Vectors and Vector Arithmetic

WHAT IS A VECTOR

Un vector es una **tupla** de uno o más valores llamados **escalares**. Son usualmente representados usando un carácter en minúscula, como v, mostrados de manera vertical u horizontal

Por ejemplo:

$$v = (v_1, v_2, v_3) \qquad v = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}$$

Donde v_1, v_2, v_3 son valores escalares, usualmente valores reales.

DEFINING A VECTOR

Podemos definir un vector en Python como un **arreglo** creado a partir de una **lista** de números. Véase el Ejemplo 1.

VECTOR ARITHMETIC

VECTOR ADDITION

Dos vectores de igual longitud pueden ser **sumados** para crear un tercer vector y éste tendrá el mismo largo que los otros dos vectores.
Matemáticamente se escribe como:

$$c = (a_1 + b_1, a_2 + b_2, a_3 + b_3)$$

Podemos sumar vectores en Python como indica en el Ejemplo 2.

VECTOR SUBTRACTION

Dos vectores de igual longitud pueden ser **restados** para crear un tercer vector y éste tendrá el mismo largo que los otros dos vectores.
Matemáticamente se escribe como:

$$c = (a_1 - b_1, a_2 - b_2, a_3 - b_3)$$

Podemos restar vectores en Python como indica el Ejemplo 3.

VECTOR MULTIPLICATION

Dos vectores de igual longitud pueden ser **multiplicados** para crear un tercer vector y éste tendrá el mismo largo que los otros dos vectores.
Matemáticamente se escribe como:

$$c = (a_1 \times b_1, a_2 \times b_2, a_3 \times b_3)$$

Podemos multiplicar vectores en Python como indica el Ejemplo 4.

VECTOR DIVISION

Dos vectores de igual longitud pueden ser **divididos** para crear un tercer vector y éste tendrá el mismo largo que los otros dos vectores.
Matemáticamente se escribe como:

$$c = \left(\frac{a_1}{b_1}, \frac{a_2}{b_2}, \frac{a_3}{b_3}\right)$$

Podemos dividir vectores en Python como indica el Ejemplo 5.

VECTOR DOT PRODUCT

Podemos calcular el **producto punto** entre dos vectores como la suma de sus elementos multiplicados. Esta operación puede ser usada en Machine Learning para calcular la **suma ponderada** de un vector.
Matemáticamente se escribe como:

$$c = (a_1 b_1 + a_2 b_2 + a_3 b_3)$$

Podemos calcular el producto punto en Python como indica el Ejemplo 6.

VECTOR-SCALAR MULTPLICATION

Un vector puede ser **multiplicado** por un **escalar** s, en efecto escalando la **magnitud** del vector y la multiplicación es realizada en **cada elemento** del vector.
Matemáticamente se escribe como:

$$c = (s \times v_1, s \times v_2, s \times v_3)$$

Podemos calcular el producto punto en Python como indica el Ejemplo 7.

Vector Norms

VECTOR NORM

La **longitud de un vector** es referida como la **norma o magnitud** de un vector y describe la **extensión** de éste en un espacio. Siempre es un **número positivo**, exceptuando a los vectores que tienen en todas sus **entradas** el valor 0. Es calculada usando una medida que indica la **distancia** del vector al origen.

VECTOR L^1 NORM

La longitud de un vector puede ser calculada usando la norma L^1 y su notación es $\|v\|_1$. De igual manera, a esta longitud se le conoce como la **norma del taxista** o la **norma Manhattan**.

$$L^1(v) = \|v\|_1$$

Esta norma es calculada como la **suma de los valores absolutos** del vector. En efecto, la norma es el cálculo de la distancia Manhattan del origen al vector.

$$\|v\|_1 = |a_1| + |a_2| + |a_3|$$

Podemos calcularla en Python como indica el Ejemplo 1.
La norma L^1 es usualmente ocupada como un **método de regularización** al entrenar modelos de Machine Learning, por ejemplo, para mantener pequeños los coeficientes de un modelo para volverlo menos complejo.

VECTOR L^2 NORM

La longitud de un vector puede ser calculada usando la norma L^2 y su notación es $\|v\|_2$

$$L^2(v) = \|v\|_2$$

Esta norma calcula la **distancia** de un vector al **origen**. De igual manera, es conocido como la **norma Euclideana** y es calculada como la **distancia Euclideana al origen**. El resultado es una distancia positiva pues se calcula como la raíz cuadrada de la suma del cuadrado de los valores del vector.

$$\|v\|_2 = \sqrt{a_1^2 + a_2^2 + a_3^2}$$

Podemos calcularla en Python como indica el Ejemplo 2.
Igualmente, la norma L^2 es usualmente ocupada como un **método de regularización** al entrenar modelos de Machine Learning.

