Ligera introducción a Deep Learning (Redes neuronales)

1. Introducción a Deep Learning y Redes Neuronales

Deep Learning es una rama del Machine Learning que utiliza redes neuronales profundas (con múltiples capas) para modelar y resolver problemas complejos. La idea central es que, mediante el entrenamiento, la red aprende representaciones jerárquicas de los datos.

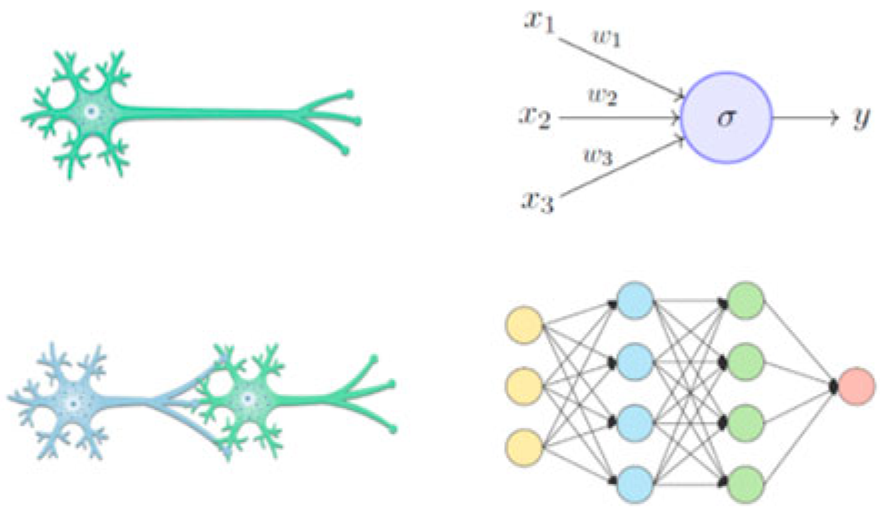
1.1 ¿Qué es una Red Neuronal?

Una red neuronal es un modelo computacional inspirado en el cerebro humano, formado por neuronas artificiales conectadas entre sí en diferentes capas.

- **Capa de entrada:** Recibe los datos.
- **Capas ocultas:** Realizan transformaciones intermedias.
- **Capa de salida:** Produce la predicción o clasificación final.

Cada **neurona** realiza una operación simple:

1. **Ponderación de entradas:** Cada entrada x_i se multiplica por un peso w_i.
2. **Suma y sesgo:** Se suma el resultado más un sesgo b.
3. **Función de activación:** Se aplica una función f que introduce no linealidad.

