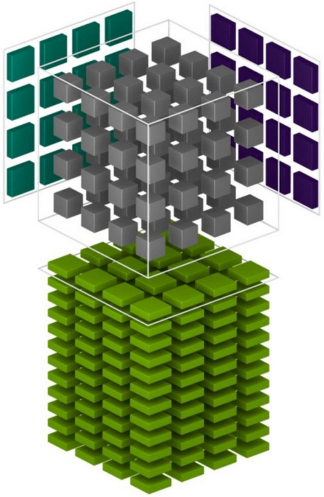
Module 0

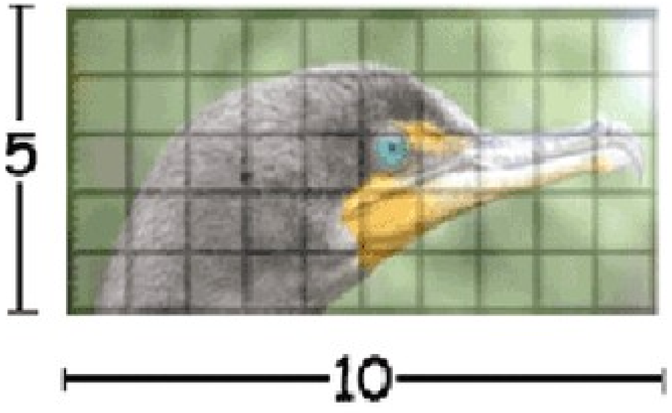
Matrix / Tensor Algebra

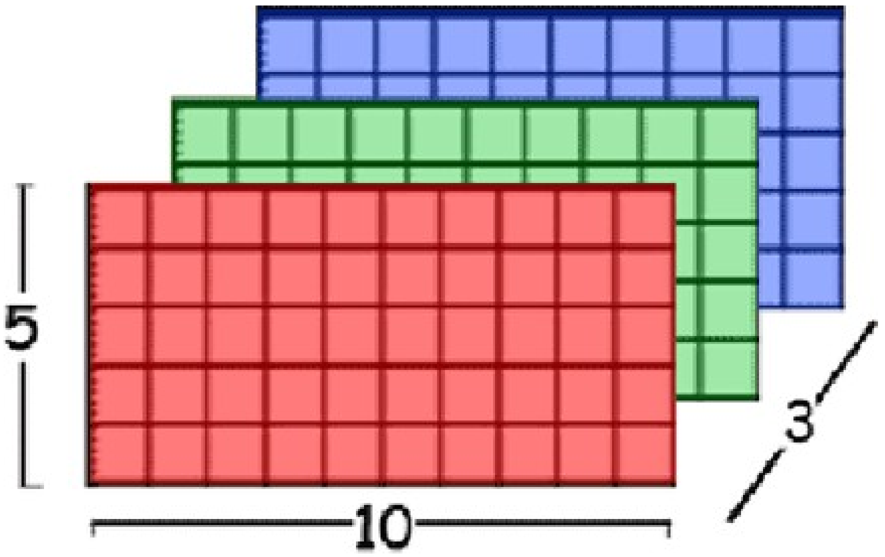
Matrices / Tensors



Data → Vectors, Matrices, Tensors

- Feature creation in ML involve transforming raw data into numerical representations (vectors, matrices, tensors)
- For example, Images

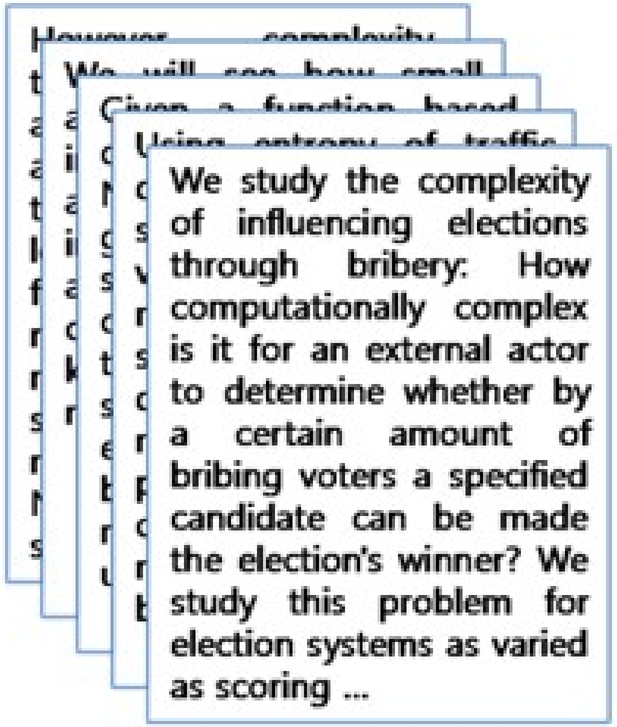




Data → Vectors, Matrices, Tensors

- Text, Documents

Documents



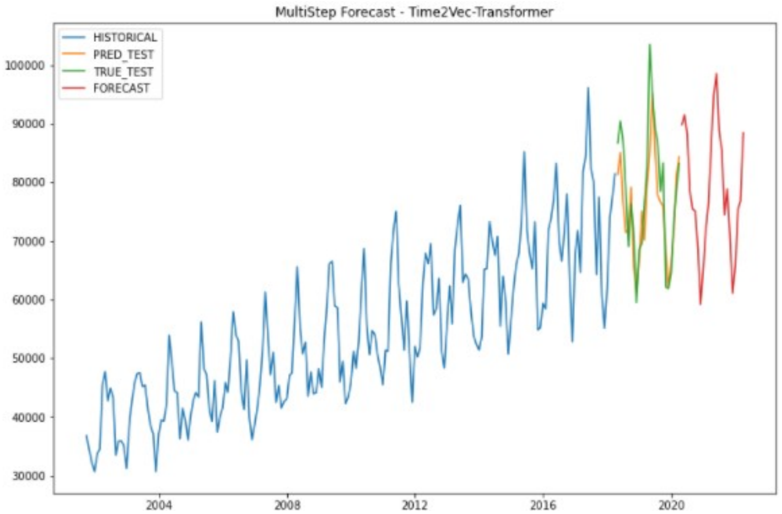
Vector-space representation

	D1	D2	D3	D4	D5
complexity	2		3	2	3
algorithm	3			4	4
entropy	1			2	
traffic		2	3		
network		1	4		

