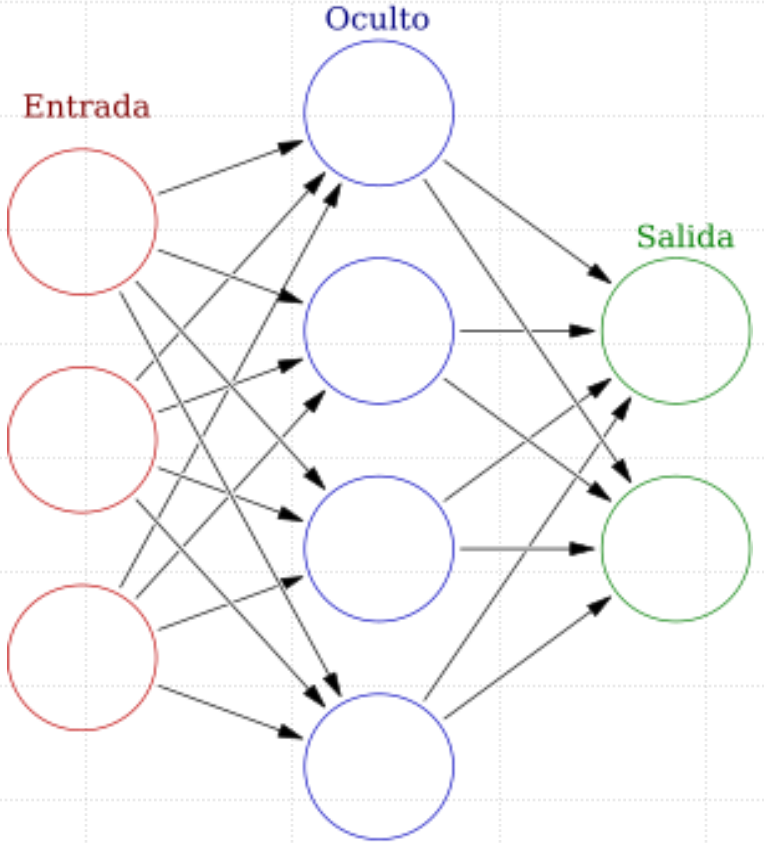
Capas densas (Fully Connected / Linear)

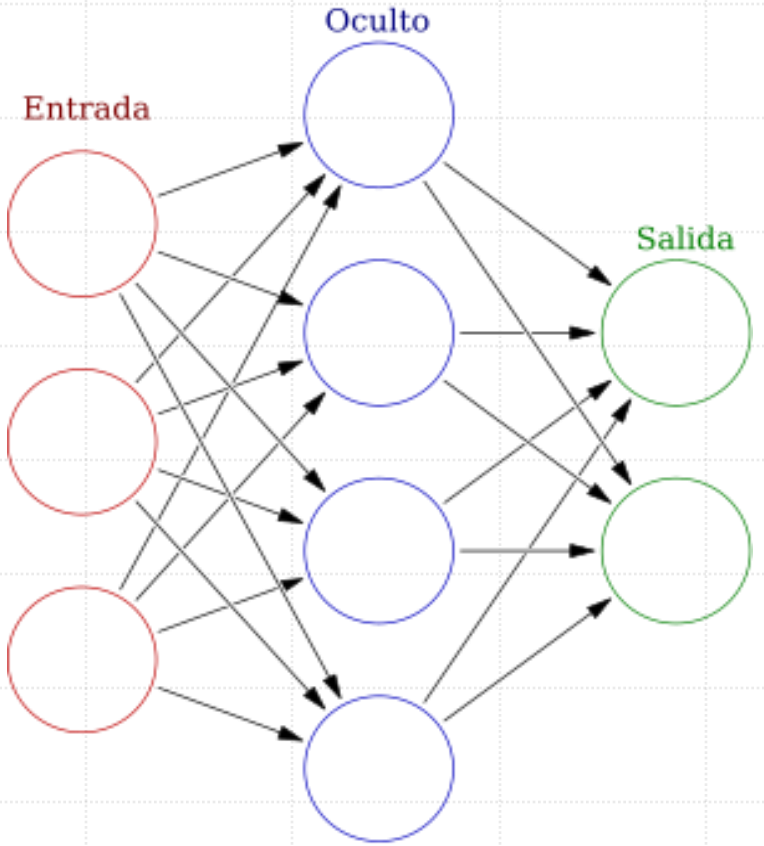
- $y=\sigma(Wx+b)$
  - y=activacion(Pesos*Input+Bias)
- Cada neurona ve todas las entradas de la capa anterior
- Uso:
  - Datos tabulares
  - embeddings,
  - “cabezas” de clasificación al final de una CNN/transformer



