Universidad Autónoma de San Luis Potosí
Facultad de ingeniería
Inteligencia Artificial Aplicada

Practica 6

Implementación de una red neuronal en sistemas embebidos

Ana Sofía Medina Martínez

Fecha 3/10/2024

Objetivo

El estudiante aprenderá a construir, compilar y entrenar una red neuronal utilizando la librería Keras para resolver problemas de clasificación y regresión y que aprenda a importar modelos de redes neuronales en sistemas embebidos.

Procedimiento

6.1.- Sigue las instrucciones del archivo “practica_6_training” para desarrollar y entrenar un modelo de red neuronal de clasificación.

6.2.- Sigue las instrucciones del archivo “practica_6_inferencia” para implementar una red neuronal en un microcontrolador.

Resultados

```
[ ] import pandas as pd

    data = pd.read_csv("./data/datos_mpu6050.csv")#TODO: Cargar el dataset usando pandas

    #Mezclamos los datos para que no haya sesgo
    data = data.sample(frac=1)

    data.head()
```

	acelX	acelY	acelZ	output
1513	10.45	-0.73	0.01	DOWN
607	1.35	9.46	2.10	RIGHT
684	1.27	9.23	3.01	RIGHT
1648	9.22	0.26	-4.97	DOWN
1166	0.99	-9.88	0.40	LEFT

Next steps: Generate code with data | View recommended plots | New interactive sheet

1.2 Preprocesar los datos

Vamos a realizar los siguientes pasos para preprocesar los datos:

- Separar los datos en entrada y salida.
- Codificar las etiquetas de salida aplicando one-hot encoding.
- Separar los datos en entrenamiento y prueba.

```
[ ] #TODO: Crear un dataframe con las columnas acelX, acelY y acelZ, guardalo en la variable X
    X = data[["acelX", "acelY", "acelZ"]]
    X.head()
```

	acelX	acelY	acelZ
1513	10.45	-0.73	0.01
607	1.35	9.46	2.10

	acelX	acelY	acelZ
1513	10.45	-0.73	0.01
607	1.35	9.46	2.10
684	1.27	9.23	3.01
1648	9.22	0.26	-4.97
1166	0.99	-9.88	0.40

Next steps: Generate code with X | View recommended plots | New interactive sheet

```
#TODO: Crear un dataframe con la columna output, guardalo en la variable y
y = data["output"]
y.head()
```

	output
1513	DOWN
607	RIGHT
684	RIGHT
1648	DOWN
1166	LEFT

dtype: object

Para codificar las etiquetas de salida vamos a utilizar la función `get_dummies` de pandas.

```
import pandas as pd

encoded_labels = pd.get_dummies(labels, dtype=dtype)
```

- `labels`: es un arreglo de numpy o una serie de pandas con las etiquetas de salida.
- `dtype`: es el tipo de dato de las columnas de la matriz de salida (int, float, etc).

```
#TODO: Convertir la columna y en un one-hot encoding usando pd.get_dummies, recuerda especificar el tipo de dato como float
y = pd.get_dummies(y, dtype=float)
y.head()
```

	DOWN	IDLE	LEFT	RIGHT	TOP
1513	1.0	0.0	0.0	0.0	0.0
607	0.0	0.0	0.0	1.0	0.0
684	0.0	0.0	0.0	1.0	0.0
1648	1.0	0.0	0.0	0.0	0.0
1166	0.0	0.0	1.0	0.0	0.0

Next steps: Generate code with y | View recommended plots | New interactive sheet

Un paso muy importante es separar los datos en entrenamiento y prueba, esto nos permitirá evaluar el modelo en datos que no ha visto durante el entrenamiento.

