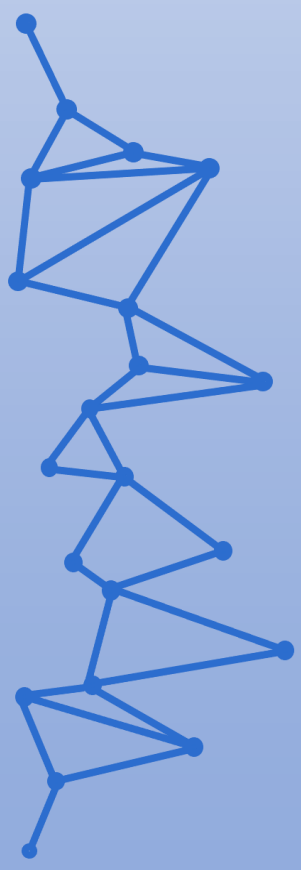
Curso de Especialización de
Inteligencia Artificial y Big Data (IABD)



# Programación de Inteligencia Artificial

UD06. Monitorización del entrenamiento de redes neuronales profundas.
Tarea Online.

JUAN ANTONIO GARCÍA MUELAS

## INDICE

**Tarea para PIA06**

**Título de la tarea: Monitorización del entrenamiento de redes neuronales profundas**

**Ciclo formativo y módulo: Curso especialización en Inteligencia Artificial y Big Data - Programación de Inteligencia Artificial**

**Curso académico: 2022-2023**

## ¿Qué te pedimos que hagas?

### ✓ Apartado 1: Inicializa un proyecto de regresión con TensorBoard

- Inicia un nuevo notebook, preferiblemente en Google Colab.

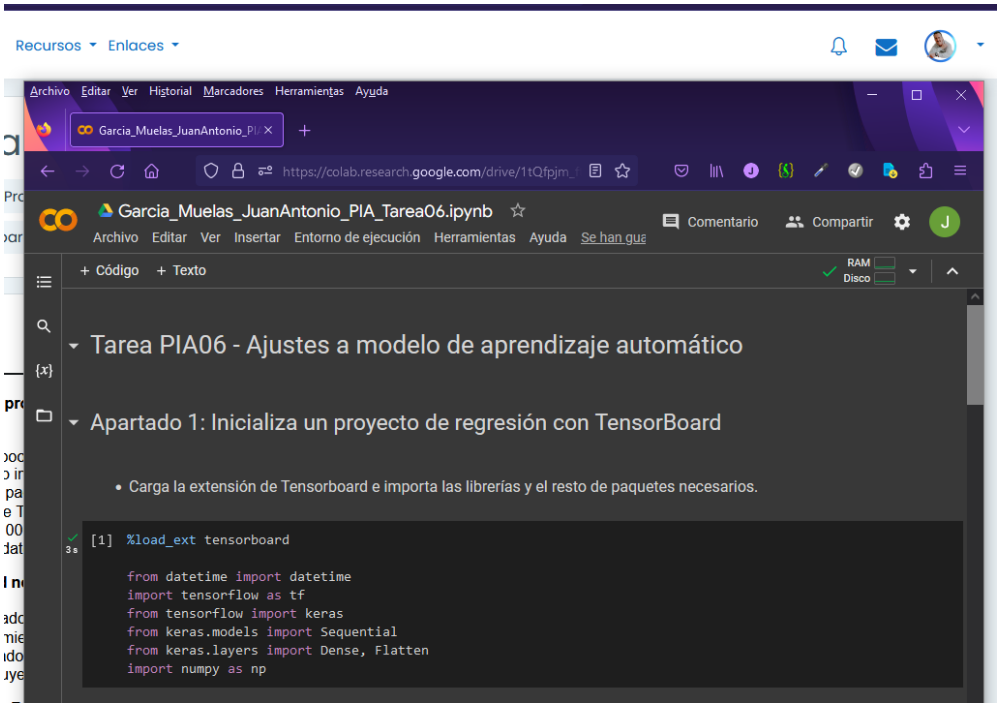
  Puede accederse al notebook creado desde este enlace.