Shapes: entrada/salida

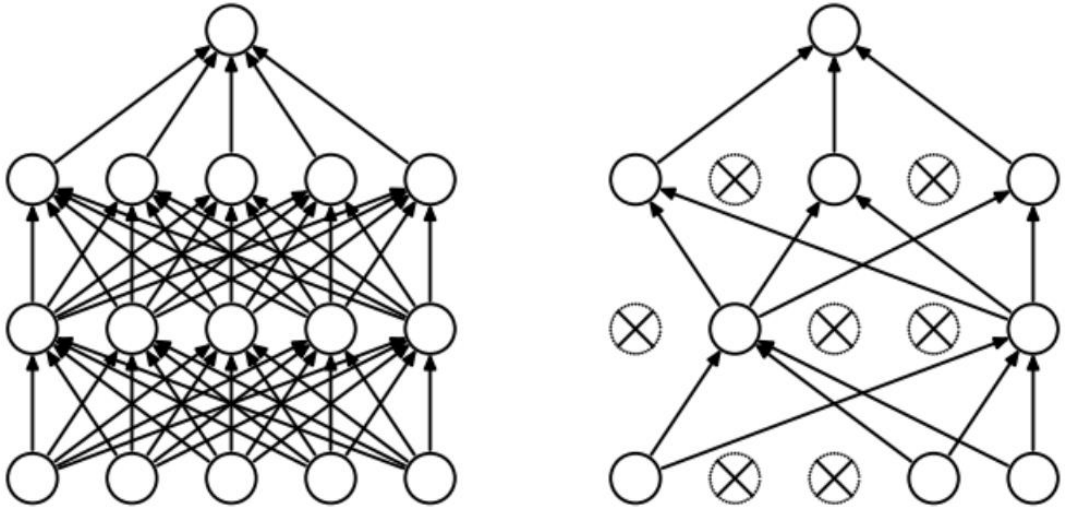
```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(input_shape=(32, 32, 3)),
    ...
    tf.keras.layers.Dense(10)
])
```

- Uso:
  - Definir vector de entrada y vector de salida
  - Necesario para calcular los pesos y conexiones entre capas



Dropout

- Durante entrenamiento **apagás neuronas al azar** para que la red aprenda patrones robustos y no “memorice”.
- Uso
  - Después de densas en MLP; en CNNs suele ir **después de bloques** o antes de la capa final de clasificación.
- Valores:
  - 0.1–0.5; más alto = más regularización

