

IA

REDES NEURONALES CON TENSORFLOW

¿Qué es TensorFlow?

TensorFlow es una biblioteca de código abierto desarrollada por Google para construir y entrenar modelos de machine learning y deep learning. Está diseñada para trabajar con grandes cantidades de datos y modelos complejos de redes neuronales. Su nombre proviene del concepto de "tensores", que son estructuras de datos similares a los arrays multidimensionales.

TensorFlow ofrece:

- Operaciones matemáticas optimizadas.
- Soporte para entrenamiento en CPU y GPU.
- Interfaz de alto nivel (Keras) y bajo nivel (gráficos de flujo de datos).
- Capacidad para guardar, exportar y desplegar modelos.

¿Qué es Keras?

Keras es una API de alto nivel integrada dentro de TensorFlow (tf.keras), que permite construir y entrenar redes neuronales de forma fácil y rápida, sin preocuparse demasiado por los detalles técnicos.

Ventajas de Keras:

- Muy intuitiva y legible.
- Permite construir modelos con pocas líneas de código.
- Fácil de depurar y experimentar.
- Compatible con herramientas de producción (como TensorFlow Serving).

El Modelo: `tf.keras.Sequential`

En tu programa se usa la clase `tf.keras.Sequential`, que permite construir un modelo capa por capa, de manera lineal. Es ideal para redes en las que cada capa tiene una sola entrada y una sola salida.

```

1 modelo = tf.keras.Sequential([
2     tf.keras.layers.Dense(units=3, input_shape=[1]), # Capa oculta 1
3     tf.keras.layers.Dense(units=3),                  # Capa oculta 2
4     tf.keras.layers.Dense(units=1)                    # Capa de salida
5 ])

```

- **Dense:** indica que cada neurona está completamente conectada a la capa anterior (fully connected).
- **units=3:** significa que esa capa tendrá 3 neuronas.
- **input_shape=[1]** : especifica que la entrada es un número real (un solo valor: el ángulo en grados).

Compilación del Modelo:

`modelo.compile()` Este paso configura el modelo para el entrenamiento.

Se le especifica:

```

1 modelo.compile(
2     optimizer=tf.keras.optimizers.Adam(0.1),
3     loss="mean_squared_error"
4 )

```

- **El optimizador (optimizer)** se encarga de actualizar los pesos del modelo para reducir el error. En este caso:
 - **Adam(0.1):** Es un optimizador avanzado que combina las ventajas de los métodos momentum y RMSProp. Es rápido y eficiente.
 - **learning rate:** 0.1 es la tasa de aprendizaje, en otras palabras, qué tan grande es el paso que da el algoritmo para ajustar los pesos en cada iteración.
- **loss (Función de pérdida):**
 - **"mean_squared_error":** Esta función calcula el promedio del cuadrado de las diferencias entre los valores reales (radianes) y los valores predichos. El modelo busca minimizar esta función durante el entrenamiento.

Entrenamiento del Modelo

```
1 historial = modelo.fit(grados, radianes, epochs=1000, verbose=False)
```

Aquí se entrena el modelo durante 1000 épocas, donde:

- **Época:** Una pasada completa por todos los datos de entrenamiento.
- **verbose=False:** No imprime detalles en cada época. Durante el entrenamiento, el modelo ajusta sus pesos para que la salida se aproxime a la función real: $\text{radianes} = \text{grados} \times \left(\frac{\pi}{180}\right)$

¿Qué aprende el modelo?

Aunque no le damos explícitamente la fórmula de conversión entre grados y radianes, el modelo aprende a aproximarla a partir de los datos de entrenamiento. Al final, se comporta como una función matemática que predice correctamente los valores radianes dados los grados.

