Ficha Técnica Avanzada: Redes Neuronales Artificiales (MLP)

Jordi Pozo
I.E.S. Ribera de Castilla

Curso 2025/26

1 Concepto

Las **Redes Neuronales Artificiales (RNA)** son modelos computacionales inspirados vagamente en el cerebro humano. Están compuestas por unidades interconectadas llamadas **neuronas artificiales** (o nodos) organizadas en capas.

En el contexto de SAA y MIA, nos centramos principalmente en el **Perceptrón Multicapa (MLP)**, que es una red neuronal "feedforward" (hacia adelante).

La estructura básica se compone de:

- **Capa de Entrada (Input Layer):** Recibe los datos originales (las características o *features*).
- **Capas Ocultas (Hidden Layers):** Una o más capas donde ocurre el procesamiento. Cada neurona aquí aplica una transformación lineal seguida de una función de activación no lineal.
- **Capa de Salida (Output Layer):** Produce la predicción final (clase o valor numérico).

[Aquí iría el diagrama de la arquitectura]

2 Mecanismo de Aprendizaje

El aprendizaje en una red neuronal es un proceso iterativo que consta de dos fases principales:

2.1 1. Propagación Hacia Adelante (Forward Propagation)

Los datos entran por la capa de entrada, se multiplican por unos **pesos (weights)**, se les suma un **sesgo (bias)** y pasan por una **función de activación**. Este proceso se repite capa por capa hasta obtener una salida.

$$y = f(\sum(w_i \cdot x_i) + b)$$

2.2 2. Retropropagación (Backpropagation)

La red compara su predicción con el valor real usando una **función de pérdida** (Loss Function). Luego, calcula el gradiente del error y lo "envía hacia atrás" para ajustar los pesos y sesgos, utilizando un optimizador (como el Descenso de Gradiente o Adam) para minimizar el error en la siguiente iteración.

3 Funciones de Activación Clave

Sin estas funciones, una red neuronal sería simplemente una regresión lineal gigante. Aportan la **no-linealidad**.

- **ReLU (Rectified Linear Unit):** $f(x) = max(0, x)$. Es la más usada en capas ocultas por su eficiencia y porque evita el problema del desvanecimiento del gradiente.
- **Sigmoide (Logistic):** Aplasta valores entre 0 y 1. Útil para probabilidad binaria.
- **Softmax:** Convierte un vector de números en probabilidades que suman 1. Se usa exclusivamente en la **capa de salida** para clasificación multiclase.

4 Ilustración del Modelo

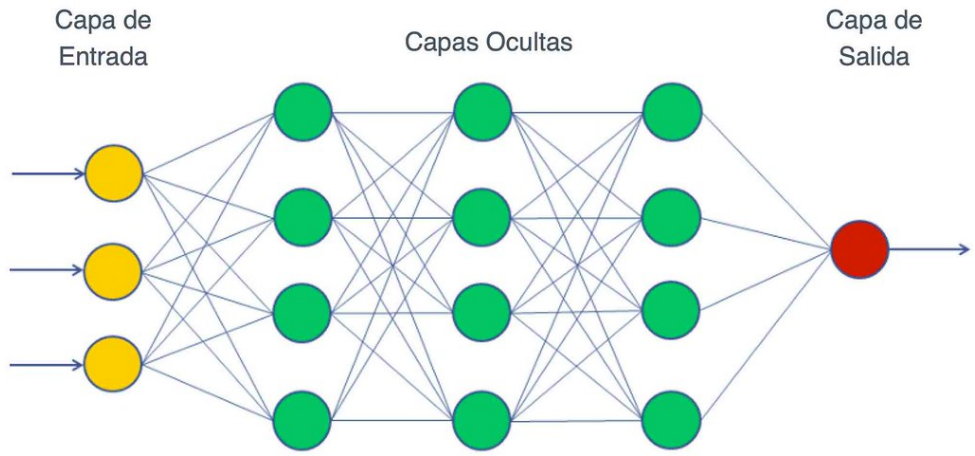

Figure 1: Esquema conceptual de un Perceptrón Multicapa (MLP).