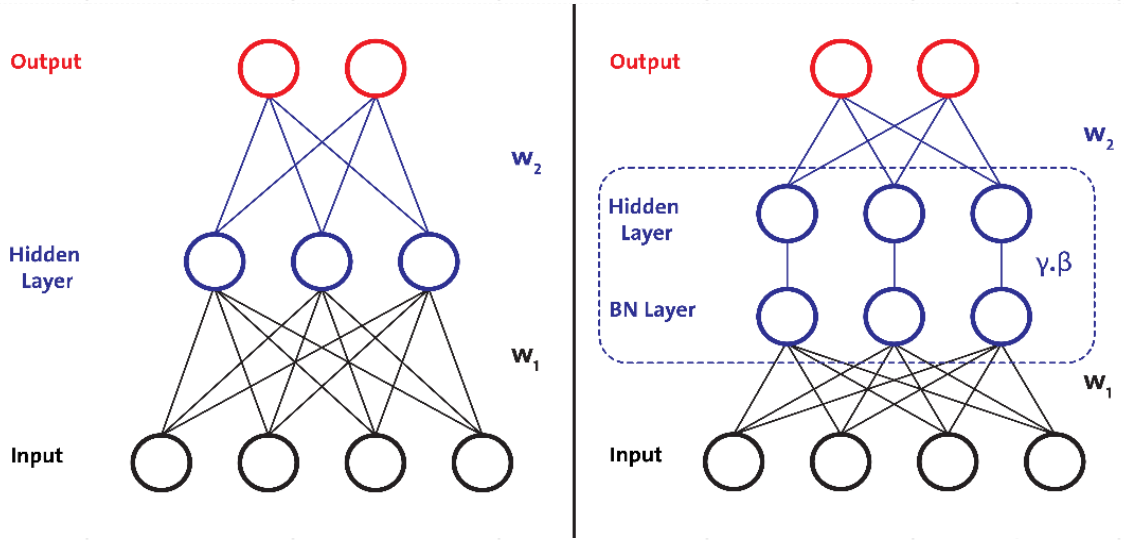


(a) Standard Neural Net

(b) After applying dropout.

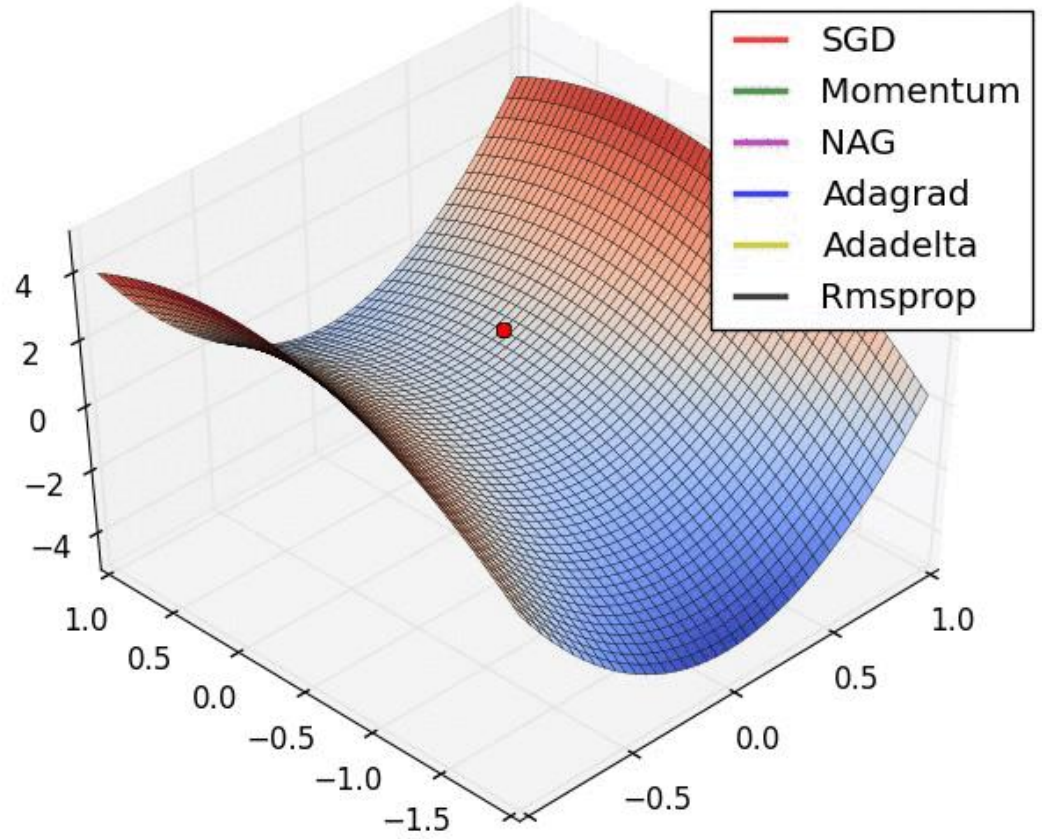
BatchNorm

- Normaliza la activación de cada canal/neuron para que el entrenamiento sea **más estable** (aprende con pasos más grandes).
- Uso
  - Acelera convergencia, permite **LR más altos**, estabiliza gradientes.