- Importa la librerías y paquetes necesarios

```
from datetime import datetime
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, Flatten
import numpy as np
```

- Carga la extensión de TensorBoard.

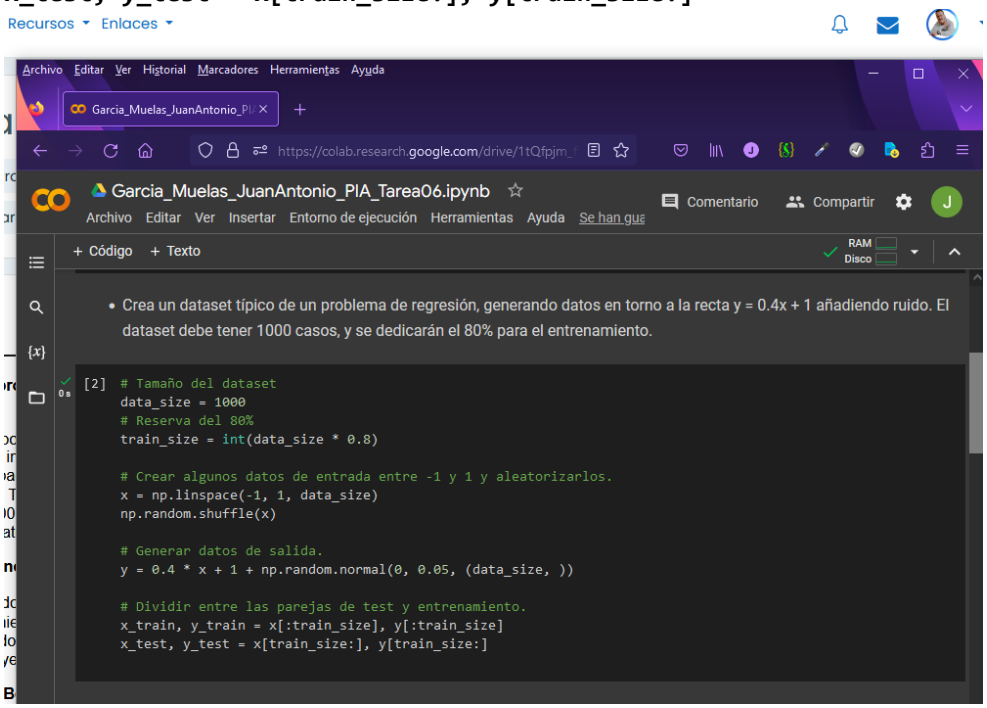
```
%load_ext tensorboard
```




- Crea un dataset de 1000 casos de variables x e y basados en añadir ruido a puntos aleatorios de la recta de ecuación y = 0.4x + 1
- Reservar el 80% de datos para el entrenamiento y el 20% para validación o test.

```
# Tamaño del dataset
data_size = 1000
# Reserva del 80%
train_size = int(data_size * 0.8)
```

```
# Crear algunos datos de entrada entre -1 y 1 y aleatorizarlos.
x = np.linspace(-1, 1, data_size)
np.random.shuffle(x)

# Generar datos de salida.
y = 0.4 * x + 1 + np.random.normal(0, 0.05, (data_size, ))

# Dividir entre las parejas de test y entrenamiento.
x_train, y_train = x[:train_size], y[:train_size]
x_test, y_test = x[train_size:], y[train_size:]
```




✓ **Apartado 2: Crea una red neuronal profunda de dos capas con Loss MSE y optimizador SGD.**

➢ Crea un modelo basado en una red neuronal de dos capas.

```
# Programar el modelo:
model = tf.keras.models.Sequential([
  tf.keras.layers.Dense(32, activation='relu', input_shape=[1]),
  tf.keras.layers.Dense(1)
])
```