Realizaremos la separación utilizando la función `sample` de pandas.

```
train_data = data.sample(frac=0.8)
test_data = data.drop(train_data.index)
```

- **frac**: es el porcentaje de datos que se utilizarán para entrenamiento. En este caso, el 80% de los datos se utilizarán para entrenamiento.
- **drop**: Elimina las filas con los índices especificados, en este caso, eliminamos las filas que se utilizaron para entrenamiento.

```
#TODO: Dividir los datos en entrenamiento y prueba usando X.sample y y.sample
# Guarda los datos de entrenamiento en las variables X_train y y_train
# Guarda los datos de prueba en las variables X_test y y_test
X_train = X.sample(frac=0.8, random_state=0)
X_test = X.drop(X_train.index)

y_train = y.sample(frac=0.8, random_state=0)
y_test = y.drop(y_train.index)
```

```
mean = X_train.mean(axis=0)
std = X_train.std(axis=0)

print(f"MEAN: {mean.values}")
print(f"STD: {std.values}")
```

```
MEAN: [ 1.47187113 -0.27938042  3.00908922]
STD: [6.02016609 5.63723474 4.26806085]
```

Los valores obtenidos en la celda anterior se utilizaran para normalizar los datos en la ESP32.

```
X_train = (X_train - mean) / std
X_test = (X_test - mean) / std

X_train.head()
```

	acelX	acelY	acelZ
1124	-1.325856	-0.032040	0.937407
1674	1.376728	-0.152667	-1.433693
1001	-1.427182	-0.482616	0.672650
785	-1.691294	0.054882	-0.044303
1935	1.431543	0.274848	-0.674566

Next steps: Generate code with X_train | View recommended plots | New interactive sheet

```
[ ]  #TODO: Convertir los datos de entrenamiento y prueba a un arreglo de numpy usando .values
     X_train = X_train.values
     X_test = X_test.values
     y_train = y_train.values
     y_test = y_test.values
```

2. Crear el modelo

Ya que hemos preprocesado los datos, vamos a crear el modelo de la red neuronal.

El modelo que vamos a crear es un modelo secuencial, que consta de las siguientes capas:

- Capa de entrada.
- Capa densa con 8 neuronas y función de activación ReLU.
- Capa de salida con 5 neuronas y función de activación softmax.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()

model.add(Input(shape=(3,)))
model.add(Dense(8, activation='relu'))
```

- `Input`: Capa de entrada.
- `Dense`: Capa densa.
- `shape`: Forma de los datos de entrada.
- `activation`: Función de activación.

```
[ ]  from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Dense, Input

     #TODO: Crear un modelo secuencial
     model = Sequential()
```

```
[ ]  #TODO: Crear un modelo secuencial
     model = Sequential()

     #TODO: Agregar una capa de entrada con 3 neuronas, una para cada columna de X
     model.add(Input(shape=(3,)))

     #TODO: Agregar una capa densa con 8 neuronas y activación relu
     model.add(Dense(8, activation='relu'))

     #TODO: Agregar una capa densa con 5 neuronas y activación softmax, 5 neuronas porque tenemos 5 clases en la salida
     model.add(Dense(5, activation='softmax'))

     model.summary()
```

```
Model: "sequential_4"
```

Layer (type)	Output Shape	Param #
dense_9 (Dense)	(None, 8)	32
dense_10 (Dense)	(None, 5)	45

```
Total params: 77 (308.00 B)
Trainable params: 77 (308.00 B)
Non-trainable params: 0 (0.00 B)
```

Después de definir la arquitectura de la red, vamos a compilar el modelo.

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

- `optimizer`: Optimizador.
- `loss`: Función de pérdida, en este caso, utilizamos la entropía cruzada categórica porque estamos realizando una clasificación multiclase.
- `metrics`: Métricas que se utilizarán para evaluar el modelo.