Optimizadores – ¿Qué hace un optimizador?

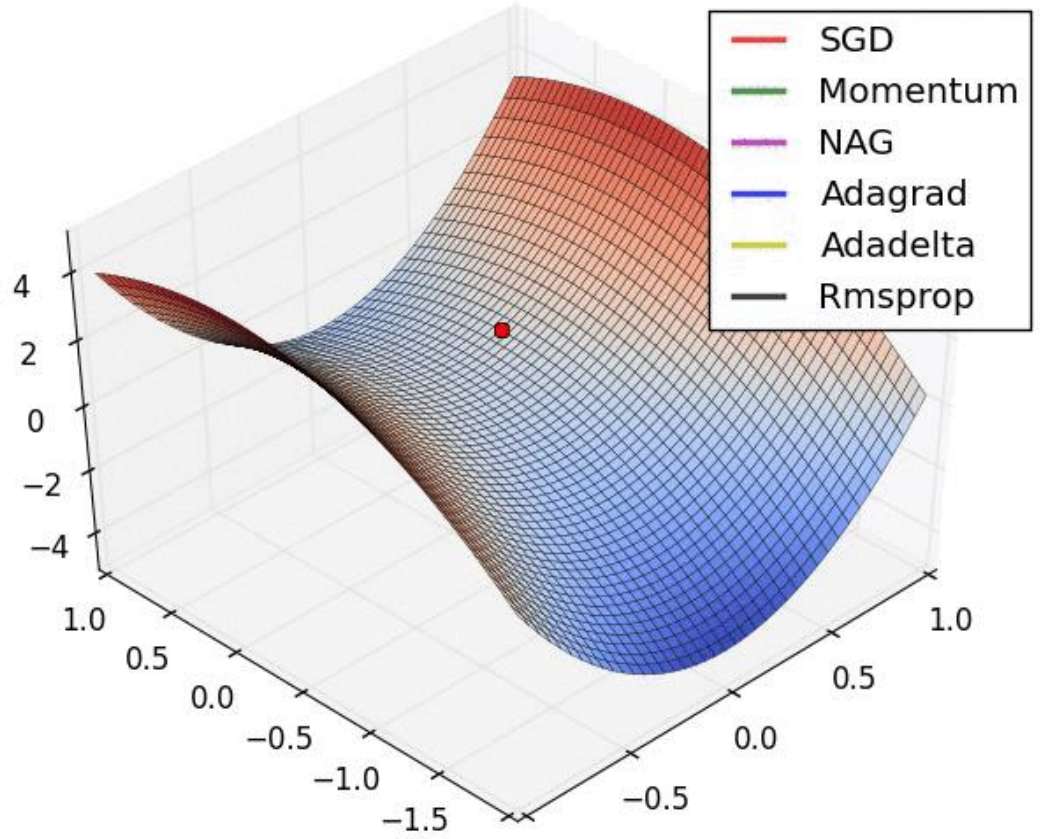
- **Objetivo**: encontrar pesos que minimicen la **función de pérdida**.
- **Componentes**:
  - **learning rate (LR)** = tamaño del paso
  - **momentum/adaptativo** = cómo "suaviza" o adapta el paso por parámetro.



Optimizadores - SGD

Estás bajando una montaña con pasos de tamaño fijo (**learning rate**). Si el terreno es ruidoso, te tambaleás.

