NumPy

NumPy (Numerical Python) es una librería fundamental de Python para computación científica. Se especializa en:

- **Manejo eficiente de arrays**: Estructuras de datos similares a listas pero optimizadas para cálculos numéricos
- **Matrices multidimensionales**: Permite trabajar con datos en 1D (vectores), 2D (matrices), 3D y más dimensiones
- **Operaciones matemáticas**: Álgebra lineal, estadística y cálculos vectorizados (más rápidos que loops en Python)

¿Por qué es importante para IA? En Machine Learning, se traba constantemente con grandes conjuntos de datos representados como matrices y vectores. NumPy hace estos cálculos mucho más rápidos y eficientes.

Funciones de comunes de NumPy

np.array() - Crear Arrays

¿Qué hace? Convierte listas de Python en arrays de NumPy, que son más eficientes para cálculos numéricos.

¿Cuándo usarlo? Cuando necesitas transformar tus datos (listas) en un formato optimizado para operaciones matemáticas.

```
X = np.array([[1.0, 2.0], [2.0, 1.0]])
y = np.array([5.0, 4.0])
```

np.zeros() - Inicializar con Ceros

¿Qué hace? Crea un array lleno de ceros con la forma (shape) que especifiques.

¿Cuándo usarlo? Para inicializar pesos o parámetros de un modelo de IA antes de entrenar. Es común empezar con valores en cero.

```
w0 = np.zeros(X.shape[-1])  # [0. 0.]
```

.shape - Obtener Dimensiones

¿Qué hace? Es un atributo (no función) que te dice las dimensiones de tu array.

¿Cuándo usarlo? Para saber cuántos datos tienes (filas) y cuántas características tiene cada dato (columnas).

```
m = x.shape[0]  # número de filas
w0 = np.zeros(X.shape[-1])  # número de columnas
```

- `shape[0]`: filas
- `shape[1]` o `shape[-1]`: columnas

Operador @ - Multiplicación de Matrices

¿Qué hace? Realiza la multiplicación matricial (producto punto) entre matrices y vectores.

¿Cuándo usarlo? Es fundamental en IA para calcular predicciones. Por ejemplo, `X @ w` multiplica tus datos por los pesos del modelo.

```
y_estimada = x @ w + b
gradiente_w = -(1/m)*(x.T @ error)
```

Equivalente a `np.dot(x, w)`

.T - Transponer Matriz

¿Qué hace? Transpone una matriz, intercambiando filas por columnas.

¿Cuándo usarlo? Muy común al calcular gradientes en algoritmos de optimización. Si `x` es (4,2), entonces `x.T` es (2,4).

```
x.T @ error
```

- Intercambia filas por columnas
- Si `x` es (4,2), entonces `x.T` es (2,4)

np.square() - Elevar al Cuadrado

¿Qué hace? Eleva cada elemento del array al cuadrado.

¿Cuándo usarlo? Para calcular errores al cuadrado, como en la función de costo MSE (Mean Squared Error).

```
np.square(error)  # error²
```

.mean() - Calcular Promedio

¿Qué hace? Calcula el promedio (media aritmética) de todos los elementos del array.

¿Cuándo usarlo? Para obtener el error promedio de tu modelo o calcular el MSE (Mean Squared Error).

```
np.square(error).mean()  # MSE (Mean Squared Error)
error.mean()
```

np.abs() - Valor Absoluto

¿Qué hace? Calcula el valor absoluto de cada elemento (convierte negativos a positivos).

¿Cuándo usarlo? Para medir la magnitud del error sin importar si es positivo o negativo, o para verificar convergencia.

```
np.abs(gradiente_w).max()
```

.max() - Valor Máximo

¿Qué hace? Encuentra el valor más grande en el array.

¿Cuándo usarlo? Para verificar convergencia (si el gradiente máximo es pequeño, el modelo ya convergió).

```
np.abs(gradiente_w).max()
```

.copy() - Copiar Array

¿Qué hace? Crea una copia independiente del array.

¿Cuándo usarlo? Cuando quieres guardar el estado original de tus pesos antes de modificarlos. Sin `.copy()`, ambas variables apuntarían al mismo array.

```
w = w0.copy()
```

Otras funciones

Algebra Lineal

```
np.dot(a, b)           # Producto punto
np.linalg.norm(v)      # Norma de un vector
np.linalg.inv(A)       # Inversa de matriz
```

Estadistica

```
np.mean(x)             # Media
np.std(x)              # Desviación estándar
```

```
np.sum(x)               # Suma
np.min(x), np.max(x)    # Mínimo y máximo
```

Generar Datos

```
np.random.rand(3, 2)    # Números aleatorios uniformes [0,1)
np.random.randn(3, 2)   # Números aleatorios normales (media
=0, std=1)
np.linspace(0, 10, 50) # 50 números entre 0 y 10
np.arange(0, 10, 0.5)   # De 0 a 10 con paso 0.5
```

Manipular Arrays

```
x.reshape(2, 3)         # Cambiar forma
x.flatten()             # Convertir a 1D
np.concatenate([a,b])   # Unir arrays
x[x > 0]                # Filtrado booleano
```

Funciones Matematicas

```
np.exp(x)               # e^x
np.log(x)               # ln(x)
np.sqrt(x)              # √x
np.sin(x), np.cos(x)    # Trigonométricas
```