VECTOR MAX NORM

La longitud de un vector puede ser calculado usando la norma máxima. Éste es denotado como L^∞.

$$L^\infty(v) = \|v\|_\infty$$

Esta norma es calculada como el máximo valor absoluto de los valores del vector.

$$\|v\|_\infty = \max\{a_1, a_2, a_3\}$$

Podemos calcularla en Python como indica el Ejemplo 3.
La norma L^∞ es también ocupada como un **método de regularización** al entrenar modelos de Machine Learning, como en **redes neuronales**.

Matrices and Matrix Arithmetic

WHAT IS A MATRIX

Una matriz es un **arreglo** escalar de **dos dimensiones** con una o más columnas y una o más filas. Su notación es normalmente una letra mayúscula y sus entradas son indicadas por su fila (i) y columna (j).

Por ejemplo, una matriz A de tres filas y dos columnas.

$$A = \left((a_{1,1}, a_{1,2}), (a_{2,1}, a_{2,2}), (a_{3,1}, a_{3,2})\right)$$

O bien,

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \end{pmatrix}$$

DEFINING A MATRIX

Podemos definir en Python una matriz como un arreglo de dos dimensiones creado a partir de una lista de listas. Véase el Ejemplo 1

MATRIX ARITHEMTIC

MATRIX ADDITION

Dos matrices de igual dimensión pueden ser **sumadas** para crear una tercera matriz. Los elementos de la nueva matriz serán calculados como la **suma** de los elementos en cada matriz.
Matemáticamente se escribe como:

$$C = \begin{pmatrix} a_{1,1} + b_{1,1} & a_{1,2} + b_{1,2} \\ a_{2,1} + b_{2,1} & a_{2,2} + b_{2,2} \\ a_{3,1} + b_{3,1} & a_{3,2} + b_{3,2} \end{pmatrix}$$

Podemos sumar matrices en Python como indica en el Ejemplo 2.

MATRIX SUBTRACTION

Dos matrices de igual dimensión pueden ser **restadas** para crear una tercera matriz. Los elementos de la nueva matriz serán calculados como la **resta** de los elementos en cada matriz.
Matemáticamente se escribe como:

$$C = \begin{pmatrix} a_{1,1} - b_{1,1} & a_{1,2} - b_{1,2} \\ a_{2,1} - b_{2,1} & a_{2,2} - b_{2,2} \\ a_{3,1} - b_{3,1} & a_{3,2} - b_{3,2} \end{pmatrix}$$

Podemos sumar matrices en Python como indica en el Ejemplo 3.

MATRIX MULTIPLICATION (HADAMARD PRODUCT)

Dos matrices de igual dimensión pueden ser **multiplicadas** para obtener una tercera matriz. Esta operación es conocida como el **producto Hadamard**. Dado que no es la multiplicación de matrices usual, se utiliza un círculo $\circ$ como notación.

$$C = A \circ B$$

Los elementos de la nueva matriz serán calculados como la **multiplicación** de los elementos en cada matriz.
Matemáticamente se escribe como:

$$C = \begin{pmatrix} a_{1,1} \times b_{1,1} & a_{1,2} \times b_{1,2} \\ a_{2,1} \times b_{2,1} & a_{2,2} \times b_{2,2} \\ a_{3,1} \times b_{3,1} & a_{3,2} \times b_{3,2} \end{pmatrix}$$

Podemos implementar esta multiplicación en Python como indica el Ejemplo 4

MATRIX DIVISION

Dos matrices de igual dimensión pueden ser **divididas** para crear una tercera matriz. Los elementos de la nueva matriz serán calculados como la **división** de los elementos en cada matriz.

$$C = \begin{pmatrix} \frac{a_{1,1}}{b_{1,1}} & \frac{a_{1,2}}{b_{1,2}} \\ \frac{a_{2,1}}{b_{2,1}} & \frac{a_{2,2}}{b_{2,2}} \\ \frac{a_{3,1}}{b_{3,1}} & \frac{a_{3,2}}{b_{3,2}} \end{pmatrix}$$

Podemos dividir matrices en Python como indica el Ejemplo 5

MATRIX-MATRIX MULTIPLICATION

La multiplicación de matrices, también conocida como el **producto punto de matrices**, implica una regla por la cual no todas las matrices pueden multiplicarse entre sí. La regla es la siguiente:

El número de columnas n en la primera matriz A debe ser igual al número de filas m de la segunda matriz B.

