

Uso del script en Colaboratory

Práctica 2 - parte 1

Curso 2021-22

Introducción

- Colaboratory es un entorno gratuito de Jupyter Notebook que no requiere configuración y que se ejecuta completamente en la nube.
- Es un Jupyter Notebook (JN) almacenado en Google Drive. Estos cuadernos están compuestos de celdas que pueden contener código, texto e imágenes entre otros elementos. Podremos ejecutar código Python sin necesitar una instalación específica en nuestro ordenador.
- Podemos utilizar un JN y guardar una copia en Drive. (Archivo/guardar una copia en Drive).
- Para poder utilizar Colaboratory es necesario tener abierta una cuenta de Google (la cuenta del correo de la Universidad bastaría).
- <https://colab.research.google.com/notebooks/welcome.ipynb>

Conjunto de datos a utilizar

- Utilizaremos el conjunto de datos “Vehicle Silhouettes”
- 846 instancias
- 4 clases: : BUS, OPEL, SAAB, VAN
- 18 atributos: características extraídas de imágenes de distintos tipos de vehículos
- Entrenamiento: 2/3 de los datos
- Test: 1/3 de los datos
- El alumno normalizará las entradas de los datos y obtendrá los conjuntos de entrenamiento y test
- El script se encarga de codificar la salida (one-hot encoding: 4 columnas)

Metodología

- Con el conjunto de entrenamiento se realizarán pruebas con modelos con diferentes hiperparámetros para encontrar la red que menor error obtenga.
- Para ello, del conjunto de entrenamiento se extraerá un conjunto de validación (20% de los datos).
- El criterio de red óptima: la que menor *loss* obtenga en validación.
- Se utilizarán redes sencillas
 - MSE como función objetivo (loss)
 - 1 o varias (pocas) capas ocultas
 - Función de activación sigmoideal

Metodología

- La red realmente hará regresión y generará 4 salidas numéricas
- Se tomará la neurona con mayor activación para asignar la clase a esa instancia
- Una vez obtenida la red óptima, se construirá un modelo con esos hiperparámetros tomando **todo el conjunto** de entrenamiento
- Se evaluará con el conjunto de test (sólo la red óptima). Obtener:
 - Las salidas en bruto de las primeras instancias de test
 - La clase correspondiente para las primeras instancias en test
 - El valor de Accuracy (porcentaje de aciertos)
 - La matriz de confusión

Descripción del script

- Se trabajará con Keras que es una librería de alto nivel, escrita en Python, que utiliza la librería de más bajo nivel TensorFlow.
- Fue desarrollada para poder hacer experimentación de una forma muy rápida (prototipado).
- Importaremos TensorFlow, ciertos módulos de Keras y de la librería de aprendizaje automático scikit-learn que nos permitirá obtener las matrices de confusión y otros informes de una manera muy sencilla

```
import tensorflow as tf
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import confusion_matrix, classification_report
```

Cargar los datos

- Tendremos preparados los datos normalizados y separados en entrenamiento y test
- Separamos las entradas y las salidas de entrenamiento y test:
 - X_train, y_train
 - X_test, y_test

```
train_set = pd.read_csv('/content/sample_data/vehicleTrainValid.csv', header='infer', delimiter=',')
test_set = pd.read_csv('/content/sample_data/vehicleTest.csv', header='infer', delimiter=',')

# SELECCION DE LA SALIDA
y_train = train_set.iloc[:, -1:]
X_train = train_set.iloc[:, :-1]

y_test = test_set.iloc[:, -1:]
X_test = test_set.iloc[:, :-1]
```

Codificar la salida

```
# CONVERTIR TARGET A CATEGORICAL
from sklearn.preprocessing import LabelBinarizer
encoder = LabelBinarizer()
y_train_transformed = encoder.fit_transform(y_train)
y_test_transformed = encoder.fit_transform(y_test)

print(y_train[:11])
print(y_train_transformed[:11])
```

	target		bus	opel	saab	van
0	bus		[1	0	0	0]
1	bus		[1	0	0	0]
2	van		[0	0	0	1]
3	van		[0	0	0	1]
4	van	→	[0	0	0	1]
5	saab		[0	0	1	0]
6	van		[0	0	0	1]
7	bus		[1	0	0	0]
8	saab		[0	0	1	0]
9	opel		[0	1	0	0]]

