Cheatsheets / **Linear Algebra**

codecademy

Linear Algebra

Scalars, Vectors, and Matrices

Scalars, *vectors*, and *matrices* are fundamental structures of linear algebra, and understanding them is integral to unlock the concepts of deep learning.

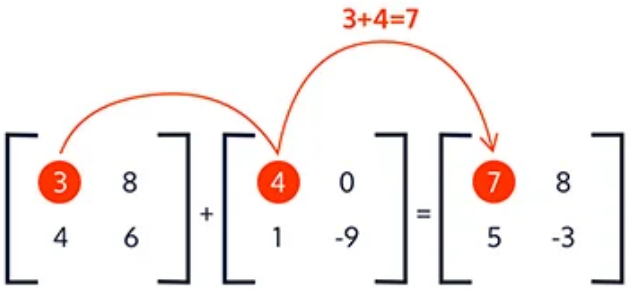
- A scalar is a singular quantity like a number.
- A vector is an array of numbers (scalar values).
- A matrix is a grid of information with rows and columns.

They can all be represented in Python using the NumPy library.

```
import numpy

# Scalar code example
x = 5

# Vector code example
x = numpy.array([1, 2, 3, 4])

# Matrix code example
x = numpy.array([1, 2, 3], [4, 5, 6],
[7, 8, 9])
```

Matrix Addition

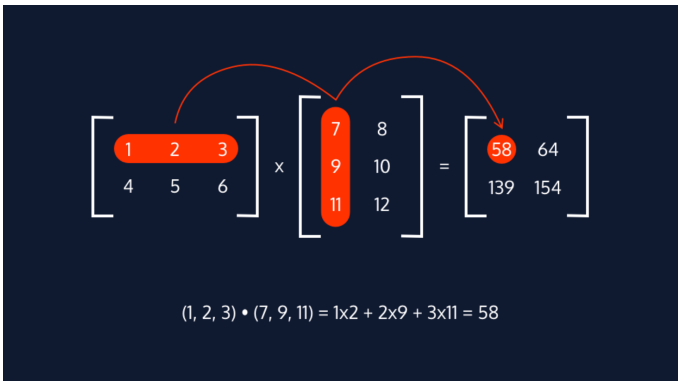
In *matrix addition*, corresponding matrix entries are added together.



codecademy

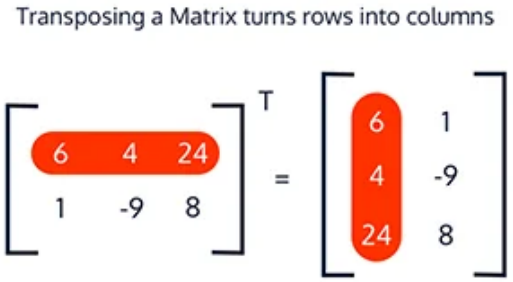
Matrix Multiplication

In *matrix multiplication*, the number of rows of the first matrix must be equal to the number of columns of the second matrix. The dot product between each row and column of the matrix and placed as an entry into the resulting matrix as shown in the image.



Matrix Transpose

A matrix transpose switches the rows and columns of a matrix.



codecademy

Vector Magnitude

To calculate the magnitude of a vector, use the following formula:

$$||v|| = \sqrt{v_1^2 + v_2^2 + \cdots +$$

```
import numpy as np

v = np.array([3, 6, -6])
np.linalg.norm(v)
# equals 9
```

For example, if we have the following 3D vector:

$$v = \begin{bmatrix} 3 \\ 6 \\ -6 \end{bmatrix}$$

To calculate the magnitude, we do the following:

$$||v|| = \sqrt{3^2 + 6^2 + (-6)^2}$$
$$||v|| = \sqrt{81}$$
$$||v|| = 9$$

We can also calculate the magnitude of a vector using the np.linalg.norm() function from the NumPy library.

codecademy

Basic Vector Operations

Vectors can be added and subtracted from each other when they are of the same dimension (same number of components). Doing so adds or subtracts corresponding elements, resulting in a new vector of the same dimension as the two being summed or subtracted. Any vector can also be multiplied by a scalar, which results in every element of that vector being multiplied by that scalar individually.

