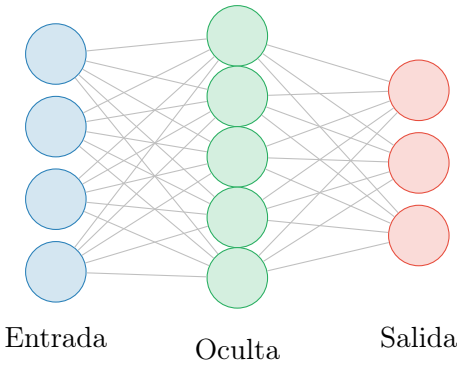
Fundamentos de Inteligencia Artificial

Para Deep Learning y Redes Neuronales



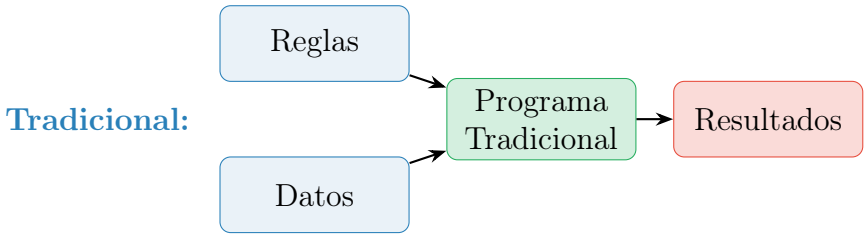
Conceptos Esenciales con Visualizaciones

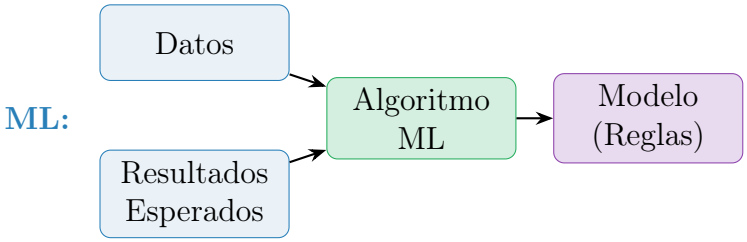
Preparacion para el Proyecto HAR

Índice

1. Introduccion: ¿Que es el Aprendizaje Automatico?

El **aprendizaje automatico** (Machine Learning) es un paradigma donde los algoritmos *aprenden patrones* a partir de datos, en lugar de ser programados explicitamente con reglas.





Tipos de Aprendizaje

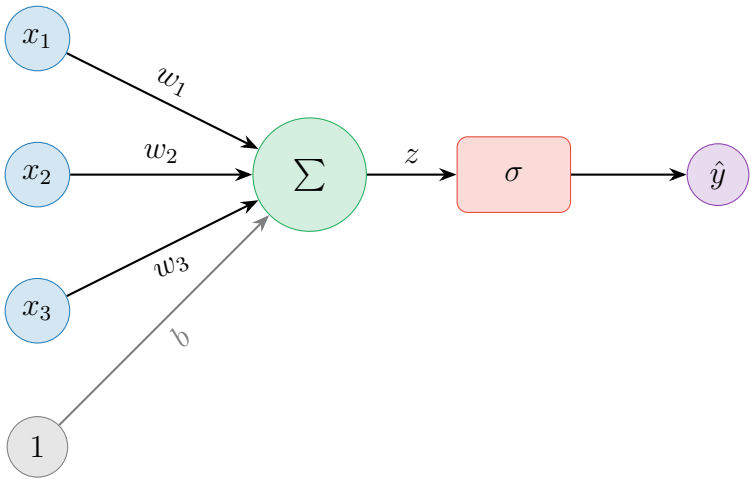
- **Supervisado:** Tenemos datos etiquetados (X, y). El modelo aprende a predecir y dado X.
- **No supervisado:** Solo tenemos X. El modelo encuentra patrones (clustering, reduccion de dimensionalidad).
- **Por refuerzo:** El modelo aprende mediante prueba y error, recibiendo recompensas.