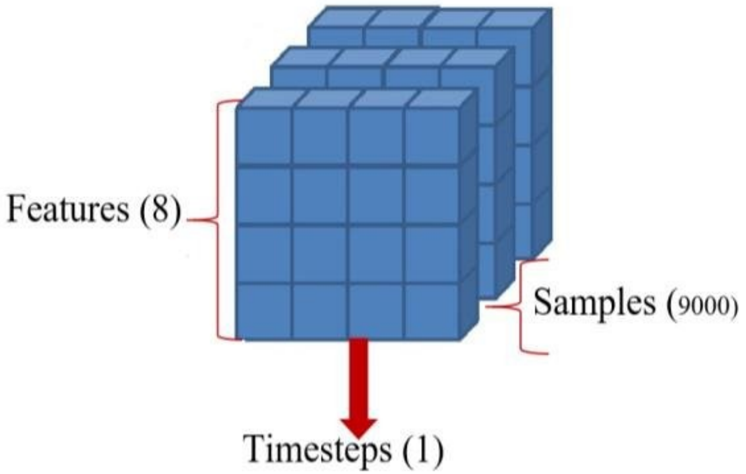
Term-document matrix

Data → Vectors, Matrices, Tensors

- Time Series



	features		target
timesteps	X_1	X_2	y
timestep 0	5	10	7
timestep 1	7	12	4
timestep 2	3	18	6
timestep 3	8	16	9
timestep 4	2	11	2
timestep 5	9	21	1



Data → Vectors, Matrices, Tensors

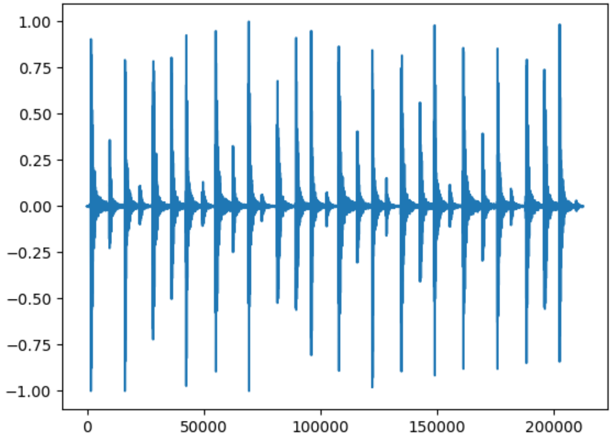
❑ Audio

```
import wave
import numpy as np
import matplotlib.pyplot as plt

with wave.open("Bongo_sound.wav") as wav_file:
    metadata = wav_file.getparams()
    frames = wav_file.readframes(metadata.nframes)

pcm_samples = np.frombuffer(frames, dtype="<h")
normalized_amplitudes = pcm_samples / (2 ** 15)

plt.plot(normalized_amplitudes);
```



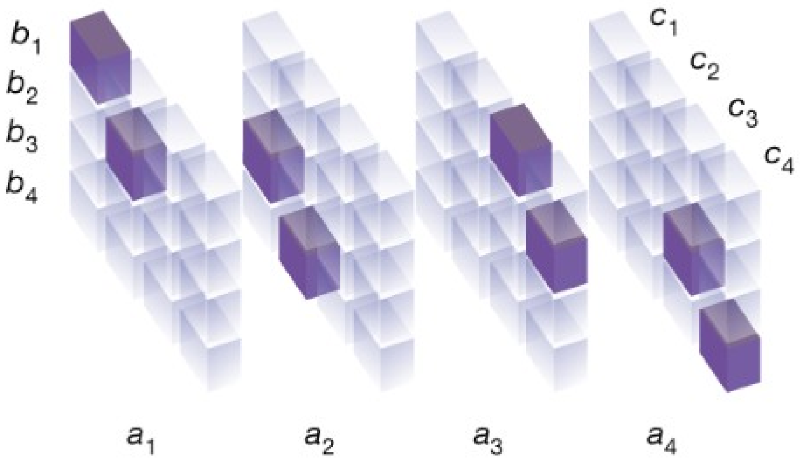

```
print(normalized_amplitudes)
```

```
[ 3.05175781e-05 -6.10351562e-05  6.10351562e-05 ... -5.49316406e-04
 -5.79833984e-04 -3.96728516e-04]
```

Data → Vectors, Matrices, Tensors

- Neural Net Weights

$$\begin{pmatrix} c_1 & c_2 \\ c_3 & c_4 \end{pmatrix} = \begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} \cdot \begin{pmatrix} b_1 & b_2 \\ b_3 & b_4 \end{pmatrix}$$



$$\begin{aligned}
m_1 &= (a_1 + a_4)(b_1 + b_4) \\
m_2 &= (a_3 + a_4)\, b_1 \\
m_3 &= a_1\,(b_2 - b_4) \\
m_4 &= a_4\,(b_3 - b_1) \\
m_5 &= (a_1 + a_2)\, b_4 \\
m_6 &= (a_3 - a_1)(b_1 + b_2) \\
m_7 &= (a_2 - a_4)(b_3 + b_4) \\
c_1 &= m_1 + m_4 - m_5 + m_7 \\
c_2 &= m_3 + m_5 \\
c_3 &= m_2 + m_4 \\
c_4 &= m_1 - m_2 + m_3 + m_6
\end{aligned}$$

$$\mathbf{U} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & -1 \end{pmatrix}$$

$$\mathbf{V} = \begin{pmatrix} 1 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