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} + 2 \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} - 3 \begin{bmatrix} x_3 \\ y_3 \\ z_3 \end{bmatrix}$$

Using NumPy, we can add equally sized vectors and matrices together using built-in Python addition between NumPy arrays. We can also use built-in Python multiplication to perform scalar multiplication on NumPy arrays. The code example shows an example implementation of both of these.

```
import numpy as np

# Matrix Addition
A = np.array([[1,2],[3,4]])
B = np.array([[-4,-3],[-2,-1]])
A + B
'''
This outputs
[[ -3 -1]
 [1 3]]
'''


# Scalar Multiplication
a = np.array([1,2])
4 * a

'''
This outputs
[4 8]
'''
```

codecademy

Dot Product

The *dot product* of two vectors measures how much one vector "goes into" the other vector by summing the products of the vectors' corresponding components. The formula for this is:

$$a \cdot b = \sum_{i=1}^{n} a_i b_i$$

To recall how to use this formula, consider the following two vectors:

$$a = \begin{bmatrix} 4 \\ 0 \\ -1 \end{bmatrix}, b = \begin{bmatrix} 2 \\ 8 \\ -2 \end{bmatrix}$$

To find the dot product between these two vectors, we do the following:

$$a \cdot b = 4 * 2 + 0 * -8 + -$$

We can use NumPy to compute vector dot products by using the np.dot() function. An example implementation is shown.

```
import numpy as np

v = np.array([4,0,-1])
u = np.array([2,8,-2])
np.dot(v,u)
# equals 10
```

codecademy

Augmented Matrix

A linear system of equations can be represented in matrix form using an *augmented matrix*, which takes the form *[A | b]* if we have the equation *Ax=b*. Let's say we have the following equations:

$$x + y + z = 32y + 4z = 6$$

We can represent this with the following augmented matrix:

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & 3 \\ 0 & 2 & 3 & 6 \\ 4 & 0 & 1 & 5 \end{array}\right]$$

codecademy

Inverse Matrix

The *inverse of a matrix, A^{-1}*, is one where the following equation is true:

$$AA^{-1} = A^{-1}A = I$$

We can use *Gauss-Jordan elimination* to solve for the inverse of a square matrix by hand (if one exists). With NumPy, we can use `np.linalg.inv()` to solve for the inverse of a square matrix. An example implementation is shown in the code block.

```
import numpy as np

A = np.array([[1,2],[3,4]])
print(np.linalg.inv(A))

"""
This outputs:
[[-2.   1. ]
 [ 1.5 -0.5]]
"""
```

codecademy

Identity Matrx

The *identity matrix* is a square matrix of elements equal to 0 except for the elements along the diagonal that are all equal to 1.

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

Any matrix multiplied by the identity matrix, either on the left or right side, will be equal to itself.

$$\begin{bmatrix} 4 & 3 & -2 \\ -3 & 0 & 6 \\ 1 & 2 & 8 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

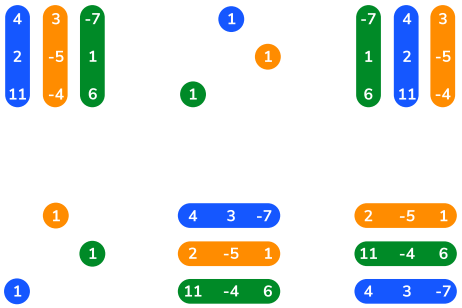
Using NumPy, we can create any *n x n* identity matrix using the np.eye() function. It takes one argument which determines the size of the matrix. A code implementation example is shown.

```
# 4x4 identity matrix
identity = np.eye(4)

'''
In the output terminal, identity renders
as:
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
'''
```

codecademy

Permutation Matrix

A *permutation matrix* is a square matrix that allows us to flip rows and columns of a separate matrix.



codecademy

NumPy Vectors and Matrices

We can represent both vectors and matrices using NumPy arrays.
For example, the following creates a NumPy array representation of a vector:

```
v = np.array([1, 2, 3, 4, 5, 6
```

We can also create a matrix, which is the equivalent of a two-dimensional NumPy array, using a nested Python list:

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

Matrices can also be created by combining existing vectors using the `np.column_stack()` function:

```
v = np.array([-2,-2,-2,-2])
u = np.array([0,0,0,0])
w = np.array([3,3,3,3])

A = np.column_stack((v, u, w))
```

This outputs:

```
[[-2  0  3]
 [-2  0  3]
 [-2  0  3]
 [-2  0  3]]
```

Print Share