Así, resultará una nueva matriz de dimensión de $m \times k$ si la primera matriz es de dimensión $m \times n$ y la segunda, de $n \times k$.
La intuición dentro de esta operación es que estamos calculando el **producto punto** entre cada fila de la matriz A con cada columna de la matriz B.

Por ejemplo, si tomamos las matrices $A = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \end{pmatrix}$ y $B = \begin{pmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{pmatrix}$ obtendremos la matriz siguiente:

$$C = \begin{pmatrix} a_{1,1} \times b_{1,1} + a_{1,2} \times b_{2,1} & a_{1,1} \times b_{1,2} + a_{1,2} \times b_{2,2} \\ a_{2,1} \times b_{1,1} + a_{2,2} \times b_{2,1} & a_{2,1} \times b_{1,2} + a_{2,2} \times b_{2,2} \\ a_{3,1} \times b_{1,1} + a_{3,2} \times b_{2,1} & a_{3,1} \times b_{1,2} + a_{3,2} \times b_{2,2} \end{pmatrix}$$

Podemos implementar esta operación en Python como indica el Ejemplo 6

MATRIX-VECTOR MULTIPLICATION

Una matriz y un vector pueden **multiplicarse** siempre cuando se cumpla la regla ya antes mencionada.

Por ejemplo, si tenemos la matriz $A = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \end{pmatrix}$ y el vector $v = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$ obtenemos la siguiente matriz al multiplicarse:

$$C = \begin{pmatrix} a_{1,1} \times v_1 + a_{1,2} \times v_2 \\ a_{2,1} \times v_1 + a_{2,2} \times v_2 \\ a_{3,1} \times v_1 + a_{3,2} \times v_2 \end{pmatrix}$$

Podemos implementar esta operación en Python como indica el Ejemplo 7

MATRIX-SCALAR MULTIPLICATION

Una matriz puede ser **multiplicada** por un escalar resultando una matriz de la **misma dimensión** en donde todos sus elementos están siendo multiplicados por el escalar.
Matemáticamente se escribe como:

$$C = \begin{pmatrix} a_{1,1} \times b + a_{1,2} \times b \\ a_{2,1} \times b + a_{2,2} \times b \\ a_{3,1} \times b + a_{3,2} \times b \end{pmatrix}$$

Podemos implementarla en Python como indica el Ejemplo 8

Types of Matrices

SQUARE MATRIX

Una **matriz cuadrada** es una matriz en la cual se tiene el **mismo número** de filas que de columnas.
Dado que el número de filas y de columnas son iguales, la **dimensión** es usualmente denotada por $n \times n$ y el tamaño de la matriz es llamado **orden** por lo que una matriz de orden m es una matriz de $m \times m$. El vector de valores en la diagonal de la matriz es llamado la **diagonal principal**.

SYMMETRIC MATRIX

Una **matriz simétrica** es un tipo de matriz cuadrada donde el triángulo superior **es el mismo** que el triángulo inferior. Por ejemplo:

$$M = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 2 \\ 3 & 2 & 1 \end{pmatrix}$$

Una matriz simétrica siempre es igual a su transpuesta, esto es $M = M^T$.

TRIANGULAR MATRIX

Una **matriz triangular** es un tipo de matriz cuadrada que tiene todos sus valores en el **triángulo superior o inferior** y los elementos restantes son llenados con el valor **cero**. Una matriz que tiene sus valores en el triángulo superior es llamada **matriz triangular superior**, mientras que una matriz que tiene sus valores en el triángulo inferior es llamada **matriz triangular inferior**.
En Python podemos calcular una matriz triangular como indica el Ejemplo 1

DIAGONAL MATRIX

Una **matriz diagonal** es una en la cual los valores **fuera** de la diagonal principal son el **valor cero**. Ésta es usualmente denotada con la variable D y puede ser representado como un vector.

Por ejemplo, una matriz diagonal de 3×3 se vería como:

$$D = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix}$$

Si lo denotamos como un vector:

$$d = \begin{pmatrix} d_{1,1} \\ d_{2,2} \\ d_{3,3} \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

Una matriz diagonal no necesariamente debe ser **cuadrada**. En el caso de una matriz rectangular, la diagonal deberá **cubrir** la dimensión **más pequeña**, por ejemplo:

$$D = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

En Python podemos crear una matriz diagonal de una matriz existente, o bien, transformar un vector en una matriz diagonal, tal como indica el Ejemplo 2.

