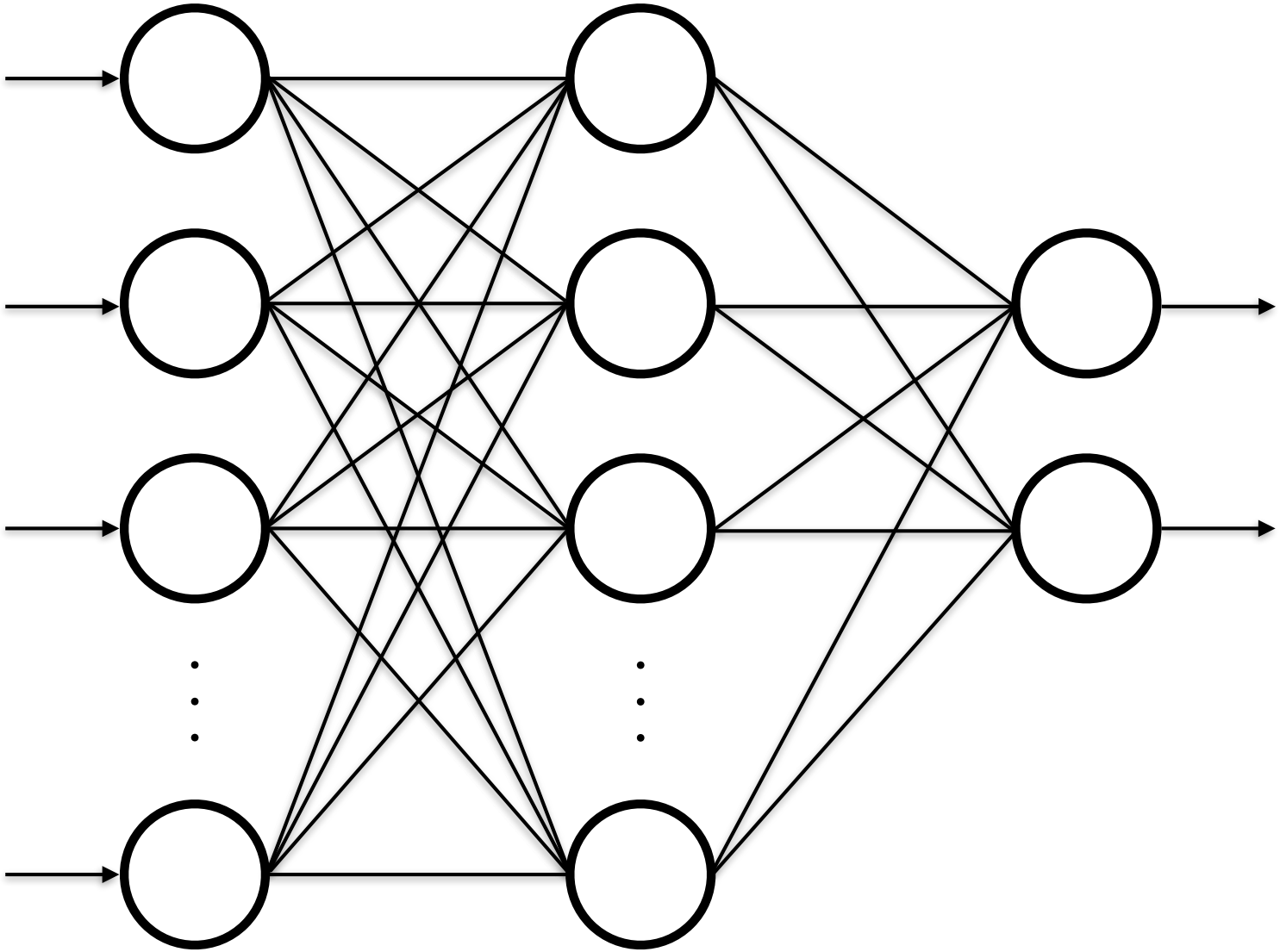
- activity of a units in a layer of a neural network model
- semantic representation of a word from latent semantic analysis