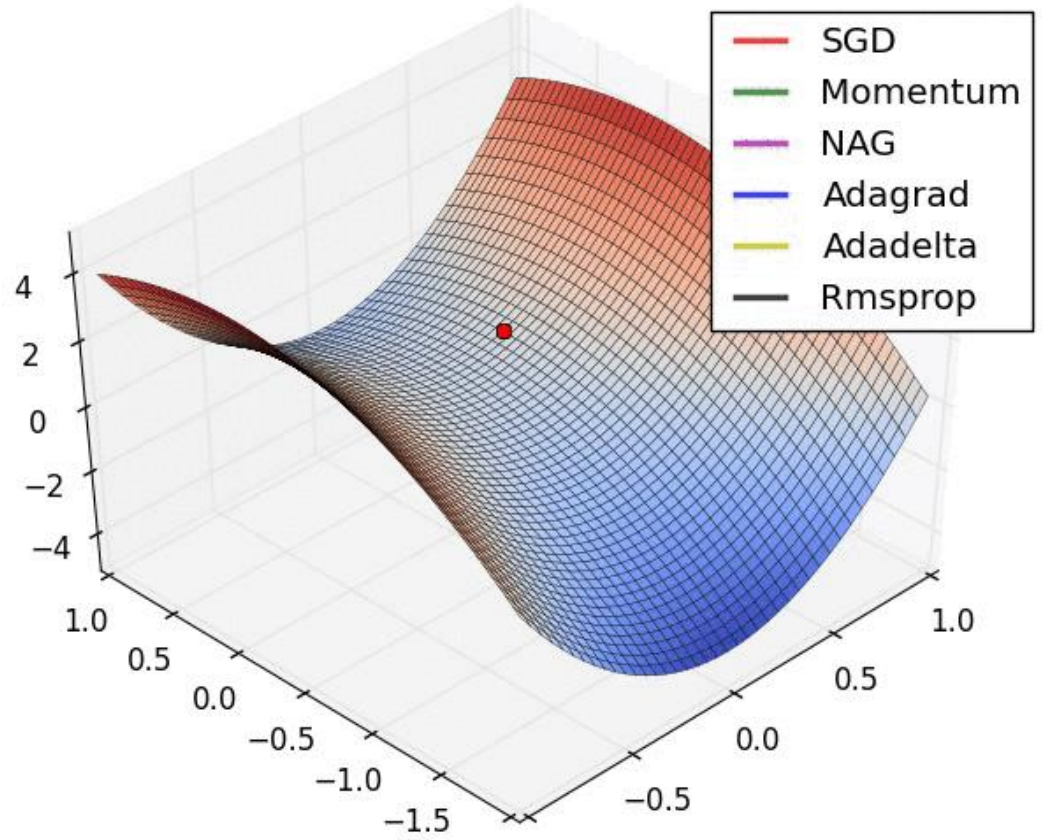
- Cuándo usarlo
  - Tenés **tiempo** para tunear **scheduler** y te importa **generalización** tope (suele ganar a largo plazo en visión).
- Hiperparámetros útiles
  - `lr`: empezar **alto**
  - `momentum`: `0.9` (default sólido).
  - `nesterov`: `True` puede ayudar a converger un poco mejor



Optimizadores – Adam

Cada peso tiene su **propio paso**; si un peso ve gradientes grandes o ruidosos, **se ajusta** para no pasarse.