IDENTITY MATRIX

Una **matriz identidad** es una matriz cuadrada que **no cambia** a un vector cuando es multiplicado por ésta. Todos los valores escalares en la **diagonal principal** tienen el **valor uno**, mientras que los demás valores, **cero**. Es representada por la variable I o I^n donde n representa la dimensión de la matriz.
Podemos crear una matriz de este estilo en Python como indica el Ejemplo 3.

ORTHOGONAL MATRIX

Dos vectores son **ortogonales** si su **producto punto** es igual a **cero**. Si la magnitud de cada vector es 1 entonces los vectores son llamados **ortonormales** pues son **ortogonales y normalizados**.
Una **matriz ortogonal** es un tipo de matriz cuadrada en la cual sus columnas y filas son **vectores ortogonales unitarios**, esto es, que sus **filas** son mutualmente ortonormales y sus **columnas** son mutualmente ortonormales. Usualmente es denotada por Q y está definida de la siguiente manera:

$$Q^T \cdot Q = Q \cdot Q^T = I$$

También decimos que una matriz es **ortogonal** si su transpuesta es **igual** a su inversa.

$$Q^T = Q^{-1}$$

Las matrices ortogonales son muy usadas en **transformaciones lineales**, tales como reflexiones o permutaciones.
Podemos implementar y verificar en Python todas las equivalencias antes mencionadas tal como indica el Ejemplo 4.

Matrix Operations

TRANSPOSE

Podemos **transponer** una matriz para crear una nueva con el número de filas y columnas intercambiadas. Esta operación **no** tiene efecto si la matriz es **simétrica**.
En Python es posible implementar esta operación tal como indica el Ejemplo 1.

INVERSE

La **inversa** de una matriz es aquella que cuando se **multiplica** con la matriz inicial obtenemos la **matriz identidad**. Dada una matriz A, encuentra la matriz B tal que se cumpla lo siguiente:

$$AB = BA = I^n$$

Una matriz puede ser invertida en Python tal como indica el Ejemplo 2.

TRACE

La **traza** de una matriz es la **suma** de los valores de la **diagonal principal** de la matriz y se describe usando la notación $tr(A)$. Por ejemplo, para una matriz de 3×3 tenemos que:

$$tr(A) = a_{1,1} + a_{2,2} + a_{3,3}$$

Podemos obtener en Python la traza de una matriz como indica el Ejemplo 3.

DETERMINANT

El **determinante** de una matriz, denotado por $\det(A)$ o en algunos casos por $|A|$, es una **representación escalar** de su volumen pues describe la **relación geométrica** de los vectores que forman las filas de la matriz. En otras palabras, nos dice el **volumen** de una caja con los lados dados por las **filas** de dicha matriz.
Técnicamente, el determinante es el **producto** de los **eigenvalores** de la matriz y la intuición detrás de este concepto es que describe la forma en que una matriz **escalará** a otra cuando son **multiplicadas**.
Un determinante 1 de **preservará** el espacio de otra matriz y uno de 0 indica que la matriz **no** puede ser **invertida**.
En Python podemos obtener el determinante de una matriz tal como indica el Ejemplo 4.

RANK

El **rango** de una matriz es la **estimación** del número de filas o columnas **independientes** de una matriz, usualmente denotado por $rango(A)$.
Una intuición es considerarlo como el **número de dimensiones** que marcan todos sus vectores. Por ejemplo, un rango de 0 indica que los vectores marcan un punto, un rango de 1 indica que los vectores forman una línea y así sucesivamente.
Éste es estimado **numéricamente** por un método de descomposición, comúnmente por la **Descomposición de Valores Singulares**.
En Python podemos estimar el rango de un vector y una matriz tal como indica el Ejemplo 5 y Ejemplo 6, respectivamente.

Sparse Matrices

SPARSE MATRIX

Una matriz dispersa es una matriz compuesta, en su mayoría, por valores cero.
La escasez de una matriz puede ser cuantificada con una puntuación, la cual es el número de valores cero entre el número total de elementos.

$$Escasez = \frac{\#\ de\ elementos\ con\ valor\ cero}{total\ de\ elementos}$$

PROBLEMS WITH SPARSITY

SPACE COMPLEXITY

Matrices muy grandes ocupan mucha memoria y muchas de ellas son dispersas.