PROGRAMA PYTHON O.O COMPLETO

```
1 """
2 Creado junio 23 de 2025
3 Nombre: APRENDIZAJE AUTOMÁTICO PARA CONVERSIÓN DE GRADOS A Radianes
4 Versión orientada a objetos (POO)
5
6 Descripción:
7 Este programa entrena una red neuronal con TensorFlow para aprender
8 la relación entre grados y radianes. El modelo se construye usando
9 dos capas ocultas y una capa de salida.
10 """
11
12 import tensorflow as tf
13 import numpy as np
14 import math
15 import matplotlib.pyplot as plt
16
17 # Definición de la clase principal
18 class ConversorGradosRadianes:
19     def __init__(self):
20         self.grados = np.array([0, 30, 45, 60, 90], dtype=float)
21         self.radianes = np.array([0, math.pi/6, math.pi/4, math.pi/3, math.pi/2], dtype=float)
22         self.modelo = self.construir_modelo()
23         self.historial = None
24
25     def construir_modelo(self):
```

```

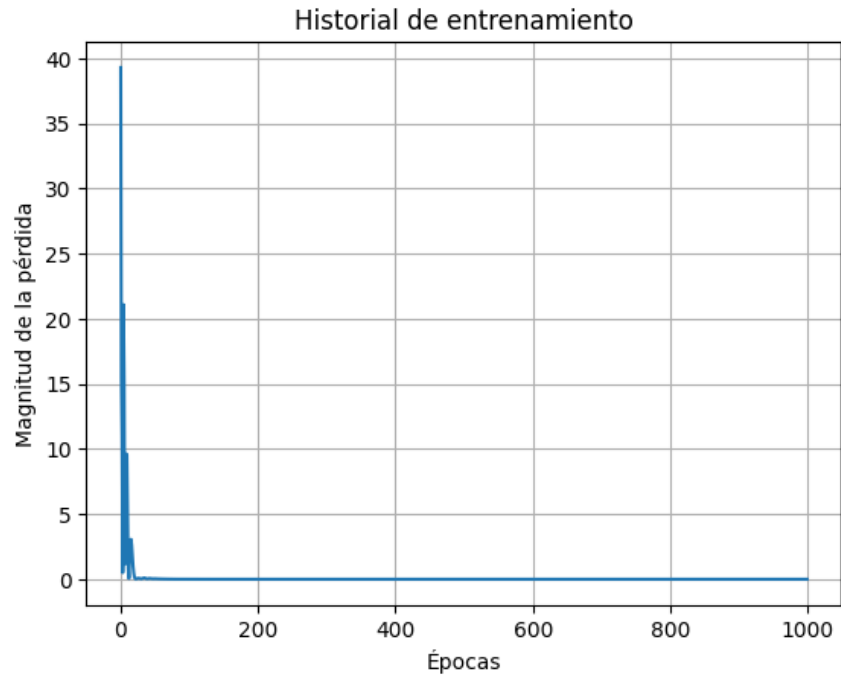
26     # Construcción secuencial del modelo
27     modelo = tf.keras.Sequential([
28         tf.keras.layers.Dense(units=3, input_shape=[1]), # Capa oculta 1
29         tf.keras.layers.Dense(units=3),                  # Capa oculta 2
30         tf.keras.layers.Dense(units=1)                    # Capa de salida
31     ])
32     # Compilación del modelo
33     modelo.compile(
34         optimizer=tf.keras.optimizers.Adam(0.1),
35         loss="mean_squared_error"
36     )
37     return modelo
38
39     def entrenar(self, epocas=1000):
40         print("Iniciando entrenamiento ...")
41         self.historial = self.modelo.fit(self.grados, self.radianes, epochs=epocas, verbose=False)
42         print("Entrenamiento finalizado")
43
44     def graficar_perdida(self):
45         if self.historial:
46             plt.xlabel("Épocas")
47             plt.ylabel("Magnitud de la pérdida")
48             plt.plot(self.historial.history["loss"])
49             plt.title("Historial de entrenamiento")
50             plt.grid(True)
51             plt.show()
52
53     def predecir(self, valor_en_grados):
54         entrada = np.array([valor_en_grados], dtype=float)
55         resultado = self.modelo.predict(entrada, verbose=False)
56         print(f"El resultado es {resultado[0][0]} radianes")
57
58     # Bloque principal
59     if __name__ == "__main__":
60         conversor = ConversorGradosRadianes()
61         conversor.entrenar()
62         conversor.graficar_perdida()
63         print("Hagamos una predicción:")
64         conversor.predecir(180.0)

```

Este código genera la salida:

Iniciando entrenamiento ...

Entrenamiento finalizado



Hagamos una predicción:

El resultado es 3.141592502593994 radianes