```
[ ]  model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Por último, vamos a entrenar el modelo.

```
history = model.fit(X_train, y_train, epochs=20, validation_data=(X_test, y_test))
```

```
Epoch 1/20
51/51 ━━━━━━━━━━━━━━━━━━━━ 4s 23ms/step - accuracy: 0.1585 - loss: 1.7272 - val_accuracy: 0.3812 - val_loss: 1.5329
Epoch 2/20
51/51 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.3956 - loss: 1.5063 - val_accuracy: 0.6485 - val_loss: 1.3359
Epoch 3/20
51/51 ━━━━━━━━━━━━━━━━━━━━ 2s 33ms/step - accuracy: 0.6186 - loss: 1.3153 - val_accuracy: 0.7005 - val_loss: 1.1424
Epoch 4/20
51/51 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7448 - loss: 1.1026 - val_accuracy: 0.9010 - val_loss: 0.9529
Epoch 5/20
51/51 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.8988 - loss: 0.9177 - val_accuracy: 0.9505 - val_loss: 0.7792
Epoch 6/20
51/51 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9602 - loss: 0.7553 - val_accuracy: 0.9678 - val_loss: 0.6341
Epoch 7/20
51/51 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9778 - loss: 0.6097 - val_accuracy: 0.9802 - val_loss: 0.5174
Epoch 8/20
51/51 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9904 - loss: 0.5000 - val_accuracy: 0.9926 - val_loss: 0.4227
Epoch 9/20
51/51 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9935 - loss: 0.4077 - val_accuracy: 0.9975 - val_loss: 0.3463
Epoch 10/20
51/51 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9968 - loss: 0.3362 - val_accuracy: 0.9975 - val_loss: 0.2851
Epoch 11/20
51/51 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9975 - loss: 0.2706 - val_accuracy: 0.9975 - val_loss: 0.2364
Epoch 12/20
51/51 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9980 - loss: 0.2244 - val_accuracy: 0.9975 - val_loss: 0.1978
Epoch 13/20
51/51 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9981 - loss: 0.1863 - val_accuracy: 0.9975 - val_loss: 0.1676
Epoch 14/20
51/51 ━━━━━━━━━━━━━━━━━━━━ 2s 20ms/step - accuracy: 0.9992 - loss: 0.1549 - val_accuracy: 1.0000 - val_loss: 0.1432
Epoch 15/20
51/51 ━━━━━━━━━━━━━━━━━━━━ 2s 29ms/step - accuracy: 0.9998 - loss: 0.1391 - val_accuracy: 1.0000 - val_loss: 0.1235
Epoch 16/20
51/51 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9982 - loss: 0.1191 - val_accuracy: 1.0000 - val_loss: 0.1076
Epoch 17/20
51/51 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9985 - loss: 0.1030 - val_accuracy: 1.0000 - val_loss: 0.0945
Epoch 18/20
51/51 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9997 - loss: 0.0887 - val_accuracy: 1.0000 - val_loss: 0.0838
Epoch 19/20
51/51 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 1.0000 - loss: 0.0774 - val_accuracy: 1.0000 - val_loss: 0.0746
Epoch 20/20
51/51 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0704 - val_accuracy: 1.0000 - val_loss: 0.0669
```

```
from tensorflow import lite as tflite

model_name = "mpu6050_model" #Nombre del archivo donde se guardará el modelo

converter = tflite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert() #Convertimos el modelo a un modelo tflite

with open(f"{model_name}.tflite", 'wb') as f: #Abrimos un archivo en modo escritura binaria
    f.write(tflite_model) #Guardamos el modelo en un archivo llamado model.tflite
```

```
Saved artifact at '/tmp/tmp7ofzr28f'. The following endpoints are available:

* Endpoint 'serve'
  args_0 (POSITIONAL_ONLY): TensorSpec(shape=(None, 3), dtype=tf.float32, name='keras_tensor_19')
Output Type:
  TensorSpec(shape=(None, 5), dtype=tf.float32, name=None)
Captures:
  135366664022928: TensorSpec(shape=(), dtype=tf.resource, name=None)
  135366529165168: TensorSpec(shape=(), dtype=tf.resource, name=None)
  135366528102144: TensorSpec(shape=(), dtype=tf.resource, name=None)
  135366528100560: TensorSpec(shape=(), dtype=tf.resource, name=None)
```

