Preliminares del aprendizaje profundo

Los conceptos fundamentales del aprendizaje profundo abarcan desde la manipulación de datos y álgebra lineal hasta el cálculo, la diferenciación automática y la probabilidad. Estos preliminares son esenciales para comprender cómo funcionan los modelos de aprendizaje profundo y cómo se implementan en la práctica.

by Cristian Barrero

Manipulación de datos

Tensores

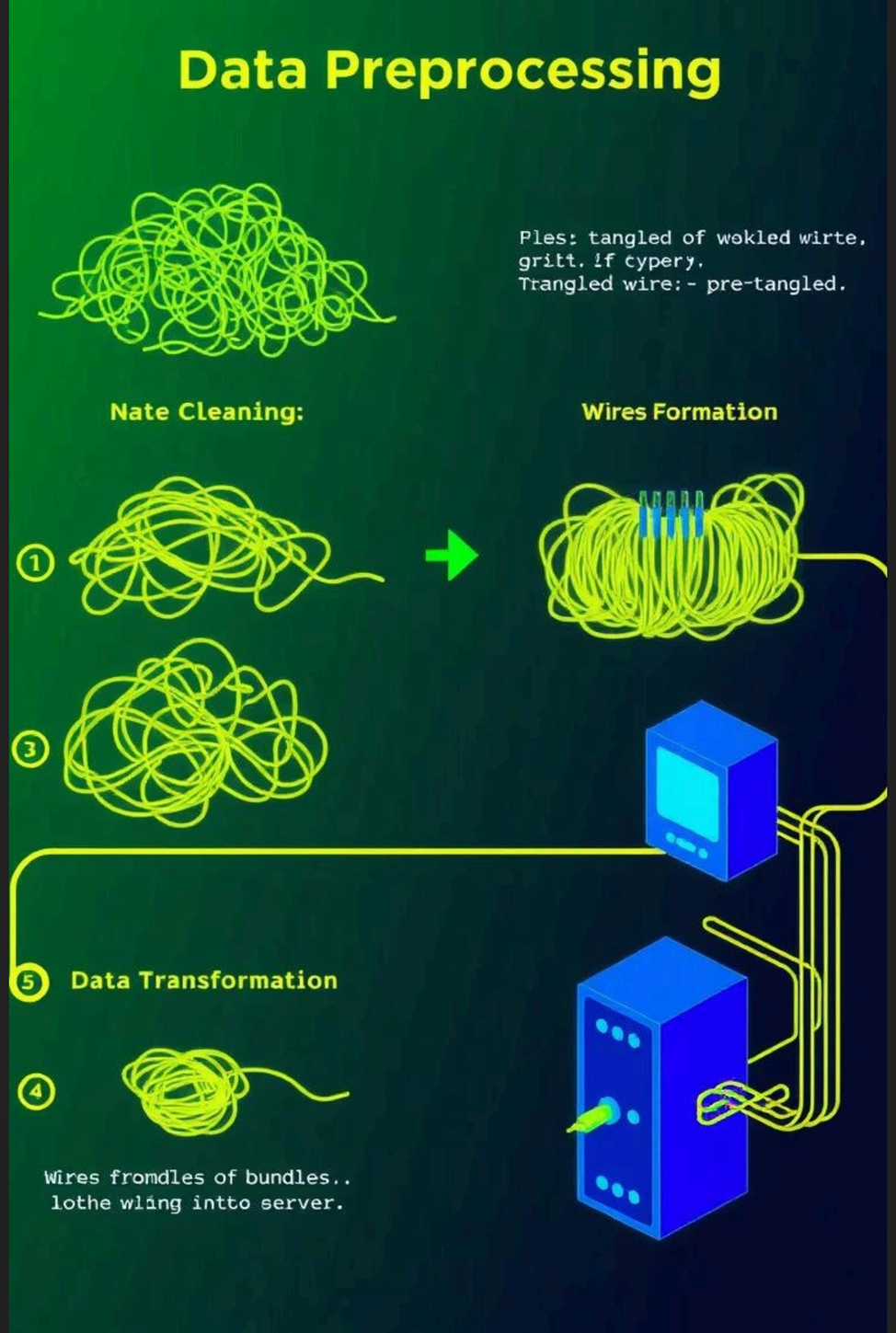
Los tensores son arreglos numéricos de *n* dimensiones (0-D escalares, 1-D vectores, 2-D matrices, etc.). Se crean y manipulan fácilmente con bibliotecas como PyTorch, MXNet, JAX o TensorFlow. Por ejemplo, torch.arange(12) genera un vector con valores 0&11. Se puede consultar el número de elementos con x.numel() (PyTorch) o x.size (MXNet).

