

Construyendo una Red Neuronal desde Cero

Forward, pérdida, gradientes y actualización (sin nn.Module)

Cesar Garcia

2025

- Implementar una red neuronal **sin** `nn.Module`
- Entender el **forward pass** como álgebra lineal
- Calcular una pérdida escalar y obtener gradientes con Autograd
- Actualizar parámetros manualmente (un paso de SGD)
- Relacionar cada línea de código con un concepto (sin “magia”)

Qué significa “desde cero”

Alcance y restricciones

Hoy “desde cero” significa:

- Parámetros explícitos: W, b
- Operaciones explícitas: $XW + b$, activación, pérdida
- Actualización explícita: $\theta \leftarrow \theta - \eta \nabla_{\theta} L$

No significa:

- implementar Autograd
- escribir CUDA
- reescribir PyTorch

¿Qué cosas estás dispuesto a “tomar prestadas” de PyTorch y cuáles no?

La capa lineal

Una operación central

Una capa lineal es:

$$z = XW + b$$

Dimensiones típicas (mini-batch):

- $X \in \mathbb{R}^{N \times d}$
- $W \in \mathbb{R}^{d \times h}$
- $b \in \mathbb{R}^h$

Si X es 32×2 y queremos $h = 16$, ¿qué forma debe tener W ?

No linealidad

Sin activación, varias capas lineales colapsan en una sola.

Ejemplos comunes: - ReLU: $\max(0, z)$ - Tanh: $\tanh(z)$ - Sigmoid: $\sigma(z)$

¿Qué problema aparece si apilamos capas lineales sin activaciones?

Red mínima (2 capas)

Forward pass completo

Pipeline:

$$X \rightarrow (XW_1 + b_1) \rightarrow \text{ReLU} \rightarrow H \rightarrow (HW_2 + b_2) \rightarrow \text{logits}$$

$$\text{logits} \rightarrow \text{softmax} \rightarrow \hat{y}$$

(En clasificación, usualmente trabajamos con **logits** y usamos cross-entropy.)

¿Por qué solemos usar “logits” en lugar de probabilidades durante el entrenamiento?

De muchas predicciones a un número

La pérdida produce un **escalar**:

- Para clasificación: cross-entropy
- Para regresión: MSE

Intuición: - pérdida baja \rightarrow predicciones correctas - pérdida alta \rightarrow predicciones incorrectas

¿Por qué es útil que todo el batch se reduzca a un solo número?

Qué ocurre con `.backward()`

- Autograd ya construyó el grafo durante el forward
- `.backward()`:
 - aplica regla de la cadena
 - llena `W.grad`, `b.grad`, etc.

Importante: - Gradientes se **acumulan** si no los limpiamos

¿Por qué `grad` se acumula y por qué eso puede romper el entrenamiento?

Actualización manual (SGD)

Un paso de aprendizaje

Con learning rate η :

$$\theta \leftarrow \theta - \eta \nabla_{\theta} L$$

En código, típicamente: - entramos en `torch.no_grad()` - actualizamos tensores de parámetros - reiniciamos gradientes

¿Qué cambia si el learning rate es demasiado grande? ¿y si es demasiado pequeño?

Qué ganamos hoy

Hoy construimos un entrenamiento real con piezas explícitas:

- parámetros
- forward
- pérdida
- backward
- update

Próxima sesión: - `nn.Module` y `optim` harán esto más limpio, - pero ya sabes exactamente **qué están automatizando**.

¿Qué parte de este flujo te parece más “frágil” o fácil de romper?