5 ¡Importante! Escalado de Datos

Al igual que con SVM, el escalado de datos es **CRÍTICO** para las redes neuronales. Como el aprendizaje se basa en multiplicar entradas por pesos y calcular gradientes, si las características tienen escalas muy diferentes (ej. edad vs salario), el algoritmo de optimización (Adam/SGD) tardará mucho en converger o se quedará atascado en mínimos locales.

Regla: Siempre usar `StandardScaler` o `MinMaxScaler` antes de entrenar un MLP.

6 Hiperparámetros Clave (en `scikit-learn`)

En `scikit-learn`, usamos `MLPClassifier` y `MLPRegressor`.

- `hidden_layer_sizes`: (tupla). Define la arquitectura. Ej. `(100,)` es una capa oculta con 100 neuronas. `(50, 25)` son dos capas ocultas.
- `activation`: (string, default='relu'). La función de activación para las capas ocultas ('identity', 'logistic', 'tanh', 'relu').
- `solver`: (string, default='adam'). El algoritmo para optimizar los pesos. 'adam' funciona bien en la mayoría de casos. 'sgd' es descenso de gradiente estocástico.
- `alpha`: (float). Parámetro de penalización L2 (regularización) para evitar sobreajuste.

- `learning_rate_init`: (float). El tamaño del paso inicial para actualizar los pesos.
- `max_iter`: (int). Número máximo de épocas (pasadas completas por los datos).

7 Características Principales

7.1 Ventajas

- Capacidad para modelar relaciones no lineales extremadamente complejas.
- Funciona bien con grandes volúmenes de datos.
- Base del *Deep Learning* (procesamiento de imágenes, NLP, audio).

7.2 Desventajas

- **Caja Negra:** Muy difícil de interpretar cómo toma las decisiones (a diferencia de los árboles).
- **Coste Computacional:** Requiere mucha CPU/GPU y tiempo de entrenamiento.
- **Propensión al Sobreajuste:** Requiere mucho ajuste de hiperparámetros y regularización.
- Requiere gran cantidad de datos para superar a modelos clásicos.

8 Ejemplos con Python y Scikit-learn

8.1 Ejemplo 1: Clasificación (MLP) - Dataset Digits

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score

# 1. Cargar datos (Digitos 8x8 pixels)
digits = load_digits()
X = digits.data
y = digits.target

# 2. Dividir
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# 3. ESCALAR (Obligatorio)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 4. Crear y entrenar
# Dos capas ocultas: una de 64 neuronas y otra de 32
mlp = MLPClassifier(hidden_layer_sizes=(64, 32),
activation='relu',
solver='adam',
max_iter=1000, # Aumentar si no converge
random_state=42)

mlp.fit(X_train_scaled, y_train)
```

```
29
30      # 5. Evaluar
31      y_pred = mlp.predict(X_test_scaled)
32      print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
33
34      # Ver curva de perdida (loss)
35      # print(mlp.loss_curve_)
36
```

Listing 1: Clasificacion de digitos manuscritos con MLPClassifier.

8.2 Ejemplo 2: Regresión (MLPRegressor)

```
1   from sklearn.datasets import fetch_california_housing
2   from sklearn.neural_network import MLPRegressor
3   from sklearn.preprocessing import StandardScaler
4   from sklearn.model_selection import train_test_split
5
6   # 1. Datos
7   housing = fetch_california_housing()
8   X, y = housing.data, housing.target
9
10  # 2. Split y Escalado
11  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=42)
12  scaler = StandardScaler()
13  X_train_s = scaler.fit_transform(X_train)
14  X_test_s = scaler.transform(X_test)
15
16  # 3. Modelo (Regresion)
17  # Capa oculta grande (100) y regularizacion alpha
18  regr = MLPRegressor(hidden_layer_sizes=(100, 50),
19  activation='relu',
20  solver='adam',
21  alpha=0.001,
22  max_iter=500,
23  random_state=42)
24
25  regr.fit(X_train_s, y_train)
26
27  # 4. Score (R2)
28  print(f"R2 Score: {regr.score(X_test_s, y_test):.4f}")
29
```

Listing 2: Regresion simple con Red Neuronal.