Indexación y slicing

La indexación sigue la convención de Python: índices desde 0, índices negativos cuentan desde el final. Usando slices [inicio:fin] se obtiene un subarreglo (el índice final no se incluye). Por ejemplo, X[-1] devuelve la última fila de una matriz y X[1:3] las filas 1 y 2.

Operaciones elementales

Las operaciones aritméticas se aplican elemento a elemento. Por ejemplo, dados x=[1,2,4,8] e y=[2,2,2,2], entonces x+y=[3,4,6,10], x-y=[-1,0,2,6], x*y=[2,4,8,16], x/y=[0.5,1,2,4] y x**y=[1,4,16,64].



Preprocesamiento de datos

Lectura de datos

Antes de alimentar datos a un modelo debemos limpiarlos y formatearlos. Se suele usar **pandas** para leer datos tabulares, p.ej. archivos CSV. Por ejemplo, un CSV con columnas NumRooms, RoofType, Price se carga con pd.read_csv("datos.csv").

Preparación de datos

Primero se separan las *características* (inputs) de la *etiqueta* (target).

Por ejemplo, inputs = data.iloc[:,0:2] y targets = data.iloc[:,2]. Luego se manejan valores faltantes (NaN).

Conversión a tensores

Finalmente, convertimos los datos limpios en tensores para el modelo. En PyTorch, por ejemplo, X = torch.tensor(inputs.values) y y = torch.tensor(targets.values) produce tensores numéricos con los valores del *DataFrame*.

Álgebra lineal: Conceptos básicos

Escalares

Son simples números reales (0-D tensor). Ejemplo: x=3.0, y=2.0 (PyTorch). Se denotan con minúscula y realizan operaciones usuales: $x+y$=5, $x*y$=6, x/y=1.5, x^y=9.

Vectores

Arreglos de dimensión fija (1-D). Se denotan en negrita o con letra minúscula negrita, $x*=n$. Por ejemplo, torch.arange(3) genera [0,1,2]. Se indexan desde 0 (x[2]=2). La dimensión len(x)=n se refiere a n coordenadas.

Matrices

Arreglos 2-D (m×n). Denotadas con letra mayúscula, $A*=m×n$. Un elemento aij es fila i, columna j. Ejemplo: A = torch.arange(6).reshape(3,2) da la matriz [[0,1],[2,3],[4,5]]. La **transpuesta** AT intercambia filas y columnas.

Álgebra lineal: Operaciones avanzadas

Producto punto (dot)

Dado *x*,*y**= *d*, el producto punto *xTy* = 3*i xi yi*. Por ejemplo, si *x*=[0,1,2] e *y*=[1,1,1], entonces *xTy*=3. En código se usa torch.dot(x,y) o np.dot(x,y), o bien sum(x*y).

Producto matriz-vector

Si *A* es *m*×*n* y *x* un vector de longitud *n*, el producto *Ax* es un vector de longitud *m* cuyo i-ésimo componente es el producto punto de la fila i de *A* con *x*. En PyTorch se usa torch.mv(A,x) o el operador A @ x.

Producto matriz-matriz

Si *A* es *n*×*k* y *B* es *k*×*m*, entonces *AB* es *n*×*m* con elementos *cij* = 3*3*=1*k ai3 b3j*. Se calcula con torch.mm(A,B) o A@B (PyTorch) y np.dot(A,B) (NumPy).

Normas

Medidas de magnitud. La norma *32* de un vector *x* es |*x*|*2* = :(3*i xi2*), la longitud euclidiana. Por ejemplo, para *u*=[3,-4] se obtiene |*u*|*2*=5. En PyTorch torch.norm(u), NumPy np.linalg.norm(u), etc.

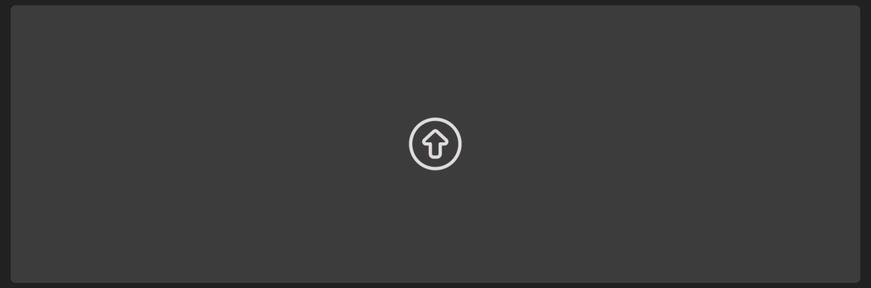
Cálculo: Derivadas y diferenciación

Derivadas

La *derivada* de una función $f(x) = x^3$ mide la tasa de cambio de *f* respecto a su argumento.