$$\mathbf{W} = \begin{pmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

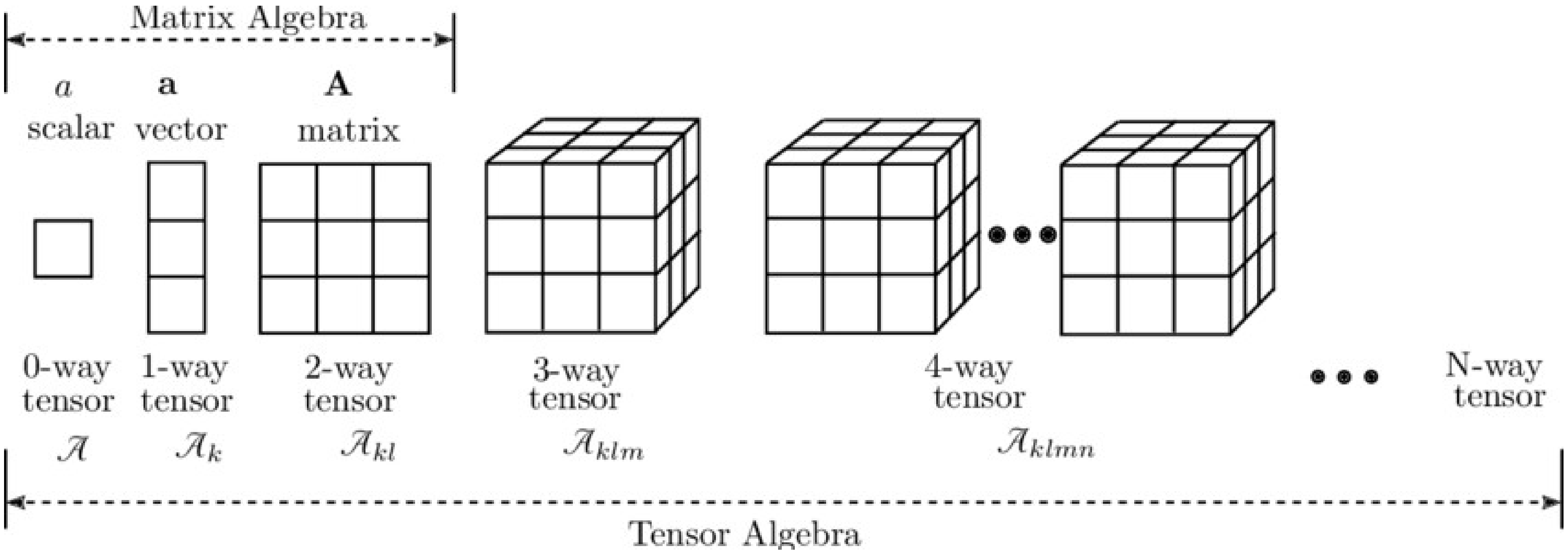
NumPy vs Tensorflow

- **NumPy** provides support for large multidimensional arrays (vectors), matrices and tensors.

- **Tensorflow** also provides support for large multidimensional arrays (vectors), matrices and tensors. Has the ability to deploy numerical manipulations to run on CPUs, GPUs.

Linear Algebra

- study of vectors / multidimensional arrays and certain algebra rules to manipulate them.



Vectors

- A vector is an object that has both a magnitude and a direction. In mathematics and physics, vector is a term that refers to some quantities that cannot be expressed by a single number (a scalar).

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_{n-1} \\ v_n \end{bmatrix}$$

Vectors

- A vector is an object that has both a magnitude and a direction. In mathematics and physics, vector is a term that refers to some quantities that cannot be expressed by a single number (a scalar).

NumPy

```
# NumPy
import numpy as np

v1 = np.array([1,2,3,4])
```

```
v1.shape
```

Vectors

- A vector is an object that has both a magnitude and a direction. In mathematics and physics, vector is a term that refers to some quantities that cannot be expressed by a single number (a scalar)

🎯 **Tensorflow**

```
# Tensorflow
import tensorflow as tf

v1 = tf.constant([1,2,3,4])
```

```
# this is a 1 dimension tensor
v1.ndim
```

```
1
```

```
v1.shape
```

```
TensorShape([4])
```

Python

Vector Operations (+,-, scalar mult)

- Adding vectors means adding the individual elements.
- Multiplying vectors by scalars means multiplying each element by scalar

$$\begin{pmatrix} 3 \\ 5 \\ -2 \end{pmatrix} + \begin{pmatrix} 1 \\ 4 \\ 6 \end{pmatrix} = \begin{pmatrix} 4 \\ 9 \\ 4 \end{pmatrix}$$

$$k\mathbf{v} = 4 \begin{pmatrix} 2 \\ -3 \\ 5 \end{pmatrix} = \begin{pmatrix} 4 \cdot 2 \\ 4 \cdot (-3) \\ 4 \cdot 5 \end{pmatrix} = \begin{pmatrix} 8 \\ -12 \\ 20 \end{pmatrix}$$

```python
# vector addition
v2 = np.array([2,3,4,5])

v1 + v2
```

```
array([3, 5, 7, 9])
```

```python
# scalar multiplication
2*v1
```

```
array([2, 4, 6, 8])
```

Vectors

- Adding vectors means adding the individual elements.
- Multiplying vectors by scalars means multiplying each element by scalar

```
# vector addition
v2 = np.array([2,3,4,5])

v1 + v2
```

```
array([3, 5, 7, 9])
```

```
# scalar multiplication
2*v1
```

```
array([2, 4, 6, 8])
```

Vectors

- Adding vectors means adding the individual elements.
- Multiplying vectors by scalars means multiplying each element by scalar

```python
# vector addition
v2 = tf.constant([2,3,4,5])

v1 + v2
```

```
<tf.Tensor: shape=(4,), dtype=int32, numpy=array([3, 5, 7, 9], dtype=int32)>
```

```python
# scalar multiplication
2*v1
```

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_{n-1} \\ v_n \end{bmatrix}$$

Python

Vectors Transpose

- The transpose of a vector changes a column vector to a row vector or vice versa.

$$v = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_k \end{bmatrix} \qquad v^\top = \begin{bmatrix} v_1 & v_2 & \cdots & v_k \end{bmatrix}$$

