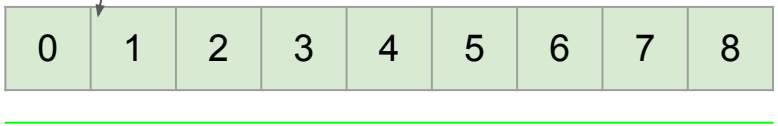
Similarities and optimization

Marcin Wierzbiński

recap: numpy array

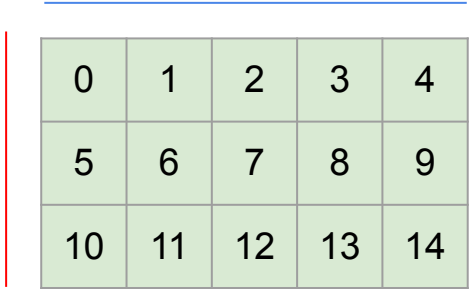
single element of data:



one dimensional array (vector)

shape: (9,)

```
arr.shape
```



two dimensional array (matrix)

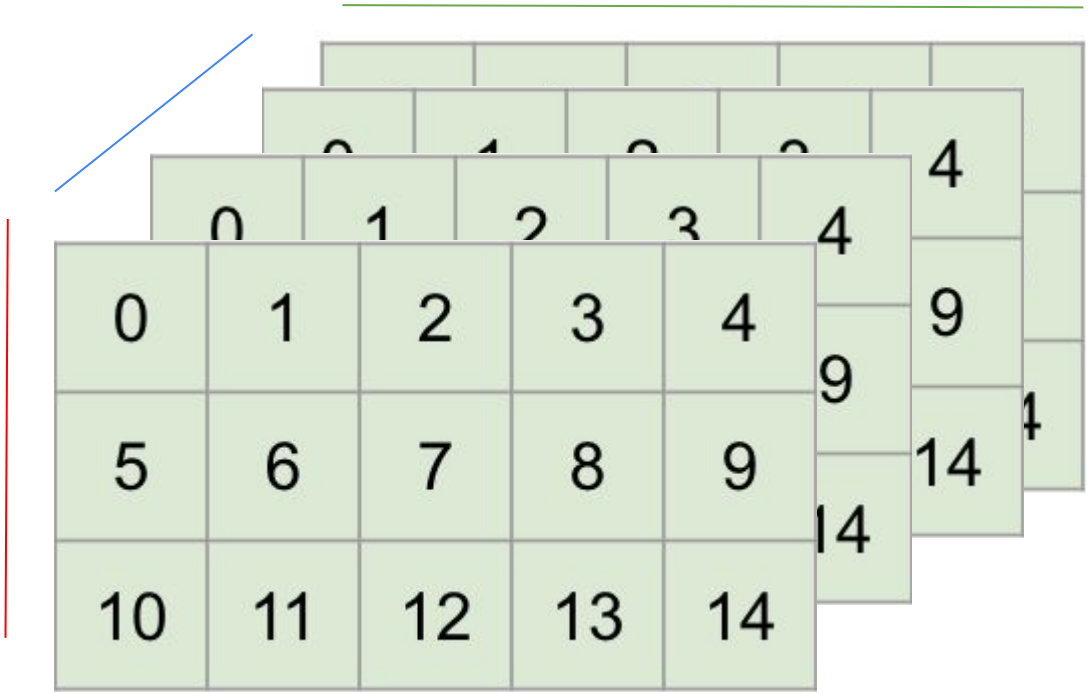
shape: (3,5)

```
arr.shape
```

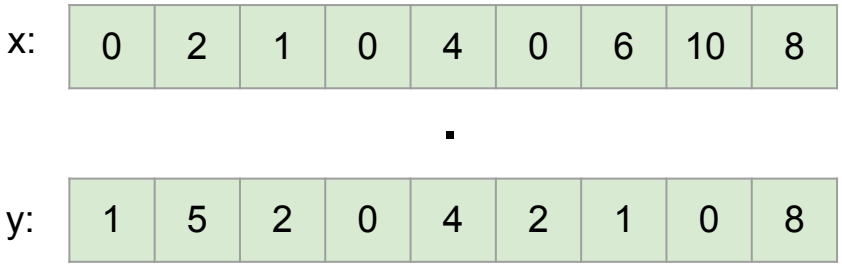
… and so on

3D array (tensor)