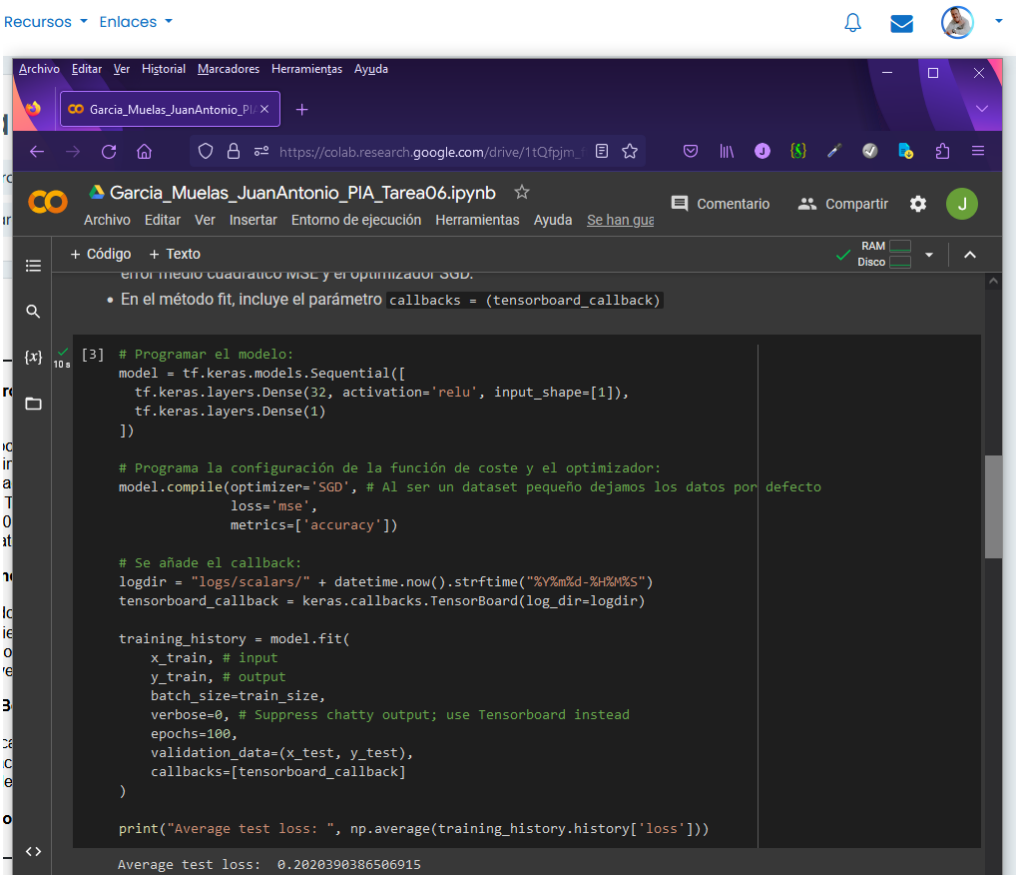
➢ Configura el entrenamiento utilizando la función de coste del error medio cuadrático (MSE).

➢ Configura el optimizador SGD

```
# Programa la configuración de la función de coste y el optimizador

# Al ser un dataset pequeño, lo dejamos con los datos por defecto
model.compile(optimizer='SGD',
              loss='mse',
              metrics=['accuracy'])
```

- En el método fit, incluye el parámetro callbacks = (tensorboard_callback)