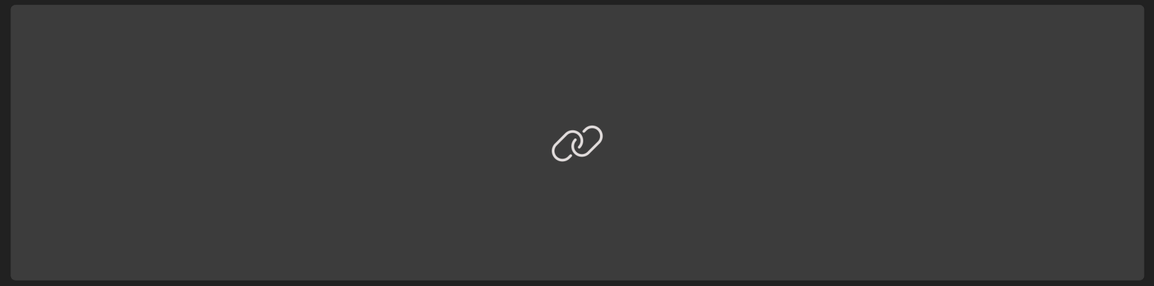
Visualización de funciones

Se incluyen utilidades en matplotlib para graficar funciones y sus tangentes.



Derivadas parciales y gradiente

Con funciones multivariables $y=f(x1,...,xn)$, la *derivada parcial* respecto a xi se obtiene tratando las demás variables como constantes.



Regla de la cadena

Para funciones compuestas $y=f(g(x))$, la derivada sigue la regla de la cadena: $dy/dx = dy/du \cdot du/dx$.

Diferenciación automática

Grafo computacional

Las librerías modernas calculan gradientes automáticamente creando un **grafo computacional** durante el *forward*. Por ejemplo, si queremos $y=2x^Tx$ (vector³escalar), primero definimos x con seguimiento de gradiente (requires_grad=True en PyTorch).

Cálculo de gradientes

Al computar y = 2 * dot(x,x), y luego llamar a y.backward(), la derivada 'xy = 4x queda almacenada en x.grad (e.g. [0,4,8,12] para x=[0,1,2,3]).

Desconectar parte del grafo

A veces no se desea propagar gradientes por ciertas sub-expresiones. Por ejemplo, con y = x*x, si hacemos u = y.detach() (PyTorch), entonces u se comporta como copia sin relación con x.

Control de flujo en Python

Incluso si la función incluye bucles, condicionales o cualquier lógica en Python, las librerías de *autograd* siguen construyendo el grafo en tiempo de ejecución.

Probabilidad y estadística: Conceptos básicos

Modelo probabilístico

Un modelo probabilístico asigna **probabilidades** (números en [0,1]) a eventos sobre un espacio de muestreo. Por ejemplo, al lanzar una moneda justa, $P(\text{cara}) = 0.5$, $P(\text{sello}) = 0.5$.

Probabilidad condicional

La **probabilidad condicional** se define como $P(B=b \mid A=a)$ ($P(A=a, B=b) / P(A=a)$), que refleja la probabilidad de $B=b$ dado que $A=a$ ocurrió.