Vectors

- The transpose of a vector changes a column vector to a row vector or vice versa.

```python
import numpy as np

## let's create a row vector
v = np.array([[2,3,4]])

print(f'The vector is: {v}')
print(f'The transpose is: \n {v.T}')
```

```
The vector is: [[2 3 4]]
The transpose is:
 [[2]
 [3]
 [4]]
```

$$v = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_k \end{bmatrix}$$

$$v^{\mathsf{T}} = \begin{bmatrix} v_1 & v_2 & \dots & v_k \end{bmatrix}$$

Vectors

- The transpose of a vector changes a column vector to a row vector or vice versa.

```
import tensorflow as tf

## let's create a row vector
v = tf.constant([[2,3,4]])

print(f'The vector is: {v}')
print(f'The transpose is: \n {tf.transpose(v)}')
```

```
The vector is: [[2 3 4]]
The transpose is:
 [[2]
 [3]
 [4]]
```

$$v = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_k \end{bmatrix}$$

$$v^{\mathsf{T}} = \begin{bmatrix} v_1 & v_2 & \dots & v_k \end{bmatrix}$$

Python

The Dot Product

- Very important in ML
- is the sum of the products of corresponding components.
- Can be expressed in terms of the transpose: $\boldsymbol{x}^\top \boldsymbol{y}$

$$\mathbf{u} = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix}, \quad \mathbf{v} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix}$$

$$\begin{bmatrix} 2 & 3 & 4 \end{bmatrix} \times \begin{bmatrix} 6 \\ 4 \\ 3 \end{bmatrix} = 36$$

$$\mathbf{u} \cdot \mathbf{v} = u_1 v_1 + u_2 v_2 + \cdots + u_n v_n = \sum_{i=1}^{n} u_i v_i$$

The Dot Product

- Very important in ML
- is the sum of the products of corresponding components.
- Can be expressed in terms of the transpose: $\boldsymbol{x}^\top \boldsymbol{y}$

$$\begin{bmatrix} 2 & 3 & 4 \end{bmatrix} \times \begin{bmatrix} 6 \\ 4 \\ 3 \end{bmatrix} = 36$$

```python
# dot product
v1 = np.array([2,3,4])
v2 = np.array([6,4,3])
v1.dot(v2)
```

```
36
```

The Dot Product

- Very important in ML
- is the sum of the products of corresponding components.

$$\begin{bmatrix} 2 & 3 & 4 \end{bmatrix} \times \begin{bmatrix} 6 \\ 4 \\ 3 \end{bmatrix} = 36$$

```
# dot product
v1 = tf.constant([2,3,4])
v2 = tf.constant([6,4,3])

tf.tensordot(v1,v2, 1)
```

```
<tf.Tensor: shape=(), dtype=int32, numpy=36>
```

Python

Vector Norms

- Used in ML to regularize (drop insignificant relationships) weights.

$$w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_k \end{bmatrix}$$

Goal is to zero out or diminish weights for insignificant variables

Vector Norms

- Used in ML to regularize (drop insignificant relationships) weights.

$$w = \begin{bmatrix} 0.44 \\ 2.83 \\ \boxed{-0.01} \end{bmatrix}$$

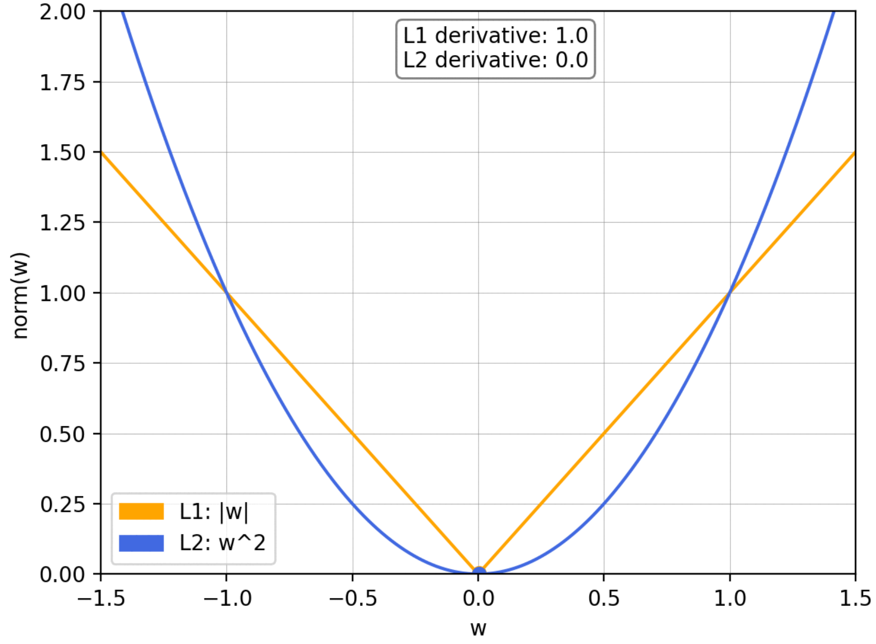
We want this weight to be 0 or as close to zero as possible

$$x_1 = \text{sq. feet}$$
$$x_2 = \#\,\text{rooms}$$
$$x_3 = \#\,\text{bulbs}$$

$$y = \text{Home Price}$$

Vector Norms

- Are added to cost functions as penalties (e.g., SSE, MSE)
- The L_1 norm (tends to zero out the unimportant weights).
- The L_2 norm (tends to to get the weights to be close to zero).



$$Cost = \mathrm{SSE} + \|\boldsymbol{w}\|_2$$

$$Cost = \mathrm{SSE} + \|\boldsymbol{w}\|_1$$

L_2 Norm

- Tends to make weights close to zero
- For a weight vector:

$$\boldsymbol{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_k \end{bmatrix}$$

- The L2 norm is defined as:

$$\|\boldsymbol{w}\|_2 = \sqrt{\sum_{i=1}^{k} w_i^2} \qquad \|\boldsymbol{w}\|_2 = \sqrt{\boldsymbol{w}^\top \boldsymbol{w}}$$

L_2 Norm

$$\boldsymbol{w} = \begin{bmatrix} 0.44 \\ 2.83 \\ -0.01 \end{bmatrix} \quad \|\boldsymbol{w}\|_2 = \sqrt{\boldsymbol{w}^\top \boldsymbol{w}}$$

```python
# NumPy
import numpy as np

w = np.array([0.44, 2.83, -0.01])
np.linalg.norm(w, ord = 2)
```

```
2.8640181563670297
```