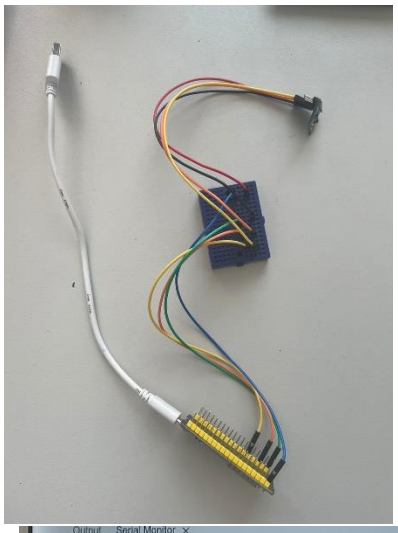
Muchas plataformas de microcontroladores no tienen soporte para TensorFlow Lite. La forma más sencilla de ejecutar un modelo de TensorFlow Lite en un microcontrolador es convertirlo a una matriz de bytes y ejecutarlo en el microcontrolador.

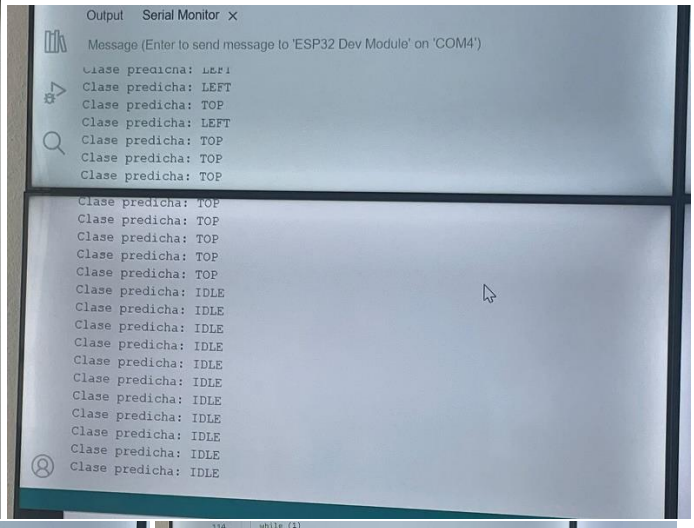
```
def tflite_to_array(model_data, model_name):
    c_str = ""

    #Creamos las cabeceras del archivo
    c_str += f"#ifndef {model_name.upper()}_H\n"
    c_str += f"#define {model_name.upper()}_H\n\n"

    #Agregamos una variable con el tamaño del modelo
    c_str += f"const unsigned int {model_name}_len = {len(model_data)};\n\n"

    #Agregamos el modelo como un arreglo de bytes
    c_str += f"const unsigned char {model_name}[] = {{\n"

    for i, byte in enumerate(model_data):
        c_str += f"0x{byte:02X},"
        if (i + 1) % 12 == 0:
```

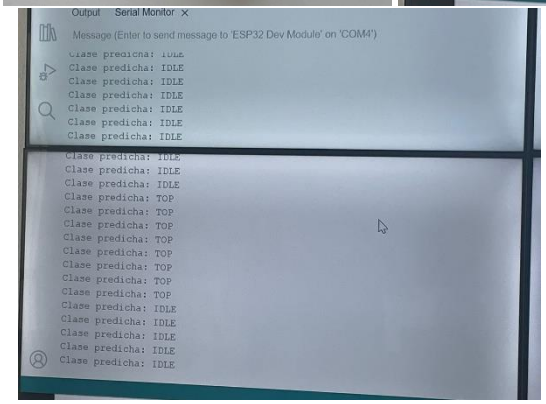
```
        for i, byte in enumerate(model_data):
            c_str += f"0x{byte:02X},"
            if (i + 1) % 12 == 0:
                c_str += "\n"

        c_str += "};\n\n"

        #Cerramos las cabeceras del archivo
        c_str += f"#endif // {model_name.upper()}_H\n"

        return c_str
```