En la Practica:

Proyecto HAR Nuestro proyecto es **aprendizaje supervisado**: tenemos datos de sensores (X) y etiquetas de actividad (y = WALKING, SITTING, etc.). El modelo aprendera a clasificar nuevas lecturas.

2. El Perceptron: La Neurona Artificial

El **perceptron** es la unidad basica de las redes neuronales, inspirado en la neurona biologica.



$$z = w_1x_1 + w_2x_2 + w_3x_3 + b \quad \hat{y} = \sigma(z)$$
$$z = \mathbf{w}^T\mathbf{x} + b$$

Componentes del Perceptron

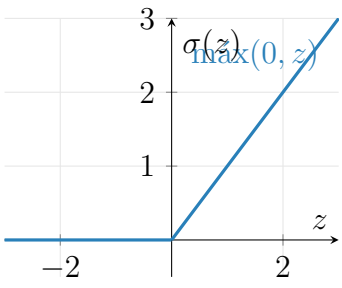
- **Entradas** $(x_1, x_2, \ldots, x_n)$: Valores de las caracteristicas.
- **Pesos** $(w_1, w_2, \ldots, w_n)$: Importancia de cada entrada (parametros aprendibles).
- **Bias** (b): Termino independiente que desplaza la frontera de decision.
- **Suma ponderada**: $z = \sum_{i=1}^{n} w_i x_i + b = \mathbf{w}^T\mathbf{x} + b$
- **Funcion de activacion** (σ): Introduce no-linealidad.

3. Funciones de Activacion

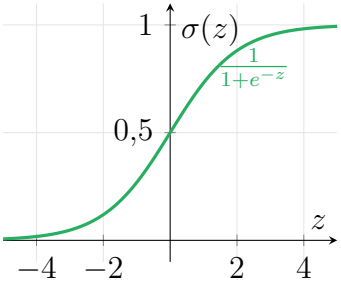
Las funciones de activacion son **cruciales** porque introducen **no-linealidad**, permitiendo a la red aprender patrones complejos.

3.1. Funciones Comunes

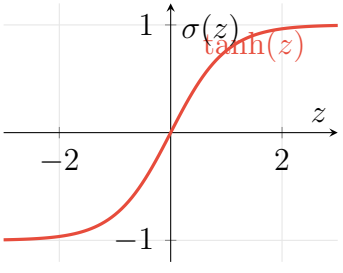
ReLU



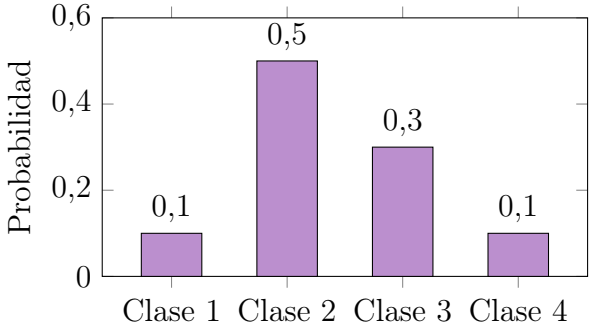
Sigmoid



Tanh



Softmax (para clasificacion multiclase)



$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Suma de todas = 1

Importante

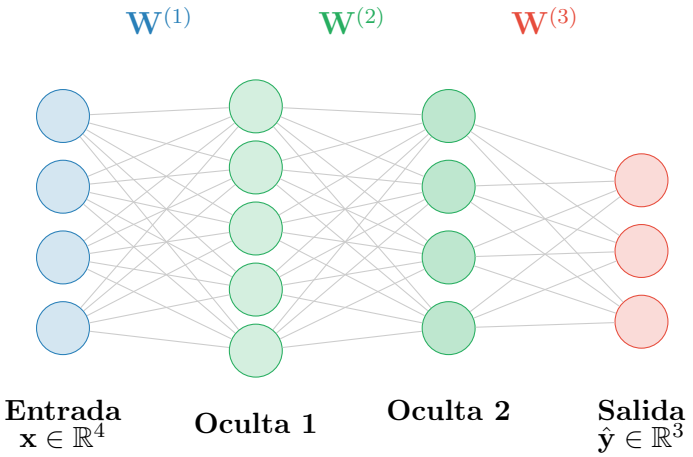
- **ReLU**: Usada en capas ocultas (rapida, evita vanishing gradient).
- **Sigmoid**: Salida binaria (0 a 1), usada en clasificacion binaria.
- **Softmax**: Salida multiclase (probabilidades que suman 1).