TIME COMPLEXITY

Dado que las matrices dispersas contienen en su mayor parte valores cero, realizar operaciones con ellas puede tardar mucho tiempo pues conlleva sumar o multiplicar valores cero.

WORKING WITH SPARSE MATRICES

Una solución para representar y trabajar con matrices dispersas es usar una alternativa en la **estructura de los datos** para así ignorar los valores cero.
Algunos ejemplos son:

SPACE COMPLEXITY

Se usa un diccionario donde cada índice de la fila y columna es mapeado a un valor.

LIST OF LISTS

Cada fila de la matriz es guardada como una lista, y en cada sublista está contenido el índice de la columna y el valor.

COORDINATE LIST

Se crea una lista de tuplas, en donde cada tupla contiene el índice la fila, de la columna y el valor.

COMPRESSED SPARSE ROW

La matriz dispersa es representada usando tres arreglos unidimensionales para los valores diferentes de cero, las extensiones de las filas y los índices de las columnas.

COMPRESSED SPARSE COLUMN

Lo mismo que en Compressed Sparse Row, sólo que las columnas son leídas antes que las filas.

SPARSE MATRICES IN PYTHON

Podemos convertir una matriz densa a una matriz dispersa en Python usando la representación CSR tal como indica el Ejemplo 1.
Si queremos saber la escasez de una matriz en Python, debemos realizar la operación que indica el Ejemplo 2.

Tensors and Tensor Arithmetic

WHAT ARE TENSORS

Un tensor es una generalización de los vectores y matrices como arreglos multidimensionales. Un vector es un tensor de primer orden y una matriz es un tensor de segundo orden. Su notación es una letra mayúscula representando al tensor y minúsculas con enteros como subíndices, las cuales representan los valores escalares del tensor.

$$T = \begin{pmatrix} t_{1,1,1} & t_{1,2,1} & t_{1,3,1} \\ t_{2,1,1} & t_{2,2,1} & t_{2,3,1} \\ t_{3,1,1} & t_{3,2,1} & t_{3,3,1} \end{pmatrix}, \begin{pmatrix} t_{1,1,2} & t_{1,2,2} & t_{1,3,2} \\ t_{2,1,2} & t_{2,2,2} & t_{2,3,2} \\ t_{3,1,2} & t_{3,2,2} & t_{3,3,2} \end{pmatrix}, \begin{pmatrix} t_{1,1,3} & t_{1,2,3} & t_{1,3,3} \\ t_{2,1,3} & t_{2,2,3} & t_{2,3,3} \\ t_{3,1,3} & t_{3,2,3} & t_{3,3,3} \end{pmatrix}$$

TENSORS IN PYTHON

Podemos crear tensores en Python como lista de listas, tal como indica el Ejemplo 1

TENSOR ARITHMETIC

TENSOR ADDITION

La suma de dos tensores con la misma dimensión nos da como resultado un nuevo tensor con la misma dimensión.

$$A = \begin{pmatrix} a_{1,1,1} & a_{1,2,1} & a_{1,3,1} \\ a_{2,1,1} & a_{2,2,1} & a_{2,3,1} \end{pmatrix}, \begin{pmatrix} a_{1,1,2} & a_{1,2,2} & a_{1,3,2} \\ a_{2,1,2} & a_{2,2,2} & a_{2,3,2} \end{pmatrix}$$

$$B = \begin{pmatrix} b_{1,1,1} & b_{1,2,1} & b_{1,3,1} \\ b_{2,1,1} & b_{2,2,1} & b_{2,3,1} \end{pmatrix}, \begin{pmatrix} b_{1,1,2} & b_{1,2,2} & b_{1,3,2} \\ b_{2,1,2} & b_{2,2,2} & b_{2,3,2} \end{pmatrix}$$

$$C = A + B$$
$$= \begin{pmatrix} a_{1,1,1} + b_{1,1,1} & a_{1,2,1} + b_{1,2,1} & a_{1,3,1} + b_{1,3,1} \\ a_{2,1,1} + b_{2,1,1} & a_{2,2,1} + b_{2,2,1} & a_{2,3,1} + b_{2,3,1} \end{pmatrix}, \begin{pmatrix} a_{1,1,2} + b_{1,1,2} & a_{1,2,2} + b_{1,2,2} & a_{1,3,2} + b_{1,3,2} \\ a_{2,1,2} + b_{2,1,2} & a_{2,2,2} + b_{2,2,2} & a_{2,3,2} + b_{2,3,2} \end{pmatrix}$$