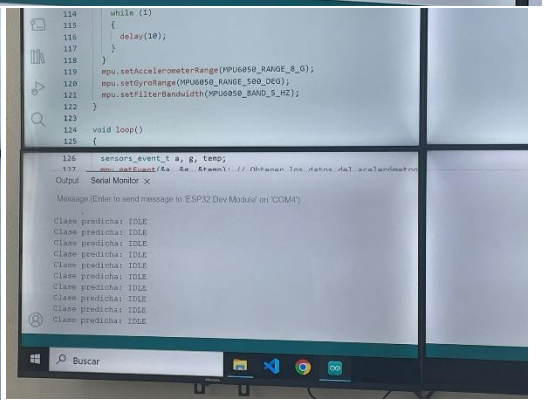
```
model_array = tflite_to_array(tflite_model, model_name)

with open(f"{model_name}.h", 'w') as f:
    f.write(model_array)
```

Resultados en Arduino









Comprensión

1. **¿Qué pasos se deben seguir para entrenar una red neuronal con Keras?**
   - Definir el modelo.
   - Compilar el modelo especificando optimizador, función de pérdida y métricas.
   - Preparar los datos.
   - Entrenar el modelo con el conjunto de entrenamiento
   - Evaluar el rendimiento con el conjunto de prueba

2. **¿Cuál es la función del conjunto de entrenamiento y del conjunto de prueba en el proceso de entrenamiento?**
   - El conjunto de entrenamiento se usa para ajustar los pesos del modelo.
   - El conjunto de prueba evalúa el rendimiento generalizado del modelo, comprobando que no haya sobreajuste.

3. **¿Qué se entiende por función de pérdida, optimizador y métricas en el contexto del entrenamiento de una red neuronal?**
   - Función de pérdida: mide qué tan bien o mal está funcionando el modelo.
   - Optimizador: ajusta los pesos para minimizar la pérdida.
   - Métricas: evaluan el rendimiento del modelo durante el entrenamiento.

4. **¿Qué tipo de problemas se pueden resolver utilizando una red neuronal entrenada con Keras?**
   Clasificación, regresión, reconocimiento de imágenes, procesamiento de lenguaje natural, predicción de series temporales, entre otros.

5. **5.- ¿Qué es IA on the Edge?**
   Es el uso de inteligencia artificial directamente en dispositivos locales, sin necesidad de depender de servidores o la nube para realizar cálculos.

6. **6.- ¿Qué ventajas tiene IA on the Edge en los sistemas embebidos?**
   Menor latencia, mayor seguridad de datos, menor consumo de ancho de banda y operación en tiempo real, incluso sin conexión a internet.

7. **7.- ¿Qué es TensorFlow Lite?**
   Es una versión optimizada de TensorFlow diseñada para ejecutar modelos de machine learning en dispositivos móviles y sistemas embebidos.

Conclusiones

En conclusión, entrenar una red neuronal con Keras implica seguir un proceso que incluye la preparación de datos, la definición del modelo y su evaluación. La importancia de usar conjuntos de entrenamiento y prueba asegura que el modelo generalice correctamente, evitando el sobreajuste. Además, herramientas como TensorFlow Lite permiten llevar la inteligencia artificial a dispositivos embebidos, impulsando el desarrollo de soluciones de IA, que ofrecen ventajas como mayor eficiencia, menor latencia y mejor privacidad de los datos.

El reto en esta práctica fue la falta de espacio de almacenamiento en mi computadora y la falta de conocimiento de microcontroladores y de Arduino, sin embargo, fui muy interesante desarrollar la práctica.