# Curso Redes Neuronales

## Entornos de desarrollo y flujo de entrenamiento

Sergio Barrachina  Nacho Mestre

January 15, 2025

# Índice

# ¿Qué entorno queremos utilizar?

TensorFlow

- Google
- Popularidad
- Facilidad para empezar

PyTorch

- Facebook
- Mayor flexibilidad
- Menor abstracción
- Interfaz de NumPy

PyDTNN

- UJI
- Muy útil para obligarte a aprender (no incluye AutoGrad)
- Personalizar absolutamente todo
- Incluye poca variedad de capas

# Configuración del entorno

1. Instalar `uv` siguiendo las instrucciones de https://github.com/astral-sh/uv
2. Abrir un terminal (`PowerShell` en Windows)
3. Crear un directorio y entrar en él ▶ `mkdir curso_ia` ▶ `cd curso_ia`
4. Instalar Python y crear un entorno:
   - `uv python install 3.10`
   - `uv venv --python 3.10`
5. Ejecutar el comando indicado por `Activate with:`
   (En Windows, si los comandos están desactivados, como administrador):
   - `Set-ExecutionPolicy RemoteSigned`
6. Instalar los módulos necesarios:
   - `uv pip install notebook torch torchvision matplotlib numpy pandas`

# Configuración del entorno

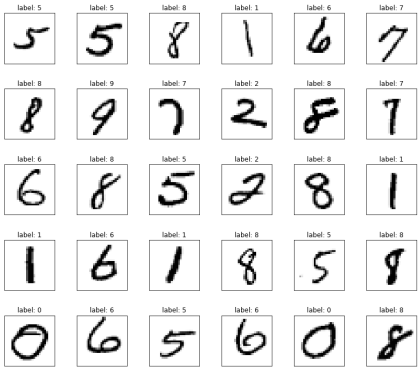
1. Lanzar Jupyter ▶ `jupyter notebook`
   (Si no se abriera automáticamente un navegador, abrir uno e introducir una de las URL mostradas)
2. Crear un nuevo cuaderno de Python ▶ New ▶ Python 3
3. En el nuevo cuaderno, introducir lo siguiente en la primera celda:

```
import torch
import matplotlib.pyplot as plt
data = torch.randn(1000)
plt.hist(data.numpy(), bins=30, alpha=0.7, color='darkblue')
plt.title('Distribution of Tensor Values')
plt.show()
```

4. Ejecutar la celda ▶ mayúsculas + enter

# Conjunto de datos

- Colección de ejemplos del problema que queremos que resuelva la red neuronal.
- Se utiliza para entrenar la red, y para validar la calidad del entrenamiento.
- Es la única fuente de información.

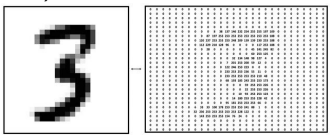



MNIST

Reconocimiento de dígitos manuscritos

# Conjunto de datos

- Entradas y etiquetas (ground truth)



Salida esperada: 3

- Preprocesado
- División en entrenamiento (80%), validación (10%) y test (10%)
- Equilibrado
- Aumento de datos

# Conjunto de datos

En la práctica, recopilar y procesar los conjuntos de datos es una de las tareas más costosas.

En investigación, conjuntos de datos mundialmente reconocidos:

- MNIST, CIFAR-10 y CIFAR-100 - Pruebas con reconocimiento de imágenes
- Imagenet - Más importante en reconocimiento de imágenes
- BookCorpus + Wikipedia - Gran cantidad de texto
- GLUE - Conjunto de tareas de procesamiento de lenguaje natural (NLP).

