**PREWORK**

**SESIÓN 06**

**Objetivo:**

En esta sesión aprenderemos como funcionan las redes neuronales artificiales como clasificadores.

**Pre-requisitos (técnicos):**

Necesitaremos que tengas presente los programas que utilizamos en la sesión 02, ya que los utilizaremos mucho durante la sesión para evaluar el desempeño de nuestras redes neuronales.

**Pre-requisito 1: Álgebra lineal.**

Antes de comenzar a trabajar con redes neuronales artificiales, necesitaremos que te familiarices con algunos conceptos de álgebra lineal. El álgebra lineal es una rama de las matemáticas que utiliza vectores, matrices y ecuaciones lineales. No te preocupes demasiado por las matemáticas detrás de ellas, en este documento abordaremos todo de la forma más sencilla posible.

Lo primero que vamos a necesitar que tengas claro es el concepto de “vector”, también conocido como array en sistemas computacionales. Es algo muy común que manejamos los programadores y no es otra cosa más que una colección ordenada de números. Los vectores son sumamente útiles para almacenar cuantos datos necesitemos en una lista ordenada.

Es muy probable que hayas manejado arrays anteriormente, sobre todo si eres programador, sin embargo si nunca te has enfrentado a ellos, no te preocupes: solo tienes que visualizarlo de esta manera:

Supongamos que tienes a 3 personas: Juan, María y Alfonso. Cada uno tiene cierta edad, por lo que podríamos utilizar tres variables para guardar sus edades, de la siguiente manera:

edadJuan = 18

edadMaria = 42

edadAlfonso = 28

Esto funciona perfectamente para tres personas, pero ¿Qué pasaría si nosotros quisiéramos tener una lista de – digamos-100,000 personas? Las cosas cambian ciertamente, y crear 100,000 variables puede que no sea la mejor idea. Podemos crear una sola variable que almacene todas las edades en una lista ordenada, la edad en el lugar 0 para Juan, la edad en el lugar 1 para María, en el lugar 2 para Alfonso, y así sucesivamente.

edades = [18, 42, 28]

Con eso hemos creado el primer paso: Un vector (o un array) de edades. La particularidad de los arrays es que pueden ser del tamaño que quieras, y almacenar cuantos datos desees (siempre y cuando quepan dentro de tu memoria RAM).

Si nuestros vectores de datos van a ser exclusivamente números, nosotros podríamos utilizar una herramienta llamada ***NumPy array*** que no es otra cosa más que una lista, pero con algunas herramientas útiles y optimizadas para manejar arrays. En general, si tus datos son números, quizás sea una buena idea utilizar un arreglo de NumPy. Esto se hace así:

import numpy as np

edades = np.array([18,42,28])

Con esto hemos creado nuestro array de datos. Pero ¿para que complicarnos la vida utilizando arrays? Sucede que nosotros podemos hacer cosas interesantes con los arrays y las operaciones se hacen en automático para todos los elementos. Por ejemplo, supongamos que Juan, María y Alfonso tienen sus salarios guardados en un Array:

salarios = np.array([10000, 25000, 8000])

Si el equipo de recursos humanos ha decidido subir el salario de todos los empleados un 10%, lo único que tiene que hacer es:

salarios = salarios \*1.10

Y todos los elementos se multiplicaron por 1.10 (el 10% de incremento). Como puedes ver es una herramienta sumamente útil para manejar una cantidad de datos importante de forma rápida y sin problemas.

**Pre-requisito 2: Producto Punto.**

El producto punto sirve para realizar operaciones de multiplicación y suma de forma muy rápida. Para no complicarnos demasiado con las matemáticas y continuar en una línea meramente de programador, imaginemos que nos piden lo siguiente:

“Juan, María y Alfonso son del área de Sistemas, y les vamos a subir el salario de la siguiente manera: A Juan un 10%, a María un 15% y a Alfonso un 20%. Queremos saber cuánto dinero tenemos que destinarle mensualmente al área de sistemas en total.”

Como podrás imaginar, lo que harás mentalmente será multiplicar el salario de juan un 10%, el de maría 15% y el de Alfonso un 20%, y una vez que tengas los nuevos salarios, sumarlos todos. A manera de programación:

Total = salarios[0] \* aumento[0] + salarios[1]\* aumento [1] + … + salarios[1000] \* aumento[1000]

¿Si te dijera que eso puedes hacerlo en un solo paso? Para eso sirve el producto punto. El producto punto multiplica un vector contra otro vector, y los resultados los va sumando en un solo valor: La suma de todas las multiplicaciones. Veamos un ejemplo en código:

Estos son nuestros salarios y los aumentos:

salarios = np.array([10000,25000,8000])

aumento = np.array([1.10,1.15,1.20])

Solo basta hacer:

nuevosSalarios = np.dot(salarios,aumento)

print(nuevosSalarios)

Y esto te da como resultado

> 49350.0

Como puedes ver, el producto punto te ayuda a agilizar el tedioso proceso de hacer ciclos, sumas y multiplicaciones, para adherir los datos en un solo punto. Si quisieras hacer el mismo proceso exactamente para toda la empresa, solo tienes que agregar arrays como sigue:

salarios = np.array([

[10000,25000,8000], # Sistemas

[5000,12500,4000], # Marketing

[10000,10000,10000], # Ventas

])

aumento = np.array([1.10,1.15,1.20])

nuevosSalarios = np.dot(salarios,aumento)

print(nuevosSalarios)

Esto te dará como resultado los incrementos de toda la empresa, área por área en el orden que lo suministraste:

> [49350. 24675. 34500.]

**¿Porque esto es importante?**

En esta sesión trataremos muchísimo el producto punto y los arrays para agilizar el proceso de nuestras redes neuronales. El producto punto servirá para que puedas procesar de forma automática una capa neuronal, no importando si utilizas 1 neurona artificial o 100,000. El código será exactamente el mismo. No te asustes con la parte matemática. En realidad usaremos muy poco las matemáticas, y mucho más los algoritmos.

**Quiz:**

**¿Qué es un vector?**

1. Una colección de valores numéricos.
2. Una estructura de Python
3. **Una colección ordenada de valores numéricos**
4. Un conjunto de datos.

**¿Por qué nos conviene utilizar álgebra lineal?**

1. **Acelera los procesos de cómputo y facilita la programación**
2. Permite realizar código estructurado y limpio
3. Permite encontrar soluciones al entrenar sistemas.
4. Empuja las barreras de Machine Learning

**¿Con que línea implementamos el producto punto?**

1. np.punto(vector1, vector2)
2. **np.dot(vector1, vector2)**
3. np.dot(variable1)
4. np.punto(vector1)

**¿Por qué necesitamos utilizar el producto punto en Redes Neuronales?**

1. **Para programar cuantas neuronas queramos con el mismo código.**
2. Para programar el entrenamiento de redes neuronales.
3. Para desglosar las neuronas en entradas y salidas.
4. Para facilitar la tarea de debuggeo.