En la Practica:

HAR - 6 Clases Para clasificar 6 actividades, usaremos **Softmax** en la capa de salida. La red producira 6 probabilidades (una por clase) que suman 1.

4. Redes Neuronales Multicapa (MLP)

Un **Perceptron Multicapa** (MLP) apila multiples capas de neuronas para aprender representaciones cada vez mas abstractas.



4.1. Forward Pass (Propagacion hacia adelante)

Forward Pass

El calculo de la salida dado una entrada $\mathbf{x}$:

$$\begin{aligned}
\mathbf{z}^{(1)} &= \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)} && \text{(combinacion lineal)} \\
\mathbf{a}^{(1)} &= \sigma(\mathbf{z}^{(1)}) && \text{(activacion, ej. ReLU)} \\
\mathbf{z}^{(2)} &= \mathbf{W}^{(2)}\mathbf{a}^{(1)} + \mathbf{b}^{(2)} \\
\mathbf{a}^{(2)} &= \sigma(\mathbf{z}^{(2)}) \\
\mathbf{z}^{(3)} &= \mathbf{W}^{(3)}\mathbf{a}^{(2)} + \mathbf{b}^{(3)} \\
\hat{\mathbf{y}} &= \text{softmax}(\mathbf{z}^{(3)}) && \text{(probabilidades de clase)}
\end{aligned}$$

Intuicion:

Por que capas ocultas? Cada capa aprende **representaciones** mas abstractas:

- **Capa 1**: Detecta patrones simples (bordes, picos en senales).
- **Capa 2**: Combina patrones simples (formas, secuencias).
- **Capa 3**: Reconoce conceptos de alto nivel (objetos, actividades).

5. Funcion de Perdida (Loss Function)

La **funcion de perdida** mide que tan lejos estan las predicciones del modelo de los valores reales. El objetivo del entrenamiento es **minimizar** esta funcion.

5.1. Cross-Entropy para Clasificacion

Cross-Entropy Loss

Para clasificacion multiclase con K clases:

$$\mathcal{L} = -\sum_{i=1}^{K} y_i \log(\hat{y}_i)$$

Donde:

- y_i: Etiqueta real (one-hot: 1 si es la clase correcta, 0 si no).
- $\hat{y}_i$: Probabilidad predicha para la clase i.

Ejemplo:

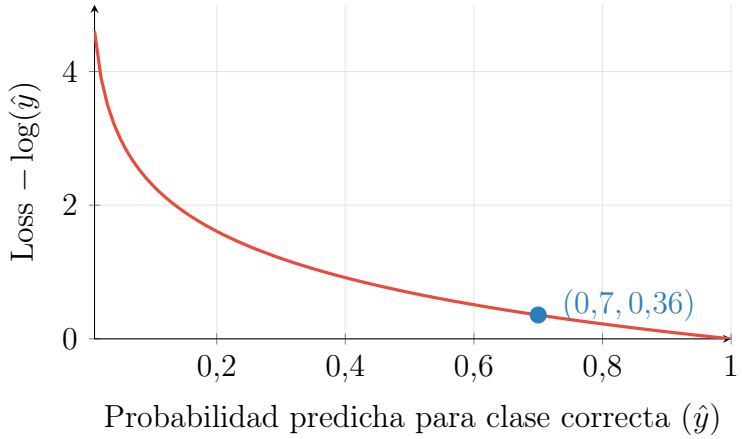
Cross-Entropy Supongamos que la clase real es "WALKING" (clase 1 de 3):

- Etiqueta one-hot: $\mathbf{y} = [1, 0, 0]$

- Prediccion del modelo: $\hat{\mathbf{y}} = [0{,}7, 0{,}2, 0{,}1]$
- Loss: $\mathcal{L} = -[1 \cdot \log(0{,}7) + 0 \cdot \log(0{,}2) + 0 \cdot \log(0{,}1)]$
- $\mathcal{L} = -\log(0{,}7) \approx 0{,}357$