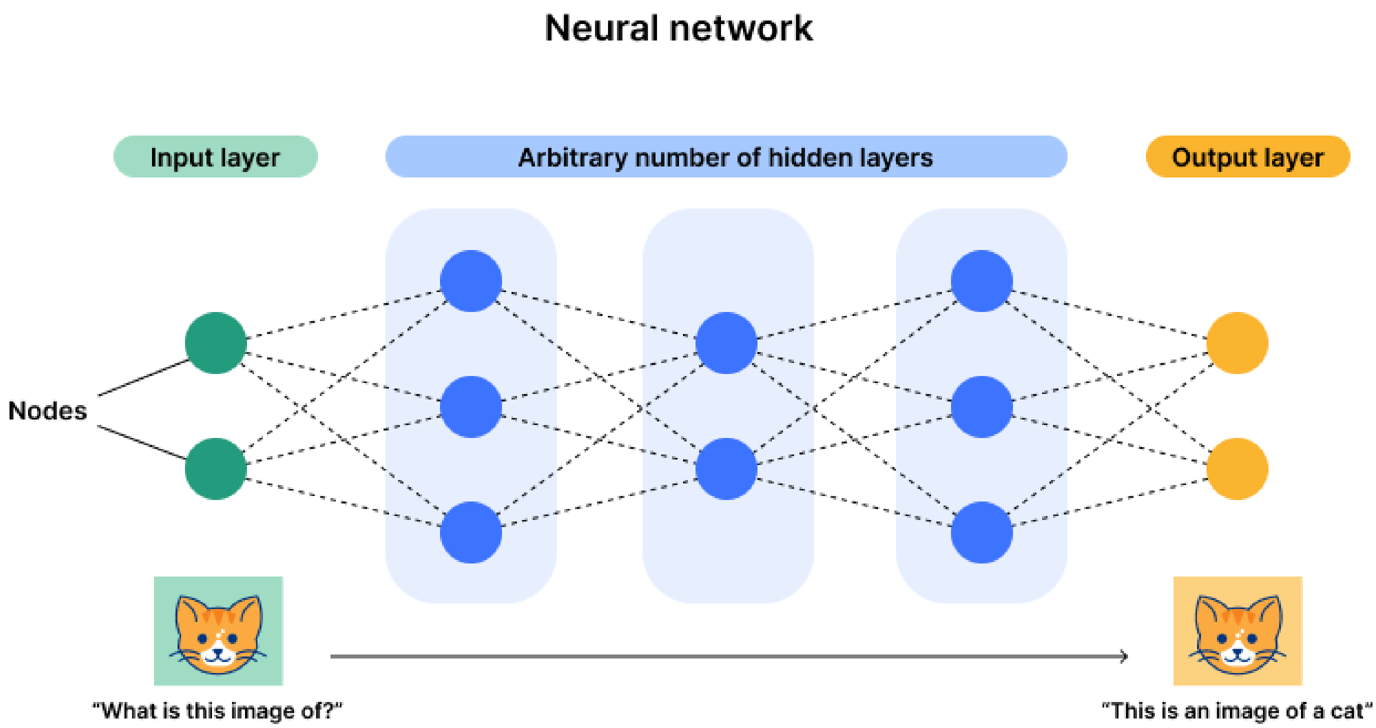


La fórmula básica de una neurona es:

$$z = \sum_{i=1}^{n} w_i x_i + b \quad \text{y} \quad a = f(z)$$

Donde:

- x_i: entradas (características del dato).
- w_i: pesos asociados a cada entrada.
- b: sesgo (bias).
- z: suma ponderada más el sesgo.
- f: función de activación.
- a: salida de la neurona (activación).



2. Funciones de Activación

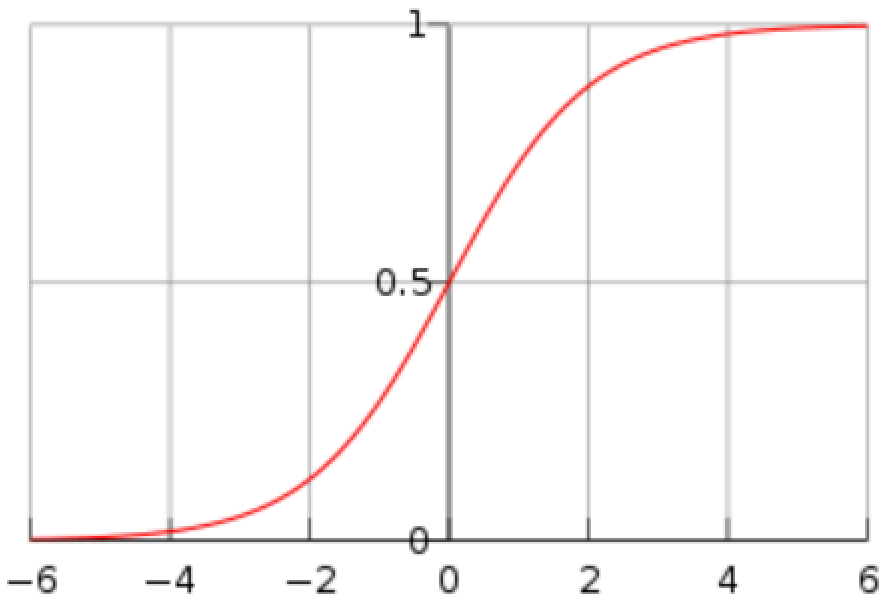
Las funciones de activación introducen no linealidad en el modelo, permitiendo aprender relaciones complejas. Algunas comunes son:

2.1 Función Sigmoide

$$f(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$

- **Propiedades:**
  - Rango: $(0, 1)$.
  - Útil para problemas de clasificación binaria.
- **Ejemplo:**
  Si $z = 0$, entonces $\sigma(0) = \frac{1}{1+1} = 0.5$.