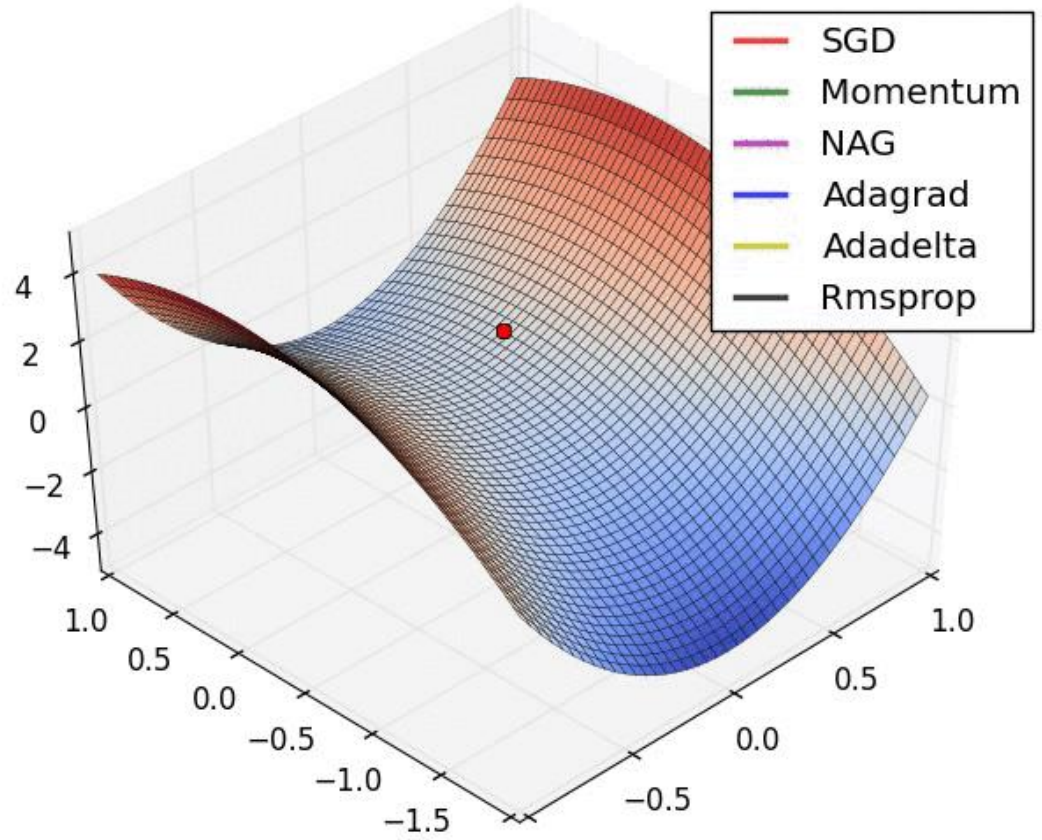
- Cuándo usarlo
  - **Baseline** rápido cuando querés resultados razonables **sin mucho tuning**.
- Hiperparámetros útiles
  - `lr`: **1e-3** suele ser buen inicio.
  - `betas`: `(0.9, 0.999)` default; rara vez cambiarlos.
  - `eps`: `1e-8` (estabilidad numérica).



Optimizadores - AdamW

Es Adam, pero el “freno” de **weight decay** está **separado** de cómo calcula sus pasos adaptativos. Regulariza **mejor**.

- Cuándo usarlo
  - Querés lo “mejor de dos mundos”: **convergencia rápida** + **mejor generalización** que Adam.
- Hiperparámetros útiles
  - `lr`: **3e-4** (inicio típico); si subís, subí de a poco.
  - `weight_decay`: **1e-2** (suele funcionar bien).
  - `betas`: `(0.9, 0.999)`; `eps=1e-8`.



Callbacks- Early Stopping: parar a tiempo

Evita entrenar de más cuando validación ya no mejora → ahorra tiempo y overfitting.

- Parámetros
  - monitor (ej. val_loss o val_macro_f1)
  - patience (p. ej. 5), mode (min o max)..

Callbacks- LR Schedulers

Subir/bajar el paso con intención

- **StepLR**: baja LR en escalones → simple, efectivo con SGD.
- **Cosine Annealing**: decae suavemente; evita saltos bruscos; **muy común** con AdamW.
- **OneCycle/Cyclical**: sube-baja el LR dentro de un rango para **salir de mesetas**.

Callbacks- Checkpoints

Guardar siempre el **mejor** por validación (no el último).

Reproducibilidad: poder retomar entrenamientos cortados y/o finetunear modelos entrenados.

TensorBoard: ver es entender