# Implementación

Con la biblioteca TorchVision de PyTorch

https://pytorch.org/vision/0.20/datasets

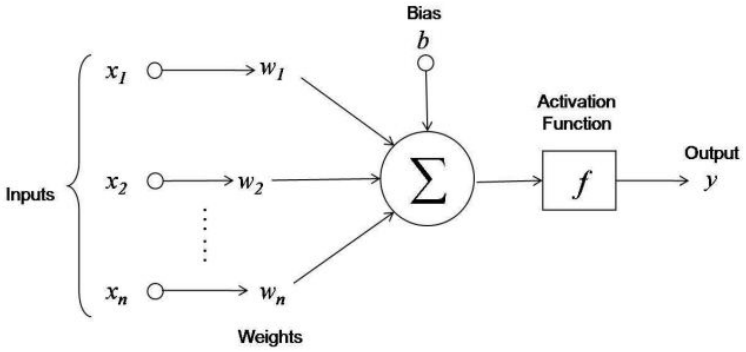
```
1 import torch
2 import torch.nn as nn
3 import torchvision
4
5 train_set = torchvision.datasets.MNIST(root='./data', train=True,
      download=True, transform=torchvision.transforms.ToTensor())
6 train_loader = torch.utils.data.DataLoader(train_set, batch_size=64)
```

# Modelo de red neuronal

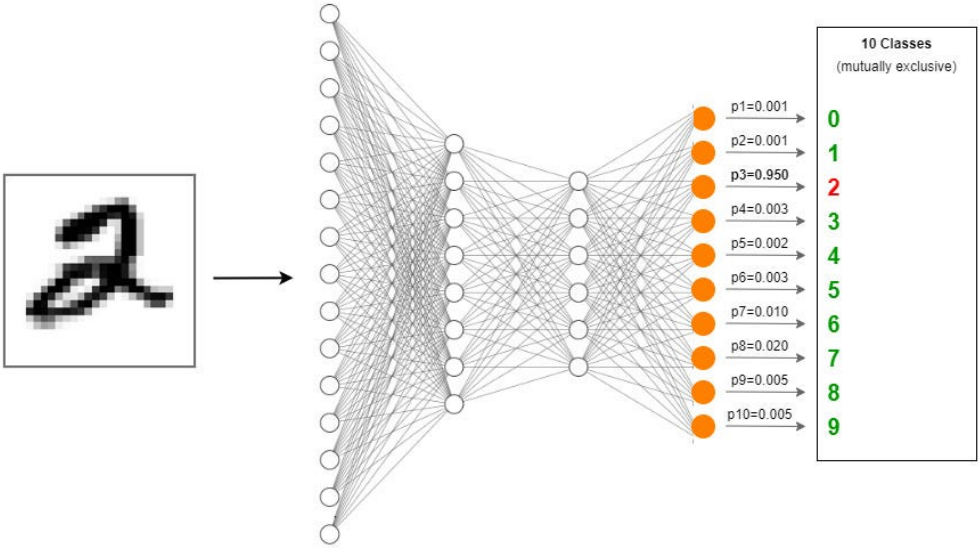
- Elegir el tipo modelo en función de la tarea
  - Multilayer perceptron (básico)
  - Modelo convolucional (imágenes) ← Hoy
  - Transformer (lenguaje)

- Crear del modelo
  - Modelos desde cero ← Hoy
  - Modelos prediseñados
  - Modelos pre-entrenados

# Neurona individual



# Multilayer Perceptron



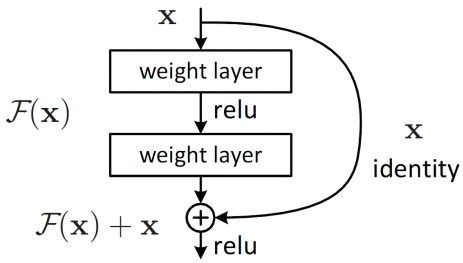
# Implementación

```python
import torch.nn as nn
import torch.nn.functional as F

class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.layer1 = nn.Linear(784, 12, bias=True)
        self.layer2 = nn.Linear(12, 12, bias=True)
        self.layer3 = nn.Linear(12, 10, bias=True)

    def forward(self, x):
        x = x.view(x.shape[0], -1)
        x = self.layer1(x)
        x = F.relu(x)
        x = self.layer2(x)
        x = F.relu(x)
        x = self.layer3(x)
        x = F.softmax(x)
        return x
```

