🧠 Tipo de red neuronal

1. Identifique el tipo:

Red Neuronal Feedforward (Densa con Embedding y Pooling Global).

2. Justificación:

Este modelo se emplea para **clasificación de texto**, donde la entrada son secuencias de palabras transformadas en vectores numéricos.
Se elige una arquitectura **feedforward** porque las oraciones del dataset no requieren mantener dependencias temporales largas como en una RNN.
Además, el uso de una **capa de embedding** permite representar palabras en un espacio semántico más compacto, y la **capa global average pooling** resume la información de toda la oración en un vector representativo.

⚙️ Características principales

1. Capas:

Estructura del modelo:

```
model.add(tf.layers.embedding({ inputDim: vocab.size + 1, outputDim: 16, inputLength: maxLen }));
model.add(tf.layers.globalAveragePooling1d());
model.add(tf.layers.dense({ units: 16, activation: 'relu' }));
model.add(tf.layers.dense({ units: labelNames.length, activation: 'softmax' }));
```

- **Capa de entrada:**
    - Recibe secuencias de texto con longitud fija (`maxLen`).
    - Cada palabra está representada por un índice entero del vocabulario.
- **Capas ocultas:**
    - **Embedding:** convierte cada palabra en un vector de 16 dimensiones (aprende relaciones entre palabras).

    - **GlobalAveragePooling1D:** reduce la secuencia a un vector promedio, manteniendo la información general del texto.
    - **Dense (16 neuronas, ReLU):** aprende combinaciones no lineales de las características generadas por las capas previas.
- **Capa de salida:**
    - `Dense(labelNames.length, activation='softmax')`
    - Una neurona por clase → produce una distribución de probabilidad para cada categoría textual.

---

2. Características de activación:

- **ReLU:** en la capa intermedia densa, mejora la velocidad de entrenamiento y evita el problema del gradiente desaparecido.
- **Softmax:** en la capa de salida, convierte los valores en probabilidades entre 0 y 1, ideales para clasificación multiclase.

---

3. Características arquitectónicas:

- **Capa Embedding:** reduce la dimensionalidad del texto y permite que el modelo aprenda relaciones semánticas entre palabras.
- **GlobalAveragePooling:** evita sobreajuste al no depender del orden exacto de las palabras, resumiendo su información global.
- **Estructura simple y eficiente:** ideal para datasets pequeños o medianos, con entrenamiento rápido y pocos parámetros.

🧩 Algoritmo de entrenamiento

1. Explique el algoritmo:

El modelo se entrena con el **optimizador Adam**, que combina el descenso de gradiente estocástico con momentos adaptativos.

Se utiliza la **función de pérdida categoricalCrossentropy**, adecuada para problemas de clasificación multiclase con salidas tipo *one-hot*.

2. Justificación:

Adam es ideal porque ajusta automáticamente la tasa de aprendizaje de cada parámetro, mejorando la convergencia en datasets textuales con vocabularios amplios.
Además, es robusto ante gradientes ruidosos y requiere poca configuración manual.

📈 Medición del rendimiento

1. Definir la medida:

- **Exactitud (Accuracy)**

2. Relevancia:

La precisión mide la proporción de textos correctamente clasificados entre el total.
Es la métrica más intuitiva y adecuada cuando las clases están equilibradas, como en este tipo de problemas de categorización textual.

🔍 Evaluación de la formación

1. Proceso de evaluación:

El entrenamiento se realiza con:

```
await model.fit(X, y, { epochs: 200, verbose: 1 });
```

Durante el entrenamiento, se evalúa la pérdida y la exactitud sobre los datos de entrenamiento.
Se podría extender para incluir un conjunto de validación (`validationSplit`) si se desea medir la generalización del modelo.

2. Criterios de efectividad:

- La pérdida (`loss`) disminuye gradualmente a lo largo de las épocas.

- La precisión (`accuracy`) aumenta de manera estable y supera un umbral aceptable (por ejemplo, 85 % o más).
- El modelo no muestra sobreajuste (es decir, la precisión no se estanca ni disminuye).

🧪 Técnicas de validación

1. Discutir técnicas:

En este script se usa **todo el conjunto de datos para entrenamiento**, pero podrían implementarse técnicas como:

- **Validación simple (hold-out):** dividir los datos en entrenamiento y validación.
- **Validación cruzada k-fold:** entrenar y evaluar en diferentes particiones para medir la estabilidad del modelo.

2. Importancia de la validación:

Estas técnicas son esenciales para:

- Comprobar la **capacidad de generalización** del modelo ante textos nuevos.
- Evitar **sobreajuste** al dataset de entrenamiento.