$$f(x) = \frac{1}{1 + e^{-x}}$$

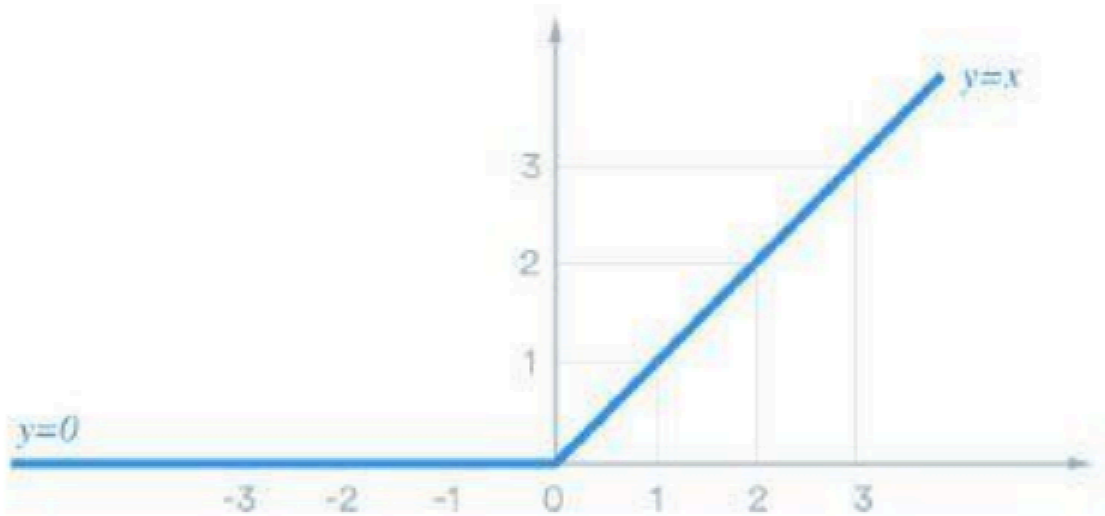


2.2 Función ReLU (Rectified Linear Unit)

$$f(z) = \max(0, z)$$

- **Propiedades:**
  - Rango: $[0, \infty)$.
  - Computacionalmente eficiente y ayuda a mitigar el problema del gradiente desaparecido.
- **Ejemplo:**
  Si $z = -3$, $\mathrm{ReLU}(-3) = 0$; si $z = 2$, $\mathrm{ReLU}(2) = 2$.

$$f(x) = max(0, x)$$



2.3 Función Tanh

$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

- **Propiedades:**
  - Rango: $(-1, 1)$.
  - Centrada en cero, lo que puede mejorar la convergencia.

3. Entrenamiento de Redes Neuronales

El objetivo del entrenamiento es ajustar los pesos w y sesgos b para minimizar una función de pérdida L que mide el error entre las predicciones y los valores reales.

3.1 Función de Pérdida

Un ejemplo común para clasificación es la **entropía cruzada**. Para un ejemplo binario:

$$L(y, \hat{y}) = -\left[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})\right]$$

Donde:

- y: etiqueta real (0 o 1).
- $\hat{y}$: salida de la red (predicción de probabilidad).

Para problemas de regresión se usa, por ejemplo, el **error cuadrático medio**:

$$L(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$$

3.2 Algoritmo de Retropropagación (Backpropagation)

La retropropagación es el método para calcular el gradiente de la función de pérdida respecto a cada peso, usando la regla de la cadena.
La actualización de los parámetros se realiza mediante **descenso del gradiente**:

$$w := w - \alpha \frac{\partial L}{\partial w}$$

$$b := b - \alpha \frac{\partial L}{\partial b}$$

Donde:

- α es la tasa de aprendizaje (learning rate).
- $\frac{\partial L}{\partial w}$ es la derivada parcial de la pérdida respecto a w.

4. Ejemplo Completo en Python

A continuación, implementaremos una red neuronal simple desde cero usando NumPy. Se tratará un problema de clasificación binaria (por ejemplo, distinguir entre dos clases).

4.1 Descripción del Ejemplo

La red tendrá:

- Una **capa de entrada** con n características.
- Una **capa oculta** con m neuronas y función de activación ReLU.
- Una **capa de salida** con 1 neurona y función de activación sigmoide para obtener una probabilidad.

Usaremos el error cuadrático medio (aunque en clasificación se suele usar entropía cruzada, para simplificar usaremos MSE en este ejemplo).