En Python podemos sumar dos tensores como indica el Ejemplo 2

TENSOR SUBTRACTION

La resta de dos tensores con la misma dimensión nos da como resultado un nuevo tensor con la misma dimensión.

$$C = A - B$$
$$= \begin{pmatrix} a_{1,1,1} - b_{1,1,1} & a_{1,2,1} - b_{1,2,1} & a_{1,3,1} - b_{1,3,1} \\ a_{2,1,1} - b_{2,1,1} & a_{2,2,1} - b_{2,2,1} & a_{2,3,1} - b_{2,3,1} \end{pmatrix}, \begin{pmatrix} a_{1,1,2} - b_{1,1,2} & a_{1,2,2} - b_{1,2,2} & a_{1,3,2} - b_{1,3,2} \\ a_{2,1,2} - b_{2,1,2} & a_{2,2,2} - b_{2,2,2} & a_{2,3,2} - b_{2,3,2} \end{pmatrix}$$

En Python podemos restar dos tensores como indica el Ejemplo 3

TENSOR HADAMARD PRODUCT

La multiplicación de dos tensores con la misma dimensión nos da como resultado un nuevo tensor con la misma dimensión.

$$C = A \circ B$$
$$= \begin{pmatrix} a_{1,1,1} \times b_{1,1,1} & a_{1,2,1} \times b_{1,2,1} & a_{1,3,1} \times b_{1,3,1} \\ a_{2,1,1} \times b_{2,1,1} & a_{2,2,1} \times b_{2,2,1} & a_{2,3,1} \times b_{2,3,1} \end{pmatrix}, \begin{pmatrix} a_{1,1,2} \times b_{1,1,2} & a_{1,2,2} \times b_{1,2,2} & a_{1,3,2} \times b_{1,3,2} \\ a_{2,1,2} \times b_{2,1,2} & a_{2,2,2} \times b_{2,2,2} & a_{2,3,2} \times b_{2,3,2} \end{pmatrix}$$

En Python podemos multiplicar dos tensores como indica el Ejemplo 4

TENSOR DIVISION

La división de dos tensores con la misma dimensión nos da como resultado un nuevo tensor con la misma dimensión.

$$C = \frac{A}{B} = \begin{pmatrix} \frac{a_{1,1,1}}{b_{1,1,1}} & \frac{a_{1,2,1}}{b_{1,2,1}} & \frac{a_{1,3,1}}{b_{1,3,1}} \\ \frac{a_{2,1,1}}{b_{2,1,1}} & \frac{a_{2,2,1}}{b_{2,2,1}} & \frac{a_{2,3,1}}{b_{2,3,1}} \end{pmatrix}, \begin{pmatrix} \frac{a_{1,1,2}}{b_{1,1,2}} & \frac{a_{1,2,2}}{b_{1,2,2}} & \frac{a_{1,3,2}}{b_{1,3,2}} \\ \frac{a_{2,1,2}}{b_{2,1,2}} & \frac{a_{2,2,2}}{b_{2,2,2}} & \frac{a_{2,3,2}}{b_{2,3,2}} \end{pmatrix}$$

En Python podemos dividir dos tensores como indica el Ejemplo 5

TENSOR DIVISION

El producto de tensores es usualmente denotado por $\otimes$. Dado un tensor A con q dimensiones y un tensor B con r dimensiones, el producto de éstos nos dará un nuevo tensor con dimensión $q + r$.
El producto de tensores para vectores es el siguiente:

$$a = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} \qquad b = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

$$C = a \otimes b = \begin{pmatrix} a_1 \times \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \\ a_2 \times \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} a_1 \times b_1 & a_1 \times b_2 \\ a_2 \times b_1 & a_2 \times b_2 \end{pmatrix}$$

El producto de tensores para matrices es el siguiente:

$$a = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix}$$

$$b = \begin{pmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{pmatrix}$$

$$C = a \otimes b = \begin{pmatrix} a_{1,1} \times \begin{pmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{pmatrix} & a_{1,2} \times \begin{pmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{pmatrix} \\ a_{2,1} \times \begin{pmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{pmatrix} & a_{2,2} \times \begin{pmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{pmatrix} \end{pmatrix}$$

El producto de tensores puede ser implementado como indica el Ejemplo 6