L_2 Norm

$$\boldsymbol{w} = \begin{bmatrix} 0.44 \\ 2.83 \\ -0.01 \end{bmatrix} \quad \|\boldsymbol{w}\|_2 = \sqrt{\boldsymbol{w}^\top \boldsymbol{w}}$$

```
[3] # Tensorflow
    import tensorflow as tf

    w = tf.constant([0.44, 2.83, -0.01])
    tf.norm(w, ord = 2)

    <tf.Tensor: shape=(), dtype=float32, numpy=2.864018>
```

L_1 Norm

- Tends to make weights zero
- For a weight vector:

$$\boldsymbol{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_k \end{bmatrix}$$

- The L1 norm is defined as:

$$\|\boldsymbol{w}\|_1 = \sum_{i=1}^{k} |w_i|$$

L_1 Norm

$$\boldsymbol{w} = \begin{bmatrix} 0.44 \\ 2.83 \\ -0.01 \end{bmatrix} \qquad \|\boldsymbol{w}\|_1 = \sum_{i=1}^{k} |w_i|$$

```
[4] # NumPy
    import numpy as np

    w = np.array([0.44, 2.83, -0.01])
    np.linalg.norm(w, ord = 1)
```

```
3.28
```

L_1 Norm

$$\boldsymbol{w} = \begin{bmatrix} 0.44 \\ 2.83 \\ -0.01 \end{bmatrix} \qquad \|\boldsymbol{w}\|_1 = \sum_{i=1}^{k} |w_i|$$

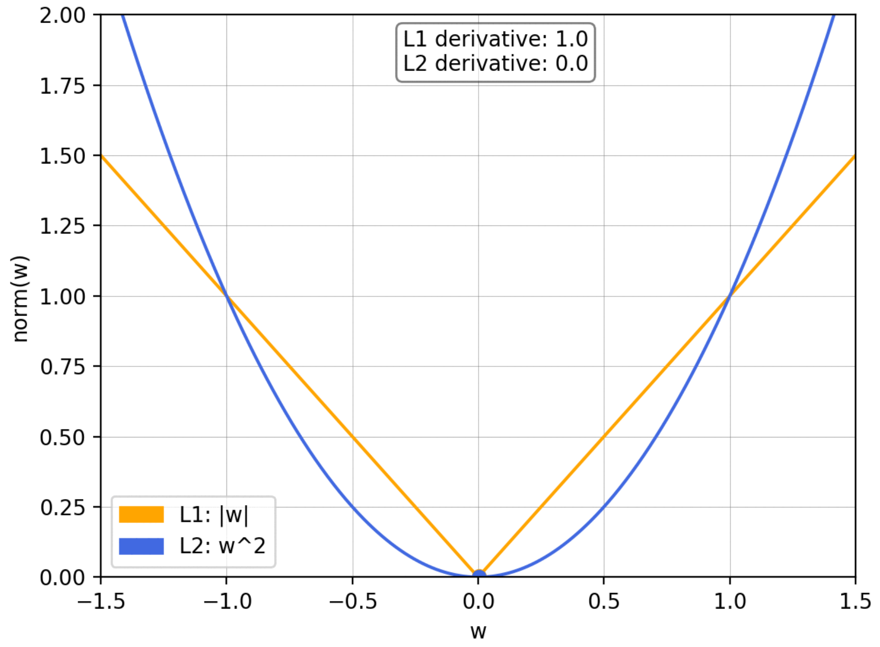
```
[5] # Tensorflow
    import tensorflow as tf

    w = tf.constant([0.44, 2.83, -0.01])
    tf.norm(w, ord = 1)

<tf.Tensor: shape=(), dtype=float32, numpy=3.28>
```

Vector Norms

- Are added to cost functions as penalties (e.g., SSE, MSE)
- The L_1 norm (tends to zero out the unimportant weights).
- The L_2 norm (tends to to get the weights to be close to zero).



$$Cost = \text{SSE} + \lambda \|\boldsymbol{w}\|_2$$

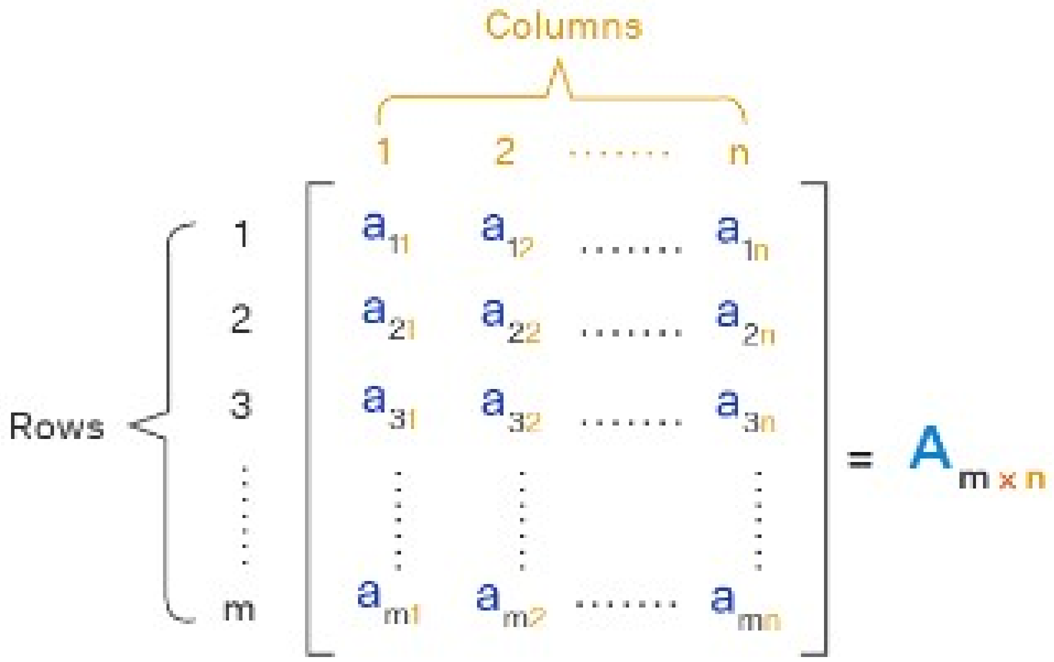
$$Cost = \text{SSE} + \lambda \|\boldsymbol{w}\|_1$$

λ is used to control amount of penalty

Python

Matrices

- A matrix is a rectangular array of numbers (called elements) arranged into rows and columns.