cosine angle vs. Euclidean distance

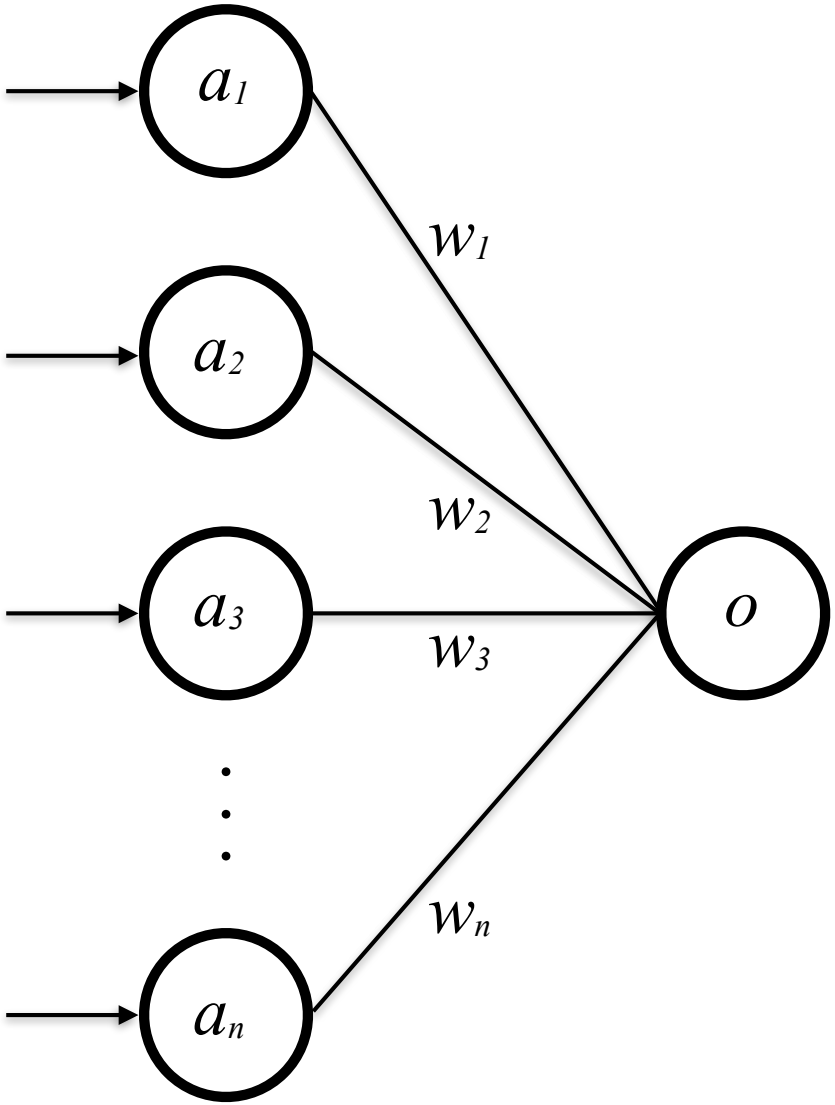
depends on what the vectors (and their distances) mean