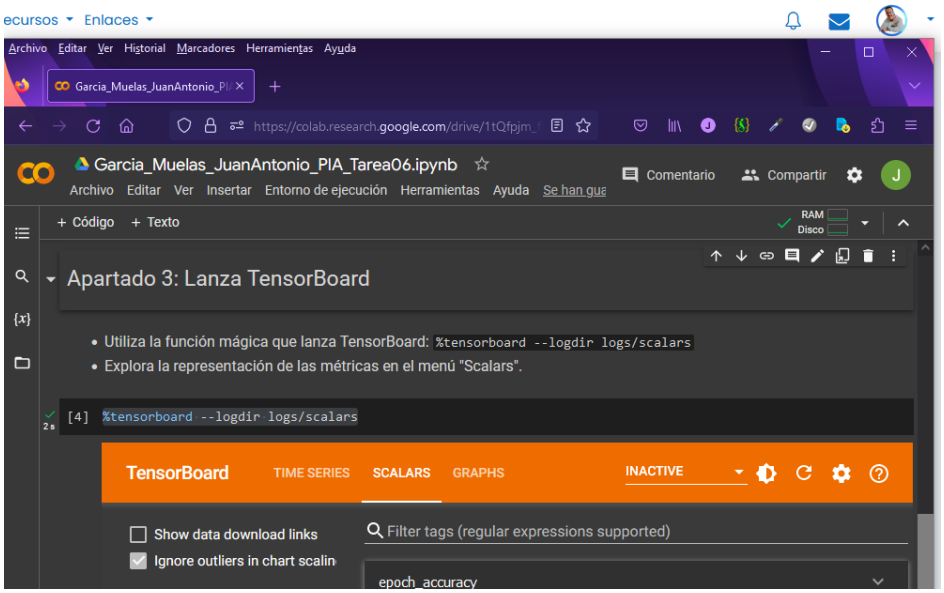
```
# Se añade el callback:
logdir = "logs/scalars/" + datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = keras.callbacks.TensorBoard(log_dir=logdir)

training_history = model.fit(
    x_train, # input
    y_train, # output
    batch_size=train_size,
    verbose=0, # Suppress chatty output; use Tensorboard instead
    epochs=100,
    validation_data=(x_test, y_test),
    callbacks=[tensorboard_callback]
)

print("Average test loss: ", np.average(training_history.history['los
s']))
```



## ✓ Apartado 3: Lanza TensorBoard.

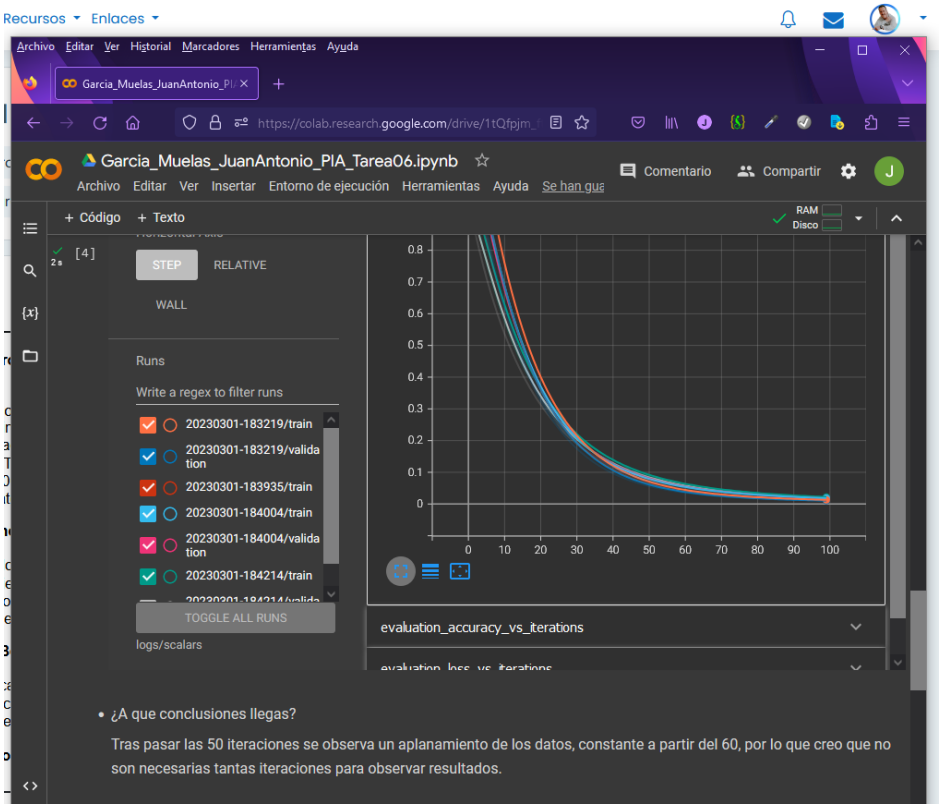
➢ Utiliza la función mágica que lanza TensorBoard: *%tensorboard --logdir logs/scalars*.

```
%tensorboard --logdir logs/scalars
```




➢ Explora las representaciones de las métricas

➢ ¿A qué conclusiones llegas?