Ver la estructura de los datos

```
# COMPROBAR DIMENSIONES DE LOS DATOS
print(X_train.shape)
print(y_train.shape)
print(y_train_transformed.shape)
print(X_test.shape)
print(y_test.shape)
print(y_test_transformed.shape)

#obtener dim de la entrada y número de salidas
input_shape = (X_train.shape [1] ,)          # (18, )
num_clases = y_train_transformed.shape [1]    # 4
```

```
(558, 18)
(558, 1)
(558, 4)
(288, 18)
(288, 1)
(288, 4)
```

Definir el modelo de red de neuronas. PM

- En esta sección definiremos el modelo utilizando Keras
- Usaremos un modelo secuencial y vamos añadiendo capas
- Ejemplo de PM muy básico
 - La capa de entrada tendrá 18 entradas (input_shape).
 - Añadimos una capa oculta de 350 neuronas con función de activación sigmoidal
 - La última capa tendrá 4 neuronas (una por clase) con función de activación 'softmax'. Es una función similar a la sigmoide pero normalizada de forma que todas las salidas suman 1. Así se podrá corresponder cada valor con una probabilidad.

```
#Definición del modelo
model = Sequential()
model.add(Dense(350, input_shape=input_shape, activation='sigmoid'))
model.add(Dense(num_clases, activation='softmax'))
```

Visualizar el modelo

- Puede ser interesante visualizar el modelo, para ver el número total de parámetros (pesos)

```
model.summary()
```

Model: "sequential_7"

Layer (type)	Output Shape	Param #
dense_6 (Dense)	(None, 350)	6650
dense_7 (Dense)	(None, 4)	1404

Total params: 8,054
Trainable params: 8,054
Non-trainable params: 0

Configurar el modelo y entrenarlo

- Configuramos:
 - El optimizador (*sgd*): descenso del gradiente estocástico (con su lr)
 - La función objetivo (*Loss*): Mean Squared Error
 - La métrica que será evaluada durante el entrenamiento (*accuracy*). Podemos añadir 'mse' pero como ya es la función objetivo, se mide en todo caso.
 - Con el argumento *validation_split* podemos separar un conjunto de validación para luego poder decidir los parámetros más adecuados
 - En *history* se guardan los valores obtenidos durante el entrenamiento

```
# CONFIGURAR MODELO Y ENTRENAMIENTO (TRAINING 80%, VALIDACION 20%)
```

```
model.compile(loss='mean_squared_error', optimizer=tf.keras.optimizers.SGD(learning_rate=0.2, momentum=0), metrics=['accuracy','mse'])
```

```
historico = model.fit(X_train, y_train_transformed, epochs=100, batch_size=1, verbose=1, validation_split=0.2, shuffle=False)
```

```
Epoch 1/100
```

```
446/446 [=====] - 1s 1ms/step - loss: 0.2497 - accuracy: 0.2422 - mse: 0.2497 - val_loss: 0.3267 - val_accuracy: 0.2411 - v.
```