Matrices

- A matrix is a rectangular array of numbers (called elements) arranged into rows and columns.

NumPy

```
M1 = np.array([[2, -5, -11, 0],
               [-9, 4, 6, 13],
               [4, 5, 12, -2]])
```

```
M1.shape
```

```
(3, 4)
```

$$\begin{array}{cc} \text{4 columns} & \\ \left[\begin{array}{cccc} 2 & -5 & -11 & 0 \\ -9 & 4 & 6 & 13 \\ 4 & 7 & 12 & -2 \end{array}\right] & \text{3 rows} \end{array}$$

Matrices

- A matrix is a rectangular array of numbers (called elements) arranged into rows and columns.

🎯 **Tensorflow**

```python
M1 = tf.constant([[2, -5, -11, 0],
                  [-9, 4, 6, 13],
                  [4, 5, 12, -2]])
```

```python
## number of dimensions of the tensor
M1.ndim
```

```
2
```

$$\begin{array}{cc} \text{4 columns} & \\ \begin{bmatrix} 2 & -5 & -11 & 0 \\ -9 & 4 & 6 & 13 \\ 4 & 7 & 12 & -2 \end{bmatrix} & \text{3 rows} \end{array}$$

Matrix Addition / Subtraction

- You can add two matrices, if they have the same dimensions.

$$\overset{A}{\begin{bmatrix} 4 & 8 \\ 3 & 7 \end{bmatrix}} + \overset{B}{\begin{bmatrix} 1 & 0 \\ 5 & 2 \end{bmatrix}} = \begin{bmatrix} 4+1 & 8+0 \\ 3+5 & 7+2 \end{bmatrix}$$

Matrix Addition / Subtraction

- You can add two matrices if they have the same dimensions

$$\overset{A}{\begin{bmatrix} 4 & 8 \\ 3 & 7 \end{bmatrix}} + \overset{B}{\begin{bmatrix} 1 & 0 \\ 5 & 2 \end{bmatrix}} = \begin{bmatrix} 4+1 & 8+0 \\ 3+5 & 7+2 \end{bmatrix}$$

NumPy

```
A = np.array([[4,8],[3,7]])
B = np.array([[1,0],[5,2]])
A+B
```

```
array([[5, 8],
       [8, 9]])
```

Matrix Addition / Subtraction

- You can add two matrices ifthey have the same dimensions

$$\overset{A}{\begin{bmatrix} 4 & 8 \\ 3 & 7 \end{bmatrix}} + \overset{B}{\begin{bmatrix} 1 & 0 \\ 5 & 2 \end{bmatrix}} = \begin{bmatrix} 4+1 & 8+0 \\ 3+5 & 7+2 \end{bmatrix}$$

Tensorflow

```
A = tf.constant([[4,8],[3,7]])
B = tf.constant([[1,0],[5,2]])
A+B
```

```
<tf.Tensor: shape=(2, 2), dtype=int32, numpy=
array([[5, 8],
       [8, 9]], dtype=int32)>
```

Python

Matrix Multiplication

- A matrix can be multiplied by any other matrix that has the same number of rows as the first has columns (conformable).
- In other words, Inner Dimensions must match

- $\mathbf{A}_{m1\times n1} * \mathbf{B}_{m2\times n2}$ can be calculated as long as $n1 = m2$
- $\mathbf{A}_{2\times 3} * \mathbf{B}_{3\times 2}$ can be calculated
- $\mathbf{A}_{3\times 3} * \mathbf{B}_{3\times 2}$ can be calculated
- $\mathbf{A}_{3\times 2} * \mathbf{B}_{4\times 2}$ cannot be calculated
- $\mathbf{A}_{3\times 2} * \mathbf{B}_{2\times 7}$ can be calculated
- $\mathbf{A}_{1\times 2} * \mathbf{B}_{2\times 12}$ can be calculated

Matrix Multiplication

- A matrix can be multiplied by any other matrix that has the same number of rows as the first has columns (conformable).
- In other words, Inner Dimensions must match
- Resulting matrix have outer dimensions

- $\mathbf{A}_{2\times3} * \mathbf{B}_{3\times2} = \mathbf{C}_{2\times2}$
- $\mathbf{A}_{3\times3} * \mathbf{B}_{3\times2} = \mathbf{C}_{3\times2}$
- $\mathbf{A}_{3\times2} * \mathbf{B}_{2\times7} = \mathbf{C}_{3\times7}$
- $\mathbf{A}_{1\times2} * \mathbf{B}_{2\times12} = \mathbf{C}_{1\times12}$
- $\mathbf{A}_{m1\times n1} * \mathbf{B}_{n1\times n2} = \mathbf{C}_{m1\times n2}$

Matrix Multiplication

- A matrix can be multiplied by any other matrix that has the same number of rows as the first has columns (conformable).
- In other words, Inner Dimensions must match
- Resulting matrix have outer dimensions

$$A_{2\times 2} = \begin{bmatrix} 2 & 1 \\ 4 & 5 \end{bmatrix} \quad B_{2\times 2} = \begin{bmatrix} 3 & 6 \\ 7 & 9 \end{bmatrix}$$

$$A * B = C_{2\times 2} = \begin{bmatrix} 2*3+1*7=13 & 2*6+1*9=21 \\ 4*3+5*7=47 & 4*6+5*9=69 \end{bmatrix}$$

Matrix Multiplication

$$A_{2\times 2} = \begin{bmatrix} 2 & 1 \\ 4 & 5 \end{bmatrix} \quad B_{2\times 2} = \begin{bmatrix} 3 & 6 \\ 7 & 9 \end{bmatrix}$$

🎯 **NumPy**

```
import numpy as np

A = np.array([[2,1],[4,5]])
B = np.array([[3,6],[7,9]])

A.dot(B)
```

```
array([[13, 21],
       [47, 69]])
```

```
B.dot(A)
```

```
array([[30, 33],
       [50, 52]])
```