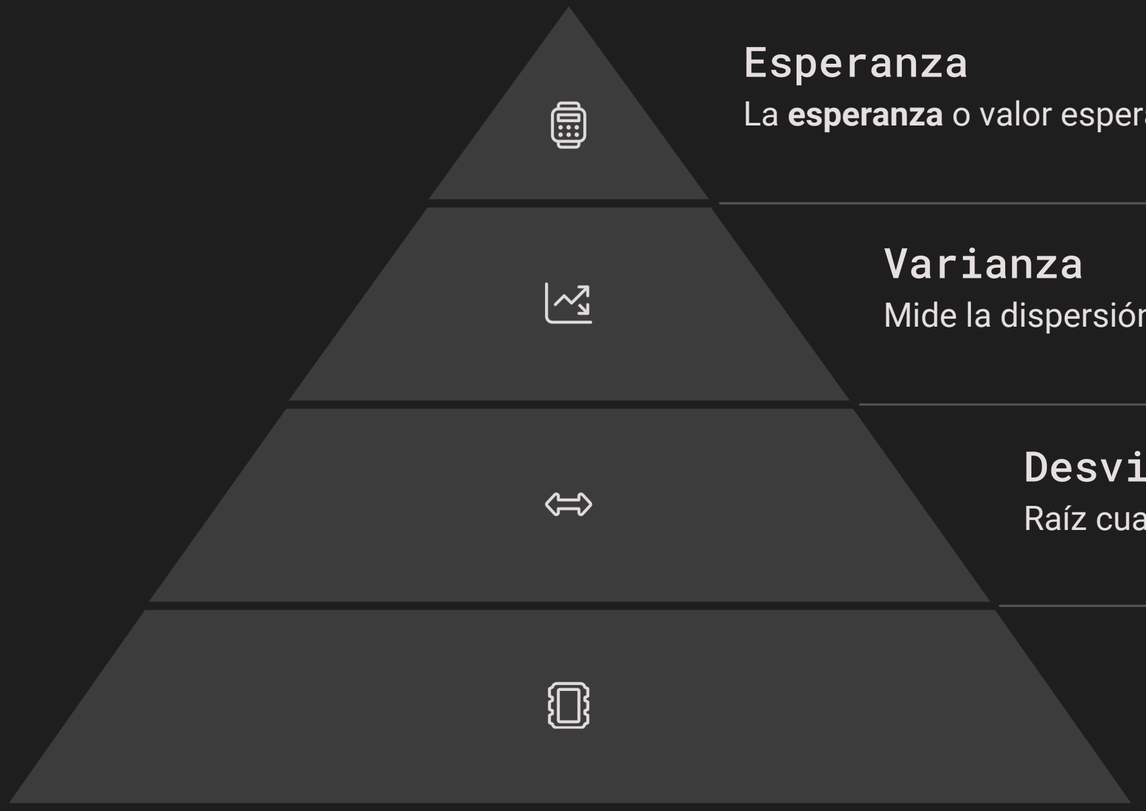
Variables aleatorias

Una **variable aleatoria** X es una función que asigna a cada resultado de S un valor (discreto o continuo). Por ejemplo, $X = \text{"es cara?"}$ vale 1 o 0.

Probabilidad conjunta

Con varias variables A, B definimos la **probabilidad conjunta** $P(A = a, B = b)$ que indica la probabilidad de que $A = a$ y $B = b$ ocurran simultáneamente.

Probabilidad y estadística: Medidas estadísticas



Esperanza

La **esperanza** o valor esperado de una variable aleatoria X discreta

Varianza

Mide la dispersión de los valores alrededor de la media

Desviación estándar

Raíz cuadrada de la varianza, misma unidad que los datos

Matriz de covarianza

Captura correlaciones entre variables aleatorias vectoriales

La **esperanza** o valor esperado de una variable aleatoria *X* discreta se define como *E*[*X*] = 3*x* *x*·*P*(*X*=*x*), que es el promedio ponderado de todos los posibles valores. La **varianza** mide la dispersión: *Var*[*X*]=*E*[(*X*-*E*[*X*])*2*] = *E*[*X2*] - (*E*[*X*])*2*. Para vectores aleatorios *x*, definimos la **matriz de covarianza** £ = *E*[(*x*-¿)(*x*-¿)*T*].

Documentación de la API

2

Métodos principales

Para consultar las funciones y clases disponibles en las librerías

100+

Funciones disponibles

En cada módulo de las bibliotecas de aprendizaje profundo

Para consultar las funciones y clases disponibles en las librerías, se recomienda:

- **Listar contenido de un módulo:** Usar la función dir(). Por ejemplo, dir(torch.distributions) enumera todas las distribuciones y transformaciones disponibles (Bernoulli, Normal, Multinomial, etc.). Esto ayuda a descubrir qué herramientas existen en un módulo.
- **Ayuda sobre una función/clase específica:** Usar help(). Por ejemplo, help(torch.ones) muestra la firma y descripción de torch.ones, indicando argumentos (size, dtype, device, requires_grad, etc.) y ejemplos. Del mismo modo, help(np.ones) da la documentación de mxnet.numpy.ones.

Estos métodos nos guían para explorar la API de PyTorch, MXNet, JAX o TensorFlow, cuyos sitios oficiales de documentación ofrecen información más extensa.