Shape: (3,5,4)



Dot product for vectors

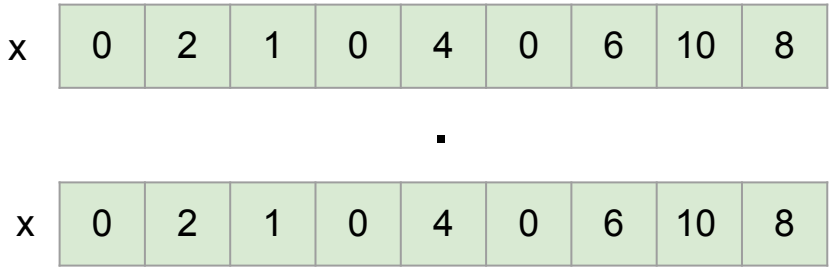


0*1 + 2*5 + 1*2 + 0*0 + 4*4 + 0*2 + 6*1 + 10*0 + 8 * 8 = 98

x . y

```
import numpy as np
x = np.array([0,2,1,0,4,0,6,10,8])
y = np.array([1,5,2,0,4,2,1,0, 8])
np.dot(x,y)
```

Length of vector

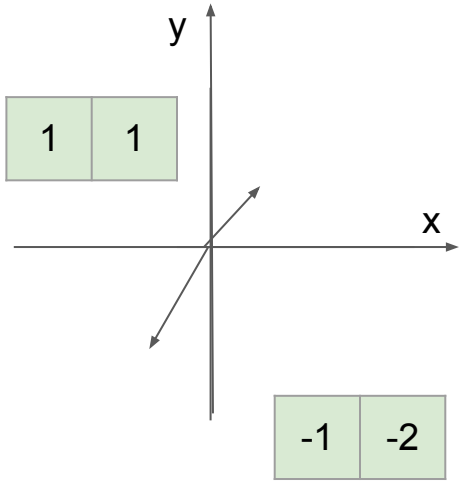


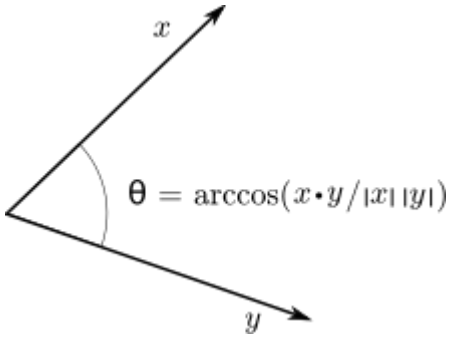

```
lenght_x = np.sqrt(np.dot(x,x))
```

0*0 + 2*2 + 1*1 + 0*0 + 4*4 + 0*0 + 6*6 + 10*10 + 8 * 8 = 221

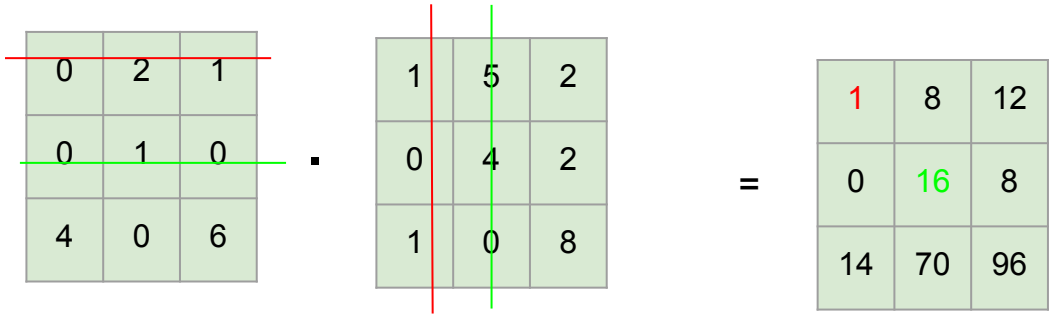
||x|| = sqrt(221) ~14.86

Angle between vectors





Dot product for matrices



Application: mean in rows

$$\frac{1}{3} * \begin{pmatrix} 0 & 2 & 1 \\ 0 & 1 & 0 \\ 4 & 0 & 6 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 3(\frac{1}{3}) & 3\frac{1}{3} & 3\frac{1}{3} \end{pmatrix}$$

```
1/3*np.array([0,2,1,0,1,0,4,0,6]).reshape(3,3).dot(np.ones((3,3)))
```