Matrix Multiplication

$$A_{2\times 2} = \begin{bmatrix} 2 & 1 \\ 4 & 5 \end{bmatrix} \quad B_{2\times 2} = \begin{bmatrix} 3 & 6 \\ 7 & 9 \end{bmatrix}$$

Tensorflow

```
import tensorflow as tf

A = tf.constant([[2,1],[4,5]])
B = tf.constant([[3,6],[7,9]])

tf.matmul(A,B)
```

```
<tf.Tensor: shape=(2, 2), dtype=int32, numpy=
array([[13, 21],
       [47, 69]], dtype=int32)>
```

```
tf.matmul(B,A)
```

```
<tf.Tensor: shape=(2, 2), dtype=int32, numpy=
array([[30, 33],
       [50, 52]], dtype=int32)>
```

Python

Matrix Transpose

- Is a matrix that is obtained by interchanging the rows and columns of the matrix

$$\begin{bmatrix} 6 & 4 & 24 \\ 1 & -9 & 8 \end{bmatrix}^T = \begin{bmatrix} 6 & 1 \\ 4 & -9 \\ 24 & 8 \end{bmatrix}$$

Matrix Transpose

🎯 **NumPy**

```
import numpy as np

A = np.array([[2,1],[4,5]])
B = np.array([[3,6],[7,9]])
print('-----------------')
print('Original Matrix A')
print(A)
print('-----------------')
print("Matrix A'")
print(A.transpose())
```

```
-----------------
Original Matrix A
[[2 1]
 [4 5]]
-----------------
Matrix A'
[[2 4]
 [1 5]]
```

Matrix Transpose

```
import tensorflow as tf

A = tf.constant([[2,1],[4,5]])
B = tf.constant([[3,6],[7,9]])
print('-----------------')
print('Original Matrix A')
print(A)
print('-----------------')
print("Matrix A'")
print(tf.transpose(A))
```

```
-----------------
Original Matrix A
tf.Tensor(
[[2 1]
 [4 5]], shape=(2, 2), dtype=int32)
-----------------
Matrix A'
tf.Tensor(
[[2 4]
 [1 5]], shape=(2, 2), dtype=int32)
```

Python

Identity Matrix

- Is a matrix of order $n \times n$ such that each main diagonal element is equal to 1

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

Identity Matrix

- Matrix of order $n \times n$ such that each main diagonal element is equal to 1

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

🎯 **NumPy**

```python
# you can create a square identity matrix with the np.eye() function
I = np.eye(3)
```

Identity Matrix

- Matrix of order $n \times n$ such that each main diagonal element is equal to 1

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

🎯 **Tensorflow**

```python
# you can create a square identity matrix with the tf.eye() function
I = tf.eye(3)
I
```

```
<tf.Tensor: shape=(3, 3), dtype=float32, numpy=
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]], dtype=float32)>
```

Python

Linear Dependence

- A matrix is linearly dependent if at least one row (or one column) is a linear combination of the others

$$A = \begin{bmatrix} 1 & 4 & 3 \\ 2 & 4 & 6 \\ 3 & 1 & 9 \end{bmatrix}$$

Linear Dependence

- A matrix is linearly dependent if at least one row (or one column) is a linear combination of the others

$$A = \begin{bmatrix} 1 & 4 & 3 \\ 2 & 4 & 6 \\ 3 & 1 & 9 \end{bmatrix} \qquad 2 \times \begin{matrix} 1 \\ 2 \\ 3 \end{matrix}$$

Linear Dependence

- The rank of a matrix is the number of columns that are independent

$$A = \begin{bmatrix} \boxed{1} & 4 & \boxed{3} \\ \boxed{2} & 4 & \boxed{6} \\ \boxed{3} & 1 & \boxed{9} \end{bmatrix} \qquad 2 \times \begin{matrix} 1 \\ 2 \\ 3 \end{matrix}$$

Linear Dependence

🎯 **NumPy**

```
## check the rank
## if rank < than # of cols or rows then A is dependent
## note matrix needs to be square

A = np.array([[1,4,6],
              [2,4,8],
              [3,1,7]])
np.linalg.matrix_rank(A) ## dependent
```

2

```
A = np.array([[2,3],
              [4,7]])
np.linalg.matrix_rank(A) ## independent
```

2

Linear Dependence

Tensorflow

```
## check the rank
## if rank < than # of cols or rows then A is dependent
## note matrix needs to be square
## for tensorflow matrix needs to be float type

A = tf.constant([[1,4,6],
                 [2,4,8],
                 [3,1,7]], dtype = 'float32')
tf.linalg.matrix_rank(A) ## dependent
```

```
<tf.Tensor: shape=(), dtype=int32, numpy=2>
```

```
A = tf.constant([[2,3],
                [4,7]], dtype = 'float32')
tf.linalg.matrix_rank(A) ## independent
```

```
<tf.Tensor: shape=(), dtype=int32, numpy=2>
```

Python

Matrix Inverse

- is a matrix A^{-1} such that $AA^{-1} = \mathrm{I}$, where I is the identity matrix.

$$\begin{bmatrix} 4 & 3 \\ 3 & 2 \end{bmatrix} \begin{bmatrix} -2 & 3 \\ 3 & -4 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad-bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

Matrix Inverse

```
# example with dependent matrix
import numpy as np

A = np.array([[1,4,6],
              [2,4,8],
              [3,1,7]], dtype = 'float32')
np.linalg.det(A) ## dependent

## there is NO inverse
```

```
0.0
```

```
A = np.array([[2,3],
               [4,7]], dtype = 'float32')
np.linalg.det(A) ## independent
```

```
2.0
```

```
np.linalg.inv(A)
```

```
array([[ 3.5, -1.5],
       [-2. ,  1. ]], dtype=float32)
```

Matrix Inverse

```
# example with dependent matrix
import tensorflow as tf

A = tf.constant([[1,4,6],
                 [2,4,8],
                 [3,1,7]], dtype = 'float32')
tf.linalg.det(A) ## dependent

## there is NO inverse
```

```
<tf.Tensor: shape=(), dtype=float32, numpy=0.0>
```

```
A = tf.constant([[2,3],
                [4,7]], dtype = 'float32')
tf.linalg.det(A) ## independent
```

```
<tf.Tensor: shape=(), dtype=float32, numpy=2.0>
```

```
tf.linalg.inv(A)
```

```
<tf.Tensor: shape=(2, 2), dtype=float32, numpy=
array([[ 3.5, -1.5],
       [-2. ,  1. ]], dtype=float32)>
```