math of a simple neural network model

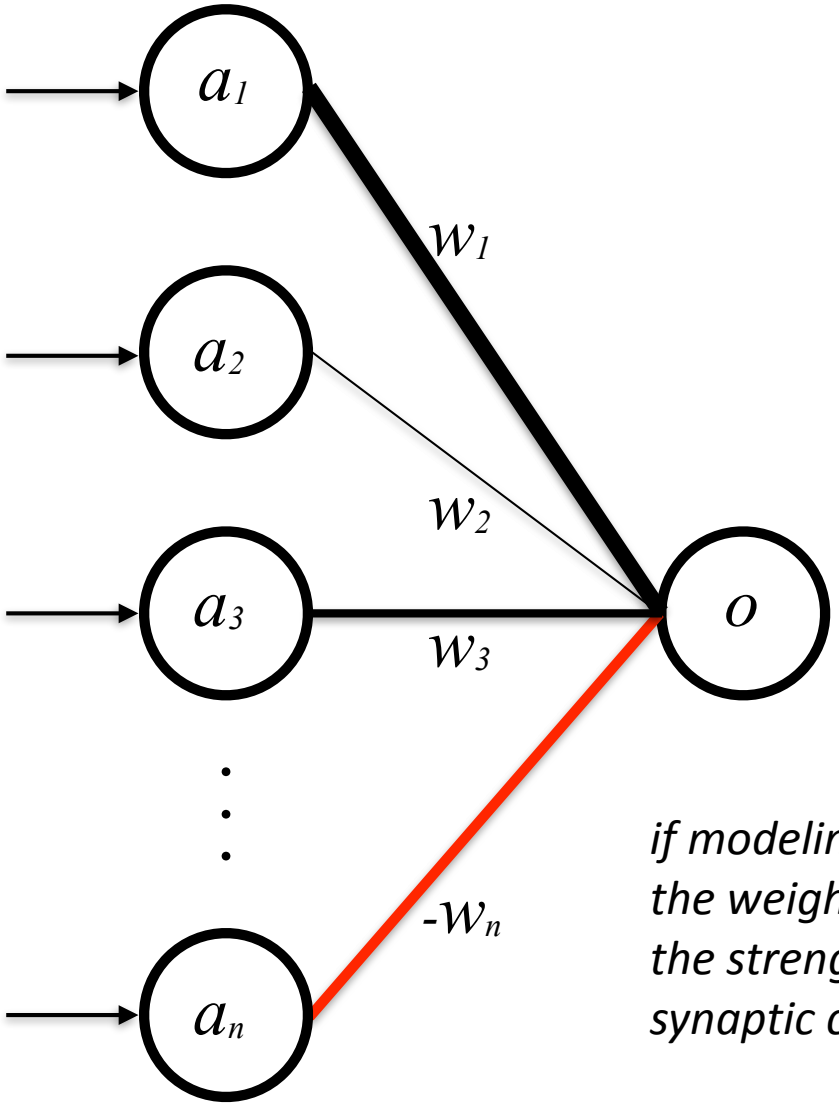


math of a simple neural network model



$$o = f\left(\sum_{i=1}^{n} a_i w_i\right)$$

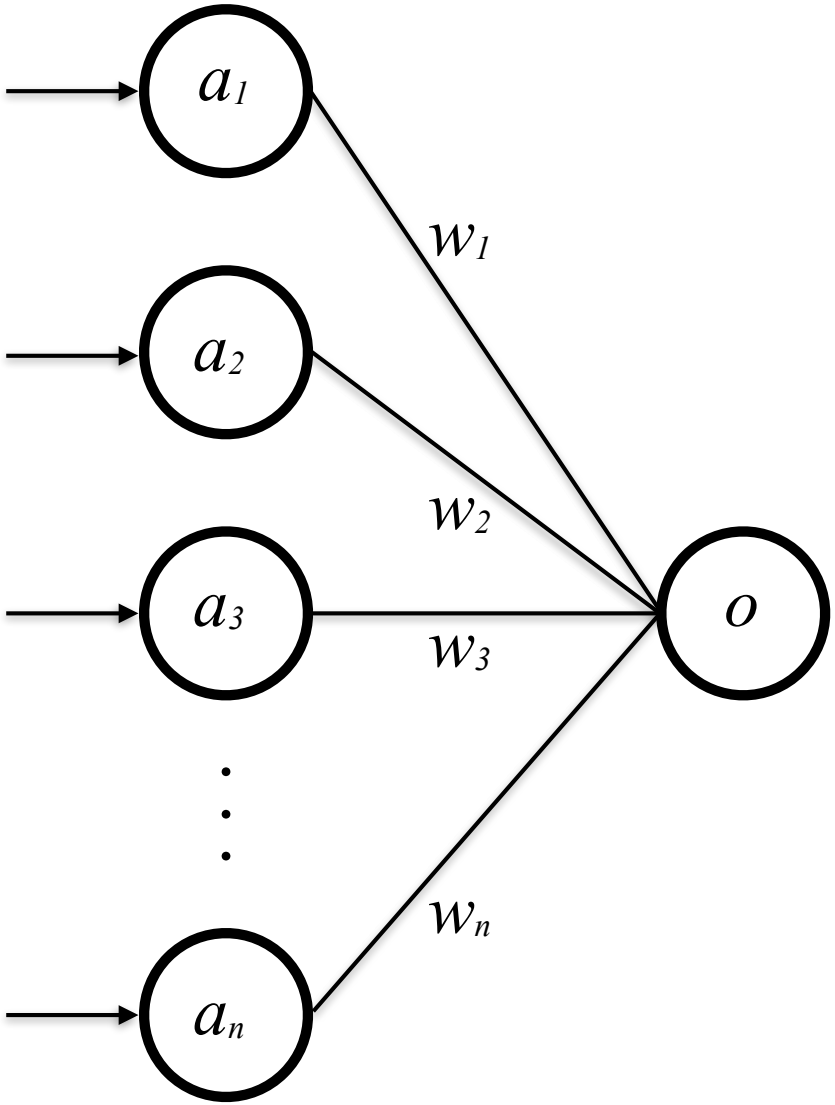
math of a simple neural network model



$$o = f\left(\sum_{i=1}^{n} a_i w_i\right)$$

if modeling actual neurons, the weights would reflect the strength (and type) of synaptic connections