# Conexión residual




```
def forward(self, x):
    residuo = x
    x = self.layer1(x)
    x = F.relu(x)
    x = self.layer2(x)
    x = x + residuo
    x = F.relu(x)
    return x
```

# Capas de normalización

**Batch Normalization**

1. Calcula la **media** y **varianza** de los valores para cada entrada.
2. Normaliza:
$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$
3. Escala y desplaza:
$$y_i = \gamma \hat{x}_i + \beta$$

Donde:

- $\mu$, $\sigma^2$: media y varianza
- $\gamma, \beta$: parámetros entrenables
- $\epsilon$: parámetro de estabilidad numérica

# Softmax

**Fórmula:**

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$$

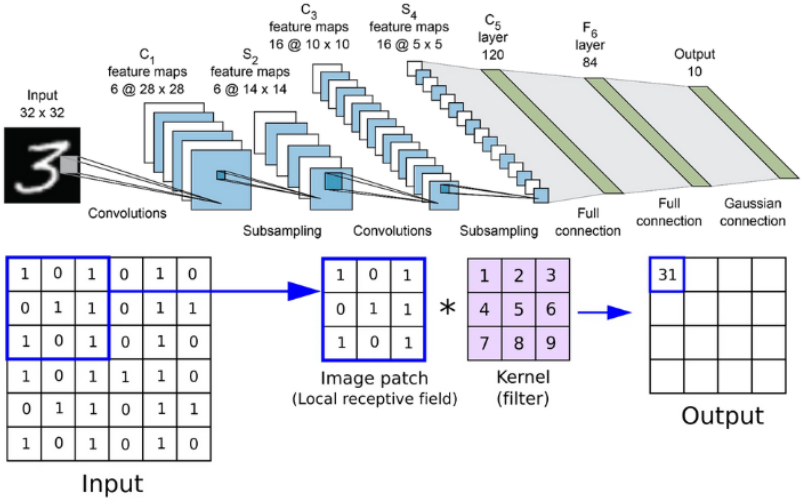
Donde:

- $x_i$: Entrada para la clase $i$.
- $N$: Número total de clases.

**Ejemplo:**

- Entrada: $x = [2.0, 1.0, 0.1]$
- Salida: $\text{softmax}(x) = [0.71, 0.26, 0.03]$

# Convolución



# Proceso completo

- Inicialización (Automático)
- Entrenamiento
  1. Cargar muestras (por lotes)
  2. Pasar las muestras por la red y obtener una predicción $\rightarrow$ Forward-Pass
  3. Comparar con el resultado esperado $\rightarrow$ Función de pérdida
  4. Calcular el gradiente de cada parámetro $\rightarrow$ Backward-Pass (AutoGrad)
  5. Actualizar los parámetros $\rightarrow$ Optimizador
  6. Volver al punto 1
- Inferencia

# Otras funciones

**Funciones de pérdida**

https://pytorch.org/docs/stable/nn#loss-functions

**Optimizadores**

https://pytorch.org/docs/stable/optim

# Implementación

```python
import torch.optim as optim

model = MLP()                           # Inicialización
criterion = nn.CrossEntropyLoss()       # Función de pérdida
optimizer = optim.SGD(model.parameters(), lr=0.01) # Optimizador

n_epochs = 10                           # Número de epocas a entrenar
for epoch in range(n_epochs):
    model.train()
    for data, target in train_loader:
        optimizer.zero_grad()           # Reinicia el optimizador
        output = model(data)            # Forward-Pass
        loss = criterion(output, target)
        loss.backward()                 # Backward-Pass
        optimizer.step()                # Actualiza los pesos
```

# Práctica

https://github.com/jmiravet/curso_ia.git