Python

Sparse vs Dense Matrices

- a sparse matrix is a matrix in which most of the elements are zero

Dense Matrix

1	2	31	2	9	7	34	22	11	5
11	92	4	3	2	2	3	3	2	1
3	9	13	8	21	17	4	2	1	4
8	32	1	2	34	18	7	78	10	7
9	22	3	9	8	71	12	22	17	3
13	21	21	9	2	47	1	81	21	9
21	12	53	12	91	24	81	8	91	2
61	8	33	82	19	87	16	3	1	55
54	4	78	24	18	11	4	2	99	5
13	22	32	42	9	15	9	22	1	21

Sparse Matrix

1	.	3	.	9	.	3	.	.	.
11	.	4	.	.	.	.	.	2	1
.	.	1	.	.	.	4	.	1	.
8	.	.	.	3	1	.	.	.	.
.	.	.	9	.	.	1	.	17	.
13	21	.	9	2	47	1	81	21	9
.	.	.	.	.	.	.	.	.	.
.	.	.	.	19	8	16	.	.	55
54	4	.	.	.	11	.	.	.	.
.	.	2	.	.	.	.	22	.	21

Sparse vs Dense Matrices

```
## sparse matrix
A = np.array([[1, 0, 0, 1, 0, 0], [0, 0, 2, 0, 0, 1], [0, 0, 0, 2, 0, 0]])
print(A)
```

```
[[1 0 0 1 0 0]
 [0 0 2 0 0 1]
 [0 0 0 2 0 0]]
```

If sparse matrices are big, Python will use a different method of storing for efficiency.

```
import scipy.sparse as sparse

# convert to sparse matrix (CSR method)
S = sparse.csr_matrix(A)
print(S)
```

```
  (0, 0)        1
  (0, 3)        1
  (1, 2)        2
  (1, 5)        1
  (2, 3)        2
```

Sparse vs Dense Matrices

We can revert a sparse matrix to a normal matrix by using the `.todense()` or `.toarray()` numpy methods.

```
# convert to dense
S.todense()
```

```
matrix([[1, 0, 0, 1, 0, 0],
        [0, 0, 2, 0, 0, 1],
        [0, 0, 0, 2, 0, 0]])
```

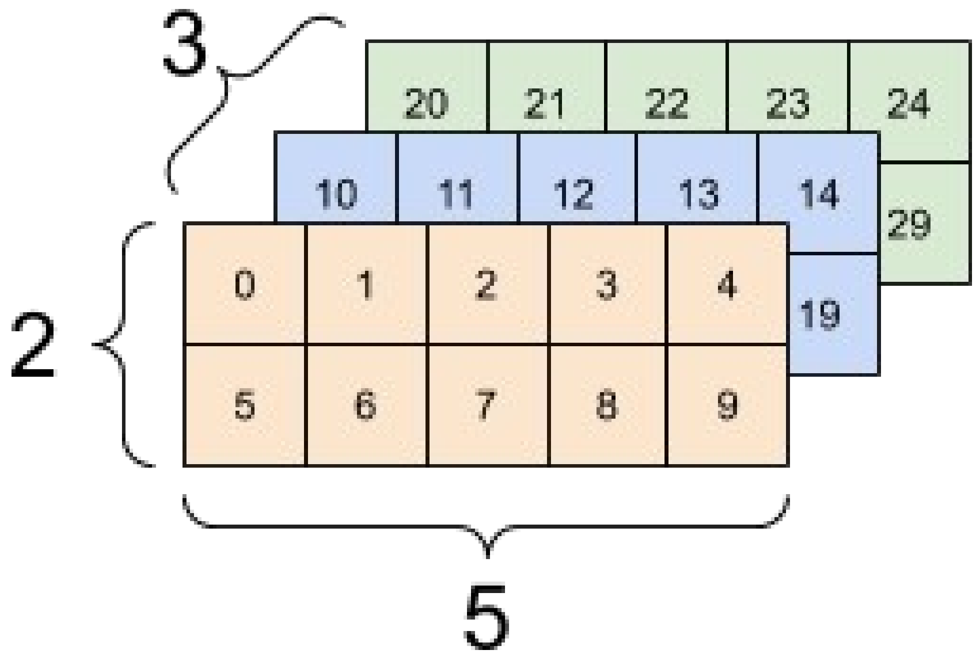
```
# convert to dense
S.toarray()
```

```
array([[1, 0, 0, 1, 0, 0],
       [0, 0, 2, 0, 0, 1],
       [0, 0, 0, 2, 0, 0]])
```

Python

Tensors

- A Tensor is a generalization of Vectors and Matrices to higher dimensions



Tensors

🎯 **NumPy**

```
import numpy as np

# let's create a tensor of 3 matrices, each 3 by 3 (3,3,3)
t1 = np.array([[[10, 11, 12], [13, 14, 15], [16, 17, 18]],
               [[20, 21, 22], [23, 24, 25], [26, 27, 28]],
               [[30, 31, 32], [33, 34, 35], [36, 37, 38]]])
```

```
t1
```

```
array([[[10, 11, 12],
        [13, 14, 15],
        [16, 17, 18]],

       [[20, 21, 22],
        [23, 24, 25],
        [26, 27, 28]],

       [[30, 31, 32],
        [33, 34, 35],
        [36, 37, 38]]])
```

Tensors

🎯 **Tensorflow**

```
import tensorflow as tf

# let's create a tensor of 3 matrices, each 3 by 3 (3,3,3)
t1 = tf.constant([[[10, 11, 12], [13, 14, 15], [16, 17, 18]],
                  [[20, 21, 22], [23, 24, 25], [26, 27, 28]],
                  [[30, 31, 32], [33, 34, 35], [36, 37, 38]]])
```

```
t1.shape
```

```
TensorShape([3, 3, 3])
```

```
t1.ndim
```

```
3
```

Python