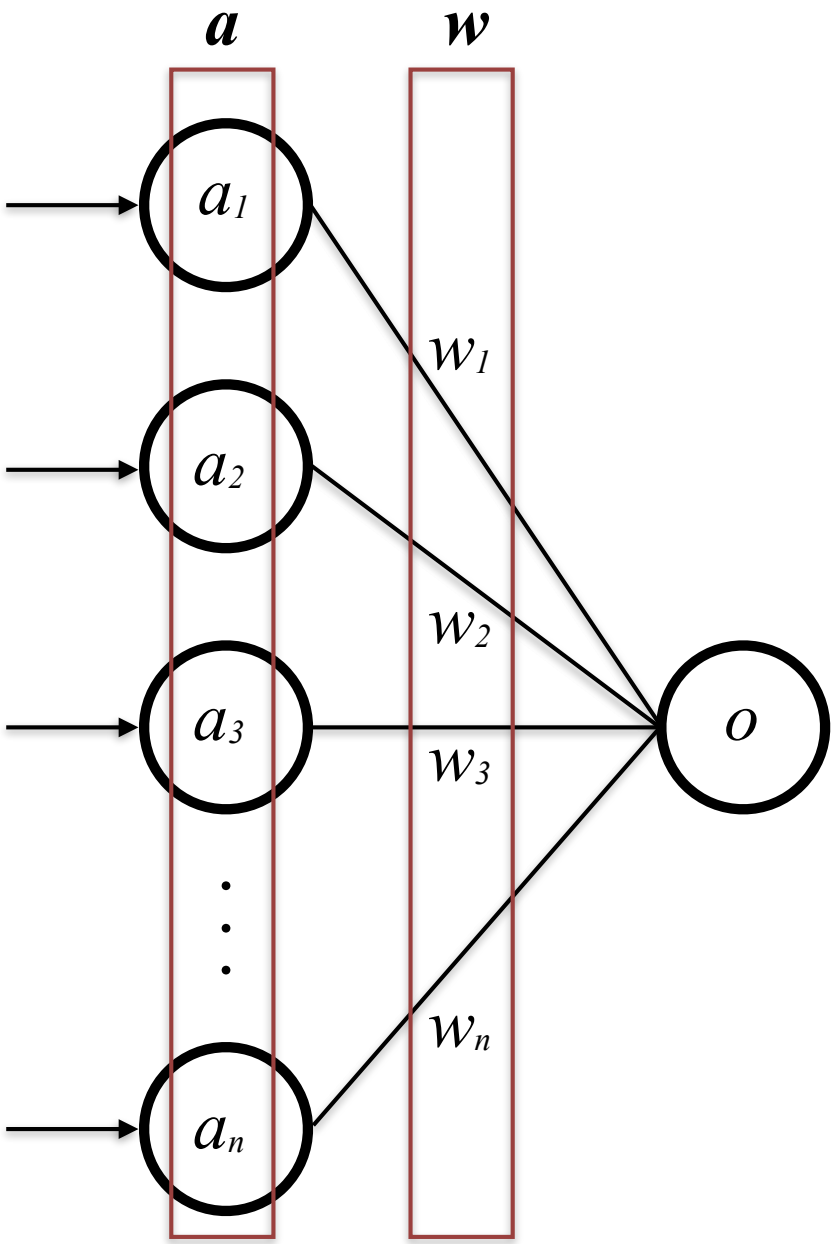
math of a simple neural network model



$$o = \sum_{i=1}^{n} a_i w_i$$

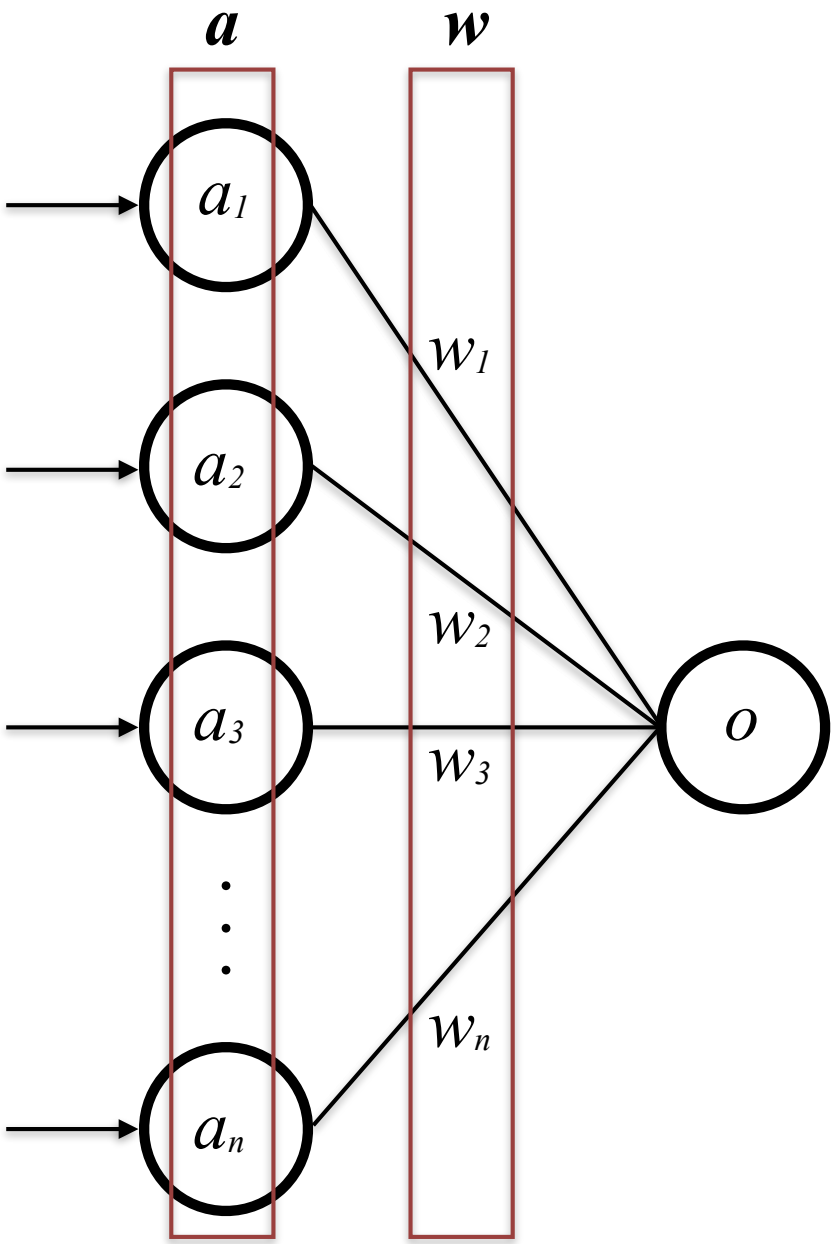
linear model (most interesting neural networks are nonlinear)

math of a simple neural network model



$$o = \sum_{i=1}^{n} a_i w_i = \boldsymbol{a} \cdot \boldsymbol{w}$$

math of a simple neural network model



$$o = \sum_{i=1}^{n} a_i w_i = \boldsymbol{a} \cdot \boldsymbol{w}$$

if the activations and weights were normalized, o would then be the cosine angle between $\boldsymbol{w}$ *and* $\boldsymbol{a}$

neural networks are pattern recognizers