Según se va entrenando, al pasar de las 50 iteraciones se muestra un aplanamiento de los datos, más constante a partir del 60, que sugiere que no serán necesarias más iteraciones con este dataset.

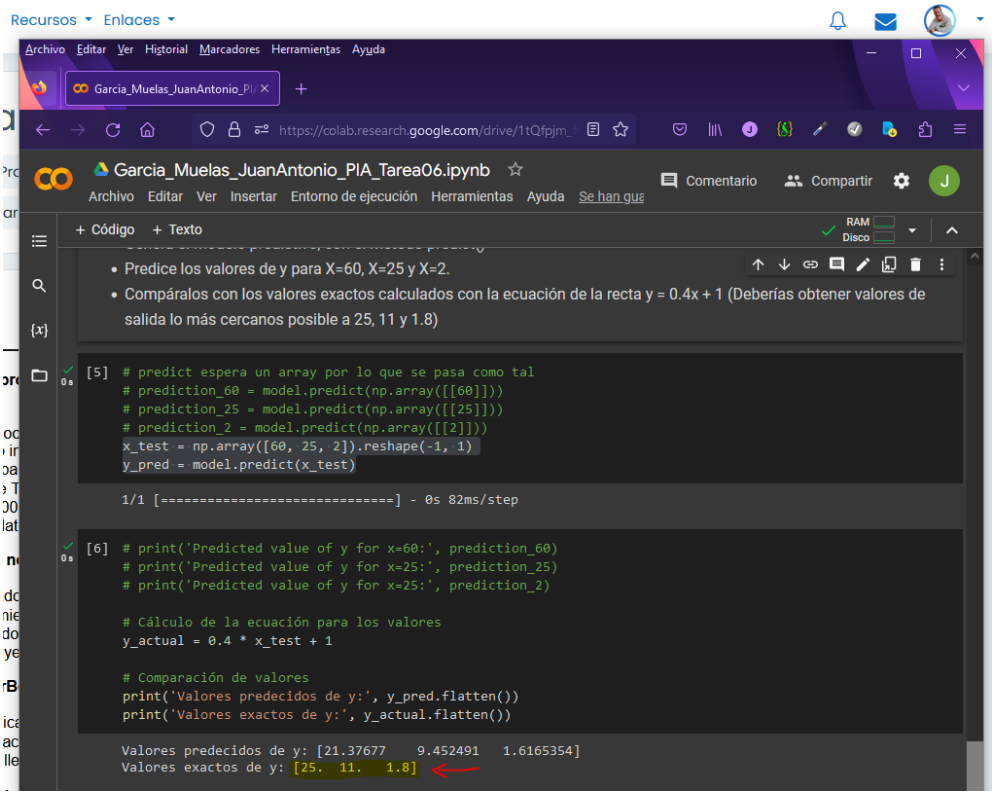
✓ **Apartado 4: Crea el modelo predictivo.**

- Genera el modelo predictivo con el método predict().
- Predice los valores para x=60, x=25 y x=2.

```
x_test = np.array([60, 25, 2]).reshape(-1, 1)
y_pred = model.predict(x_test)
```

- Compara los valores predichos con los calculados de forma exacta con la ecuación de la recta y = 0.4x + 1 y analiza los resultados.

```
# Cálculo de la ecuación para los valores
y_actual = 0.4 * x_test + 1

# Comparación de valores
print('Valores predecidos de y:', y_pred.flatten())
print('Valores exactos de y:', y_actual.flatten())
```




Se observa una diferencia en los datos que perfectamente puede coincidir si sustraemos la tasa de error (test loss: 0.2020)