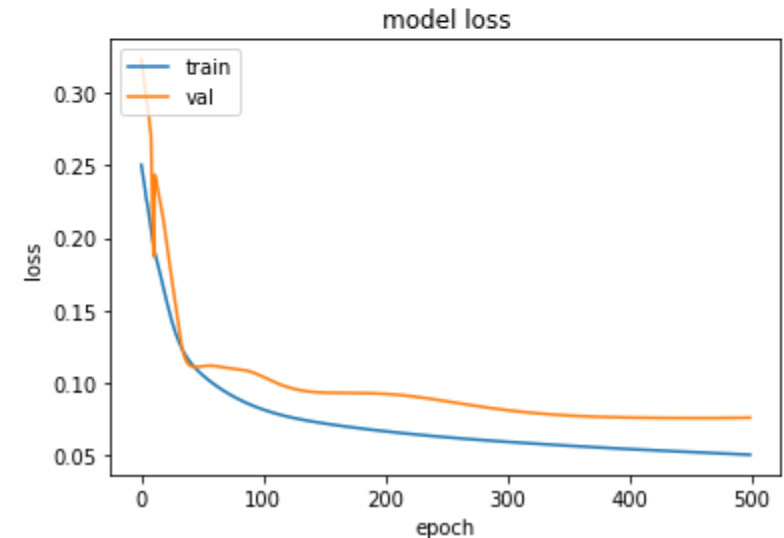
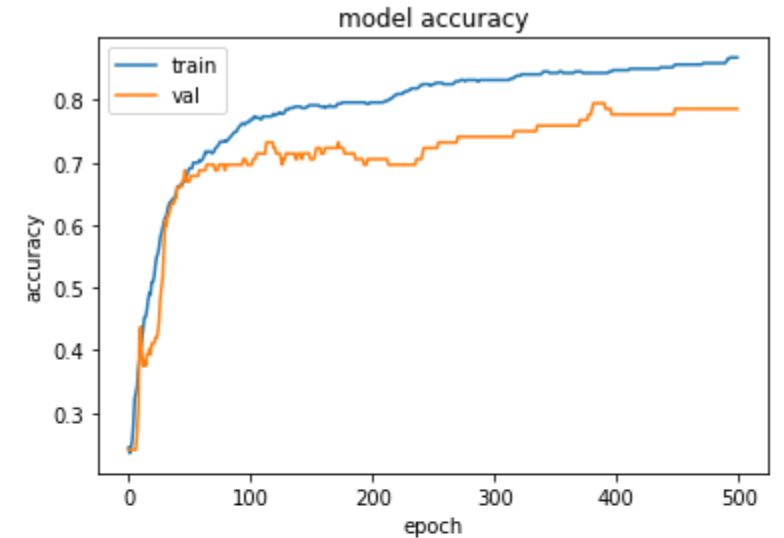
```
Epoch 2/100
```

```
446/446 [=====] - 0s 1ms/step - loss: 0.2446 - accuracy: 0.2354 - mse: 0.2446 - val_loss: 0.3205 - val_accuracy: 0.2411 - v.
```

Plots de evolución del error y accuracy

```
## plots de evolución de loss y accuracy
from matplotlib import pyplot as plt
plt.plot(historico.history['accuracy'])
plt.plot(historico.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

plt.plot(historico.history['loss'])
plt.plot(historico.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



Guardar los datos de entrenamiento en ficheros

- La variable 'histórico' contiene la *loss* y *accuracy* de cada época del entrenamiento.

```
# GUARDAR RESULTADOS EN FICHEROS
# evolución del entrenamiento (80% train 20% val) en la búsqueda de hiperparámetros
# son los datos que se usan para construir los plots
# En este caso, la variable 'historico' contiene los datos del último entrenamiento realizado
np.savetxt('historicoTrainLoss.txt', historico.history['loss'])
np.savetxt('historicoValLoss.txt', historico.history['val_loss'])
np.savetxt('historicoTrainAcc.txt', historico.history['accuracy'])
np.savetxt('historicoValAcc.txt', historico.history['val_accuracy'])
```

Evaluación del mejor modelo

- Se harán pruebas con el conjunto de entrenamiento (que a su vez separa automáticamente una parte para validación) entrenando diferentes modelos con diferentes hiperparámetros.
- Seleccionaremos los hiperparámetros que minimicen la función objetivo en validación (loss: en este caso el mse)

Construir y entrenar el modelo definitivo

- El modelo final se construirá con los mejores hiperparámetros y con todo el conjunto de entrenamiento (sin separar datos para validación)

```
# Construir y entrenar el modelo definitivo
# ejemplo:
num_neuronas=100
lr = 0.2
epochs = 100

#definir modelo
final_model = Sequential()
final_model.add(Dense(num_neuronas, input_shape=input_shape, activation='sigmoid'))
final_model.add(Dense(num_clases, activation='softmax'))
```

Evaluación del mejor modelo

```
# CONFIGURAR MODELO Y ENTRENAMIENTO (TRAINING 100%)

final_model.compile(loss='mean_squared_error',
                    optimizer=tf.keras.optimizers.SGD(learning_rate=lr, momentum=0),
                    metrics=['accuracy', 'mse'] )

history_final_model = final_model.fit(X_train, y_train_transformed, epochs=epochs,
                                     batch_size=1, verbose=1, validation_split=0, shuffle=False)
```