4.2 Código en Python

```python
import numpy as np

# Funciones de activación y sus derivadas
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def sigmoid_deriv(z):
    return sigmoid(z) * (1 - sigmoid(z))

def relu(z):
    return np.maximum(0, z)

def relu_deriv(z):
    return np.where(z > 0, 1, 0)

# Función de pérdida (Error Cuadrático Medio)
def mse_loss(y_true, y_pred):
    return np.mean(0.5 * (y_true - y_pred) ** 2)

# Parámetros de la red
input_size = 2    # Número de características de entrada
hidden_size = 3   # Número de neuronas en la capa oculta
output_size = 1   # Una neurona en la capa de salida (clasificación binaria)
learning_rate = 0.1
epochs = 10000

# Inicialización de pesos y sesgos
np.random.seed(42)  # Para reproducibilidad
W1 = np.random.randn(input_size, hidden_size)
b1 = np.zeros((1, hidden_size))
W2 = np.random.randn(hidden_size, output_size)
b2 = np.zeros((1, output_size))

# Datos de ejemplo: Entradas (X) y etiquetas (y)
# Usamos un conjunto muy simple para clasificación OR lógica
X = np.array([[0,0],
              [0,1],
              [1,0],
              [1,1]])
y = np.array([[0],
              [1],
```

```python
              [1],
              [1]])

# Entrenamiento de la red
for epoch in range(epochs):
    # Forward propagation
    z1 = np.dot(X, W1) + b1          # Cálculo de la entrada de la capa oculta
    a1 = relu(z1)                    # Salida de la capa oculta con activación ReLU
    z2 = np.dot(a1, W2) + b2          # Cálculo de la entrada de la capa de salida
    a2 = sigmoid(z2)                 # Salida de la capa de salida con activación sigmoide

    # Cálculo de la pérdida
    loss = mse_loss(y, a2)

    # Backpropagation
    # Derivada de la pérdida con respecto a a2
    dL_da2 = a2 - y  # Para MSE: (a2 - y)

    # Derivadas de la capa de salida
    dL_dz2 = dL_da2 * sigmoid_deriv(z2)
    dL_dW2 = np.dot(a1.T, dL_dz2)
    dL_db2 = np.sum(dL_dz2, axis=0, keepdims=True)

    # Derivadas de la capa oculta
    dL_da1 = np.dot(dL_dz2, W2.T)
    dL_dz1 = dL_da1 * relu_deriv(z1)
    dL_dW1 = np.dot(X.T, dL_dz1)
    dL_db1 = np.sum(dL_dz1, axis=0, keepdims=True)

    # Actualización de pesos y sesgos (descenso del gradiente)
    W2 -= learning_rate * dL_dW2
    b2 -= learning_rate * dL_db2
    W1 -= learning_rate * dL_dW1
    b1 -= learning_rate * dL_db1

    # Imprimir la pérdida cada 1000 épocas
    if epoch % 1000 == 0:
        print(f"Epoch {epoch}, Loss: {loss:.4f}")

# Evaluación final
print("\nResultados finales:")
print("Predicciones:")
print(a2)
```

```python
print("Etiquetas reales:")
print(y)
```

5. Explicación Detallada del Ejemplo

5.1 Inicialización

- **Pesos y sesgos:**
  Se inicializan con valores aleatorios y ceros, respectivamente.
  - $W1$ es una matriz de dimensiones $(input\_size, hidden\_size)$.
  - $b1$ es un vector de dimensión $(1, hidden\_size)$.
  - $W2$ es una matriz de dimensiones $(hidden\_size, output\_size)$.
  - $b2$ es un vector de dimensión $(1, output\_size)$.

5.2 Propagación hacia Adelante (Forward Propagation)

1. **Capa Oculta:**
   - Calculamos $z_1 = X \cdot W1 + b1$.
     Aquí, cada fila de X se multiplica por $W1$ y se le suma $b1$.
   - Aplicamos la función ReLU: $a_1 = \mathrm{ReLU}(z_1)$.
2. **Capa de Salida:**
   - Calculamos $z_2 = a_1 \cdot W2 + b2$.
   - Aplicamos la función sigmoide: $a_2 = \sigma(z_2)$.
     La salida a_2 representa la probabilidad de la clase 1.

5.3 Cálculo de la Pérdida

Utilizamos el error cuadrático medio (MSE):

$$L = \frac{1}{2N} \sum (y - a_2)^2$$

Donde N es el número de ejemplos. Esto mide la diferencia entre la predicción a_2 y la etiqueta real y
.

5.4 Retropropagación (Backpropagation)

Se calcula el gradiente de la pérdida respecto a cada parámetro:

1. **Capa de salida:**
   - $\frac{\partial L}{\partial a_2} = a_2 - y$ (para MSE).
   - $\frac{\partial L}{\partial z_2} = (a_2 - y) \cdot \sigma'(z_2)$.
     Donde $\sigma'(z_2)$ es la derivada de la función sigmoide.
   - Gradientes para $W2$ y $b2$ se calculan multiplicando con la salida de la capa oculta a_1.
2. **Capa oculta:**
   - Se propaga el error hacia atrás usando $W2$ y se multiplica por la derivada de ReLU: $\mathrm{ReLU}'(z_1)$.
   - Se calculan gradientes para $W1$ y $b1$.

5.5 Actualización de Parámetros

Se actualizan utilizando la regla de descenso del gradiente:

$$\theta := \theta - \alpha \frac{\partial L}{\partial \theta}$$

Donde θ representa los pesos o sesgos y α es la tasa de aprendizaje.