Si la prediccion fuera perfecta ($\hat{\mathbf{y}} = [1, 0, 0]$):

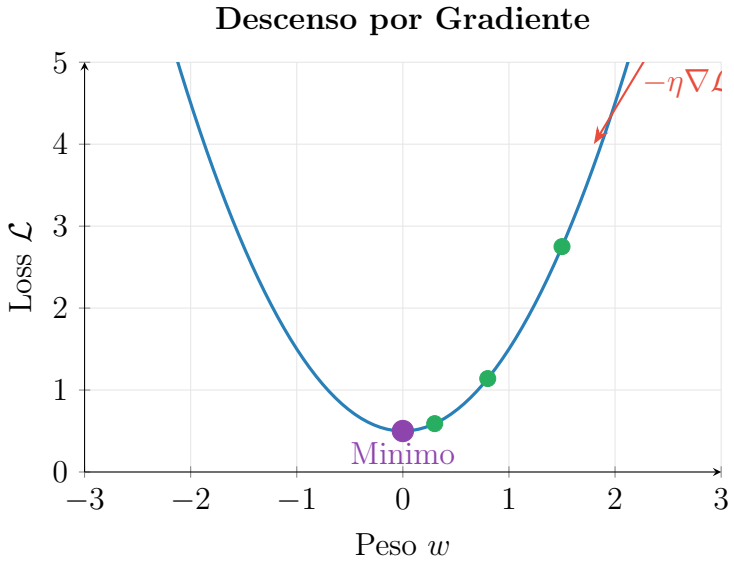
- $\mathcal{L} = -\log(1) = 0$ (perdida minima)



6. Backpropagation y Gradiente Descendente

Backpropagation es el algoritmo que calcula como ajustar los pesos para reducir la perdida. Usa la **regla de la cadena** del calculo.

6.1. Intuicion del Gradiente



Regla de Actualizacion

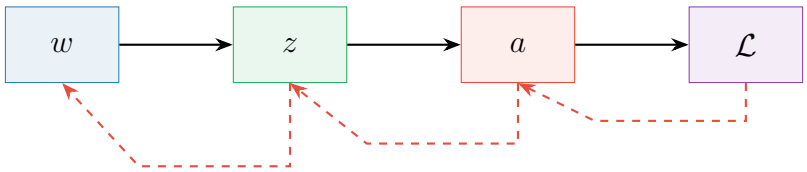
$$w_{\text{nuevo}} = w_{\text{actual}} - \eta \frac{\partial \mathcal{L}}{\partial w}$$

Donde:

- η: Tasa de aprendizaje (learning rate).
- $\frac{\partial \mathcal{L}}{\partial w}$: Gradiente (direccion de mayor aumento de la perdida).

6.2. Regla de la Cadena

Para calcular $\frac{\partial \mathcal{L}}{\partial w}$ en capas profundas, aplicamos la **regla de la cadena**:



$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial w}$$

Importante

La **backpropagation** propaga los gradientes *hacia atras* desde la salida hasta la entrada, calculando cuanto contribuye cada peso a la perdida total.

7. Optimizadores

Los **optimizadores** son algoritmos que mejoran el gradiente descendente basico.

7.1. SGD (Stochastic Gradient Descent)

SGD con Momentum

$$v_t = \beta v_{t-1} + \eta \nabla \mathcal{L}(w_t)$$
$$w_{t+1} = w_t - v_t$$

El **momentum** ($\beta \approx 0{,}9$) acumula velocidad, ayudando a superar minimos locales.

