Regularización: Concepto general

Regularización es un conjunto de técnicas diseñadas para **reducir la complejidad efectiva de un modelo** y **mejorar su capacidad de generalización**.
En términos más formales, busca **controlar el sobreajuste (*overfitting*)** penalizando las soluciones "demasiado complejas" que ajustan demasiado bien los datos de entrenamiento, pero fallan con datos nuevos.

Tipos clásicos de regularización

Tipo	Fórmula	Intuición	Efecto típico
L2 (Ridge)	(R(w) = \	w\	_2^2 = \sum w_i^2)
L1 (Lasso)	(R(w) = \	w\	_1 = \sum
Elastic Net	Combinación L1 + L2	Mezcla suavidad y sparsity	Se usa en datos correlacionados.

🧠 En redes neuronales, L2 se implementa como **weight decay**, y L1 se usa menos, pero puede ser útil para reducir parámetros irrelevantes.

Regularización en redes neuronales

Más allá del L1/L2 clásico, existen técnicas *implícitas de regularización*, que actúan sobre la estructura o el proceso de entrenamiento:

Técnica	Qué hace	Intuición pedagógica
Dropout	Apaga aleatoriamente neuronas durante el entrenamiento	"Evita que las neuronas se copien entre sí".
Early Stopping	Detiene el entrenamiento cuando el error de validación deja de mejorar	"Evita que el modelo memorice los datos".
Batch Normalization / LayerNorm	Normaliza activaciones intermedias	"Suaviza el espacio de búsqueda".
Data Augmentation	Aumenta la diversidad de los datos de entrenamiento	"Hace que el modelo aprenda lo esencial y no los detalles".
Weight Decay (L2)	Penaliza pesos grandes en el optimizador	"Mantiene la red simple, sin depender de pesos enormes".

Intuición geométrica

Puedes imaginar la regularización como una **restricción en la forma del espacio de soluciones**.
Sin regularización, el modelo puede encontrar un mínimo muy profundo y estrecho (específico de los datos).
Con regularización, el modelo se ve forzado a encontrar un mínimo **más ancho y estable**, donde pequeñas variaciones en los datos no cambian mucho la salida — eso significa *mejor generalización*.

Ejemplo simple en código (PyTorch)

```
import torch
import torch.nn as nn
import torch.optim as optim

X = torch.randn(100, 10)
y = torch.randn(100, 1)

model = nn.Sequential(
    nn.Linear(10, 50),
    nn.ReLU(),
    nn.Linear(50, 1)
)

# Sin regularización
opt_plain = optim.Adam(model.parameters(), lr=1e-3)

# Con regularización L2 (weight decay)
opt_reg = optim.Adam(model.parameters(), lr=1e-3, weight_decay=1e-2)

# Comparación: weight_decay = λ * sum(w²)
```

💡 En *weight decay*, el optimizador automáticamente añade la penalización L2 en cada paso.
Esto previene que los pesos crezcan demasiado y ayuda al modelo a generalizar mejor.

En resumen

Concepto	Objetivo	Efecto observable
Regularización	Limitar la complejidad del modelo	Menor sobreajuste
Penalización L1/L2	Restringir magnitud de pesos	Reducción de varianza
Dropout	Romper co-adaptaciones	Más robustez
Early stopping	Prevenir memorización	Mejor generalización

Data augmentation	Aumentar variabilidad del input	Mayor tolerancia a ruido