Transpose

0	2	1
0	1	0
4	0	6

0	0	4
2	1	0
1	0	6

reshape

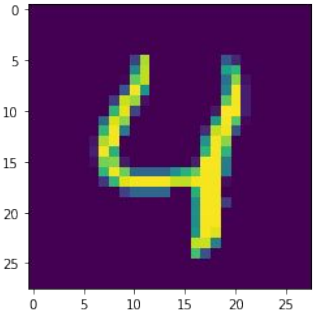

```
x1 = arr1.reshape(9)
```

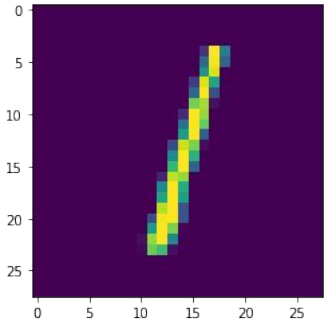
```
x = np.array([0,2,1,0,4,0,6,10,8])
```

0	2	1
0	1	0
4	0	6

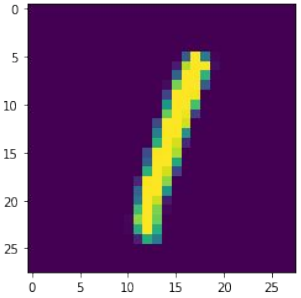
```
arr1 = x.reshape(3,3)
```

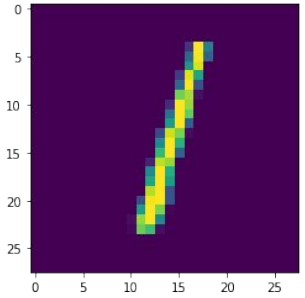
mnist





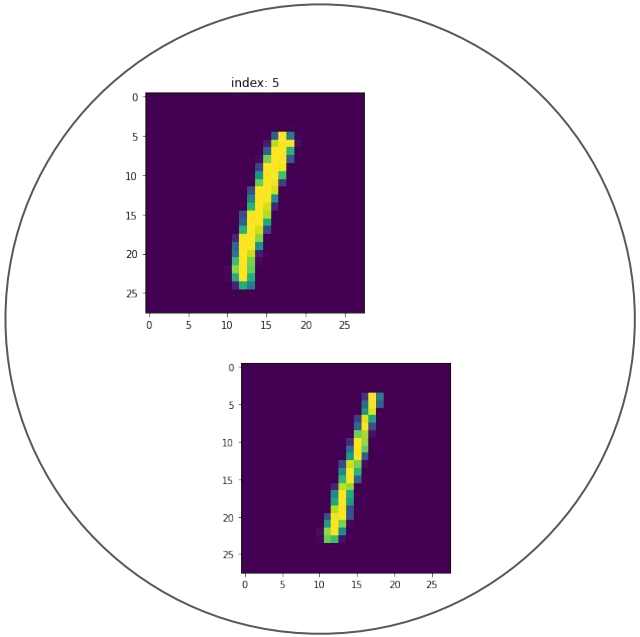
cos(theta) = 0.07613195268412658

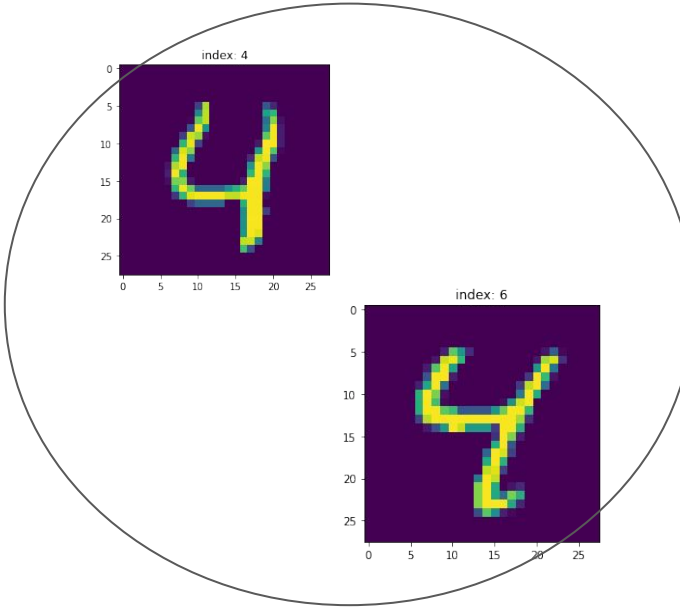




cos(theta) = 0.8958546444814048

Clustering

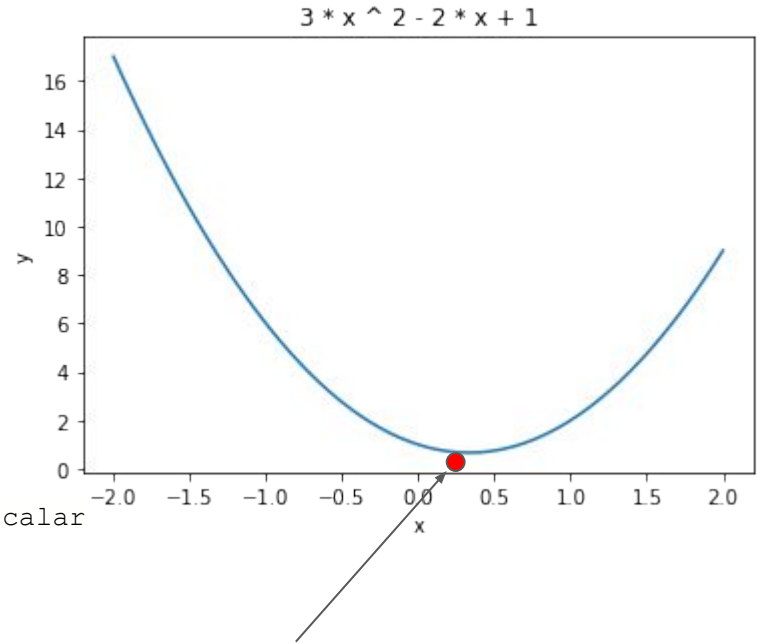




- Input - mnist vectors
- Algorithm: group elements by measure of similarity

Optimization

Problem:
Find minimum of f(x)

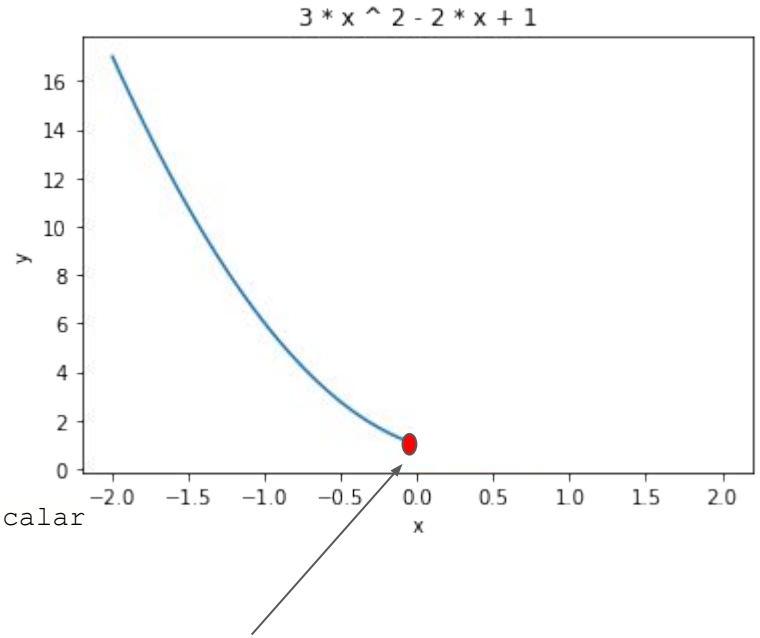



```
from scipy.optimize import minimize_scalar

def objective_function(x):
 return 3 * x ** 2 - 2 * x + 1

res = minimize_scalar(objective_function)
res.x
```

f(x) is objective function

Optimization

Problem:
Find minimum of f(x) in interval [-2,0]




```
from scipy.optimize import minimize_scalar

def objective_function(x):
 return 3 * x ** 2 - 2 * x + 1

res = minimize_scalar(objective_function, bounds=(-2, 0),
method='bounded')
res.x
```

f(x) is objective function

Practical problem

We want to maximize this objective function:

$f(x, y) = x \cdot y$

```
def objective_function(x, y):
    return -x.dot(y)


res = minimize(
    objective_function,
    x0=255 * np.random.random(784),
    args=(tensor_index_8,),
)
```