7.2. Adam (Adaptive Moment Estimation)

Adam

Combina momentum con tasas de aprendizaje adaptativas:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla\mathcal{L} \qquad \text{(primer momento)}$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2)(\nabla\mathcal{L})^2 \qquad \text{(segundo momento)}$$
$$w_{t+1} = w_t - \eta \frac{m_t}{\sqrt{v_t} + \epsilon}$$

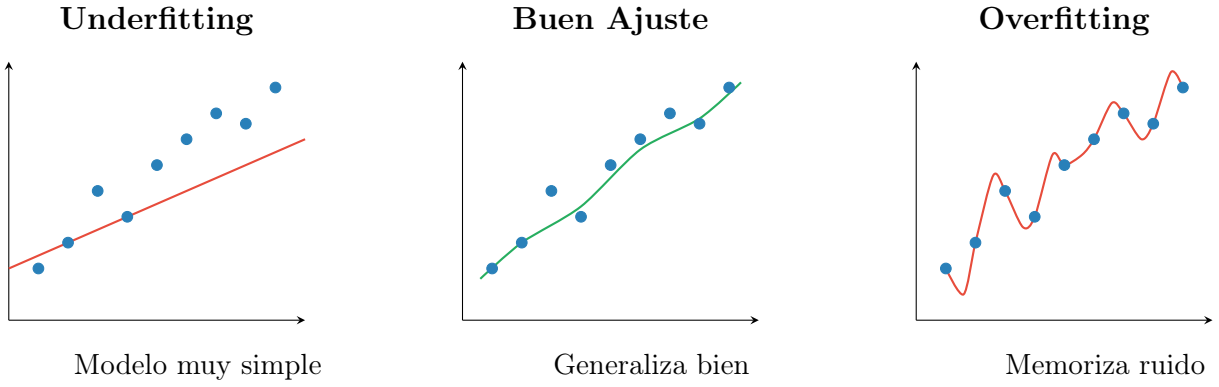
Valores tipicos: $\beta_1 = 0{,}9$, $\beta_2 = 0{,}999$, $\epsilon = 10^{-8}$.

En la Practica:

Cual usar? **Adam** es el optimizador mas popular por su robustez. Es una excelente opcion por defecto para la mayoria de problemas, incluyendo HAR.

8. Regularizacion: Evitando el Sobreajuste

El **sobreajuste** (overfitting) ocurre cuando el modelo memoriza los datos de entrenamiento pero no generaliza a datos nuevos.



8.1. Tecnicas de Regularizacion

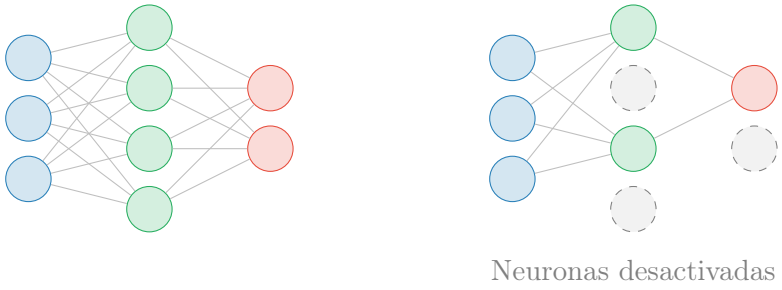
Dropout

Durante el entrenamiento, "apaga" aleatoriamente un porcentaje de neuronas en cada iteracion:

$$h_i = \begin{cases} 0 & \text{con probabilidad } p \\ \frac{a_i}{1-p} & \text{con probabilidad } 1-p \end{cases}$$

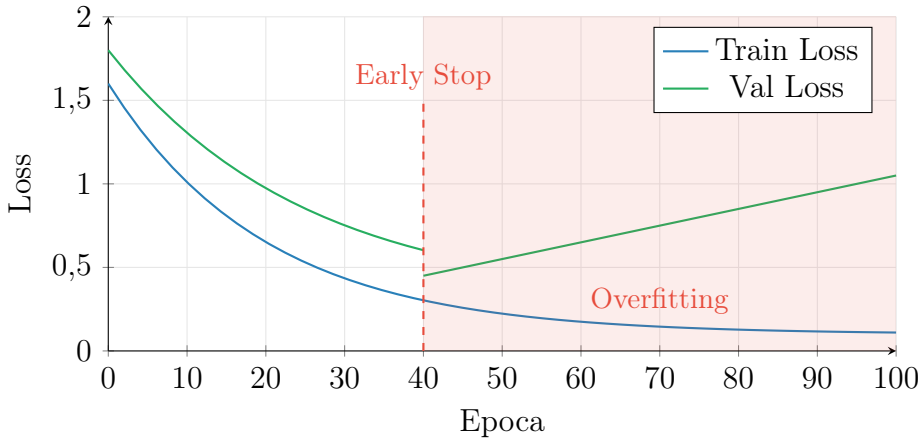
Esto fuerza a la red a no depender de neuronas especificas.