```
Epoch 1/100
558/558 [=====] - 1s 1ms/step - loss: 0.2023 - accuracy: 0.2437 - mse: 0.2023
Epoch 2/100
558/558 [=====] - 1s 1ms/step - loss: 0.1958 - accuracy: 0.2742 - mse: 0.1958
Epoch 3/100
558/558 [=====] - 1s 1ms/step - loss: 0.1900 - accuracy: 0.3226 - mse: 0.1900
```


Evaluación del mejor modelo

- Este modelo final se evaluará con el conjunto de test:


```
# EVALUAR MODELO DEFINITIVO
test_results = final_model.evaluate(X_test, y_test_transformed, verbose=1)
print(f'Test results - Loss: {test_results[0]} - Accuracy: {test_results[1]}')
```

```
9/9 [=====] - 0s 1ms/step - loss: 0.0634 - accuracy: 0.8056 - mse: 0.0634
Test results - Loss: 0.06341784447431564 - Accuracy: 0.8055555820465088
```

Evaluación del mejor modelo

- Además, sobre el conjunto de test se obtendrán:
 - Las PREDICCIONES o salidas de la red ('en bruto')
 - Incluir en la memoria las correspondientes a las primeras instancias (por ejemplo, 5 instancias)


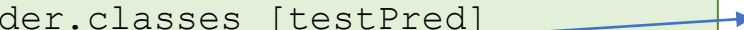
```
# predicciones en bruto
raw_testPred = final_model.predict(X_test)
#prediccion de los 5 primeros patrones de test: 5 vectores con valores reales
print(raw_testPred[:5])
```



```
[[1.16961787e-03 1.13206655e-02 1.97311587e-04 9.87312376e-01]
 [9.49936744e-04 7.96490431e-01 2.02553660e-01 5.89543970e-06]
 [2.90573698e-05 7.93308258e-01 2.04110786e-01 2.55194027e-03]
 [9.70916092e-01 1.67431906e-02 1.11590466e-02 1.18175801e-03]
 [1.66446401e-03 1.21329874e-02 1.69704091e-02 9.69232142e-01]]
```

- Las salidas con la clase asignada
 - Incluir en la memoria las correspondientes a las primeras instancias (por ejemplo, 5 instancias)

```
#predicciones de la clase
testPred = np.argmax(raw_testPred, axis=1)
#transformar el núm de col en la etiqueta
class_testPred = encoder.classes_[testPred]
print(testPred[:5])
print(class_testPred[:5])
```



```
[3 1 1 0 3]
[' van ' ' opel' ' opel' ' bus ' ' van ']
```

Evaluación del mejor modelo

- La matriz de confusión

```
#Confusion Matrix
cm=confusion_matrix(y_test, class_testPred)
print(cm)
```

```
[[73  0  0  2]
 [ 0 62 14  2]
 [ 2 32 31  2]
 [ 1  0  1 66]]
```

- Otras métricas

```
print('Classification Report')
print(classification_report(y_test, class_testPred))
```

Classification Report				
	precision	recall	f1-score	support
bus	0.99	0.99	0.99	75
opel	0.68	0.78	0.73	78
saab	0.68	0.54	0.60	67
van	0.94	0.97	0.96	68
accuracy			0.82	288
macro avg	0.82	0.82	0.82	288
weighted avg	0.82	0.82	0.82	288

Guardar los resultados finales en ficheros

- La evaluación del modelo y la matriz de confusión de las predicciones finales de test.
- El modelo (puede guardarse el modelo completo o solo los pesos)

```
# evaluación del modelo final (loss y accuracy)
np.savetxt('evaluacion.txt', test_results, newline='\n')

# guardar matriz de confusión
np.savetxt('matrizConf.txt', cm, fmt='%-3d')

#guarda el modelo completo
final_model.save('modelo.h5')
#guarda solo pesos
final_model.save_weights('pesos.h5')
```

Se pueden ver los archivos de la carpeta actual. Se pueden descargar en nuestro explorador de ficheros o realizar una copia en Drive

Apéndice

$$precision = \frac{TP}{TP + FP}$$

		predicción	
		0	1
realidad	0	TN	FP
	1	FN	TP

$$recall = \frac{TP}{TP + FN}$$

		predicción	
		0	1
realidad	0	TN	FP
	1	FN	TP

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$