Sin Dropout **Con Dropout (p=0.5)**



Early Stopping

Detener el entrenamiento cuando la **perdida de validacion** deja de mejorar:



9. Redes Neuronales Convolucionales (CNN)

Las **CNN** son arquitecturas especializadas para datos con estructura espacial o temporal (imagenes, senales).

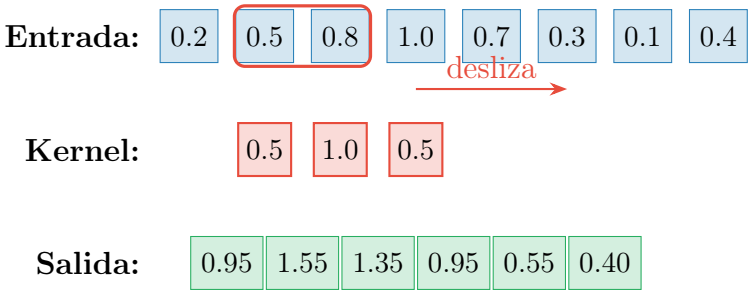
9.1. Convolucion 1D (para senales)

Convolucion 1D

Un **filtro** (kernel) se desliza sobre la senal, calculando productos punto:

$$(x * w)[i] = \sum_{k=0}^{K-1} x[i+k] \cdot w[k]$$

Donde K es el tamano del kernel.



Ejemplo (posicion 0):
$0{,}2 \times 0{,}5 + 0{,}5 \times 1{,}0 + 0{,}8 \times 0{,}5$
$= 0{,}1 + 0{,}5 + 0{,}4 = 1{,}0$

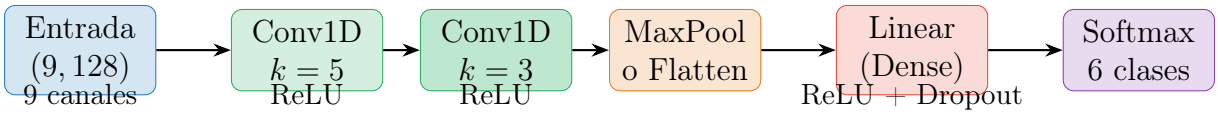
9.2. Por que CNN para HAR?

En la Practica:

Senales de Sensores Los datos HAR son **series temporales** de 128 timesteps. La convolucion 1D:

- Detecta **patrones locales** en el tiempo (picos, valles, ritmos).
- Es **invariante a la posicion**: reconoce el mismo patron sin importar cuando ocurra.
- Reduce parametros vs MLP al compartir pesos en el kernel.

9.3. Arquitectura CNN-1D + MLP



10. Entrenamiento en PyTorch

10.1. Estructura Tipica

Ciclo de Entrenamiento

1. **Forward pass**: Calcular predicciones $\hat{y} = \text{model}(x)$
2. **Calcular loss**: $\mathcal{L} = \text{criterion}(\hat{y}, y)$
3. **Backward pass**: `loss.backward()` (calcula gradientes)
4. **Actualizar pesos**: `optimizer.step()`
5. **Limpiar gradientes**: `optimizer.zero_grad()`

Ejemplo:

Codigo PyTorch

```
for epoch in range(num_epochs):
    for X_batch, y_batch in dataloader:
        # Forward
        outputs = model(X_batch)
        loss = criterion(outputs, y_batch)

        # Backward
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

10.2. Batches y Epocas

Terminologia

- **Batch**: Subconjunto de datos procesados juntos (ej. 32 muestras).
- **Epoca**: Una pasada completa por todo el dataset.
- **Iteracion**: Procesamiento de un batch.

Si tenemos 7352 muestras y batch_size=32:

$$\text{Iteraciones por epoca} = \lceil 7352/32 \rceil = 230$$

11. Metricas de Evaluacion

11.1. Accuracy

Accuracy

$$\text{Accuracy} = \frac{\text{Predicciones correctas}}{\text{Total de muestras}} = \frac{TP + TN}{TP + TN + FP + FN}$$

11.2. Matriz de Confusion

Real \ Predicho	WALK	UP	DOWN	SIT	STAND	LAY
WALK		3	2	0	0	0
UP	5		5	1	1	0
DOWN	3	4		2	1	0
SIT	0	1	1		12	1
STAND	0	0	1	10		1
LAY	0	0	0	1	1	

Importante

La diagonal muestra las predicciones correctas. Los valores fuera de la diagonal son errores. Nota como SIT y STAND se confunden entre si (actividades similares).

12. K-Fold Cross Validation

K-Fold

Divide los datos en K partes (folds). Entrena K veces, cada vez usando un fold diferente como validacion:

5-Fold Cross Validation

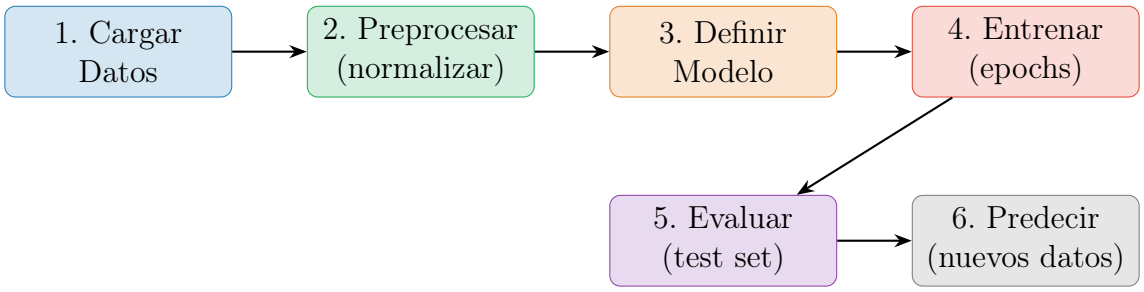
Fold 1:	Val	Train	Train	Train	Train
Fold 2:	Train	Val	Train	Train	Train
Fold 3:	Train	Train	Val	Train	Train
Fold 4:	Train	Train	Train	Val	Train
Fold 5:	Train	Train	Train	Train	Val

Resultado final:

Accuracy $= \frac{1}{K} \sum_{i=1}^{K} \text{Acc}_i$

Mas robusto que una sola division train/val

13. Resumen: Flujo Completo



14. Tabla Resumen de Conceptos

primary!20 Concepto	**Descripcion**	**En PyTorch / HAR**
Perceptron	Neurona basica: $z = \mathbf{w}^T\mathbf{x} + b$	`nn.Linear(in, out)`
ReLU	Activacion: máx$(0, z)$	`nn.ReLU()`
Softmax	Probabilidades multiclase	`nn.Softmax(dim=1)` (implicito en CrossEntropyLoss)
Cross-Entropy	Loss para clasificacion	`nn.CrossEntropyLoss()`
Backpropagation	Calculo de gradientes	`loss.backward()`
Adam	Optimizador adaptativo	`optim.Adam(params, lr=0.001)`
Conv1D	Convolucion para secuencias	`nn.Conv1d(in_ch, out_ch, kernel)`
Dropout	Regularizacion	`nn.Dropout(p=0.5)`
Batch	Subconjunto de datos	`DataLoader(batch_size=32)`
Early Stopping	Detener si val_loss no mejora	Implementar manualmente

En la Practica:

Siguiente Paso Con estos fundamentos, estas listo para implementar tu modelo CNN-1D + MLP para clasificar las 6 actividades del dataset HAR. El notebook `resolucion.ipynb` te espera.