Estudiantes
Julian Leonardo Robles Cabanzo
Diego Fernando Malagon Saenz

5. Ejercicios y Problemas

Ejercicios propuestos

> **Nota**
>
> Todos los códigos de desarrollo del taller se encuentran en Google Colaboratory para ser abiertos mediante el link y también se encuentran en el presente repositorio de Github.

1. **Los archivos que acompañan MEP incluyen conjuntos de entrenamiento para varios ejemplos, tome uno de los ejemplos y utilice los datos para realizar la misma tarea por RNAs.**

Descripción del procedimiento

Se cargó el conjunto de datos `iris` y se dividió en conjuntos de entrenamiento (80 %) y prueba (20 %), escalando las características mediante normalización Z-score. A continuación, se definió una red neuronal de tipo `Sequential` con dos capas ocultas (16 y 8 neuronas, activación ReLU) y una capa de salida con softmax para clasificación en tres categorías. El modelo se compiló con el optimizador Adam y la función de pérdida de entropía cruzada categórica, midiendo precisión como métrica. Se entrenó durante 100 épocas con validación interna (20 % de los datos de entrenamiento), y finalmente se evaluó sobre el conjunto de prueba, obteniéndose un accuracy de 96.67 %.

El código implementado fue el siguiente:

```
# 1. Instalacin de dependencias
!pip install pandas numpy scikit-learn tensorflow

# 2. Carga y visualizacin del dataset
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['species'] = iris.target

print(df.head())
print(df['species'].value_counts())  # 50 de cada clase

# 3. Separar X e y, y codificar
X = iris.data
y = iris.target  # 0,1,2

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
from tensorflow.keras.utils import to_categorical

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

y_train_cat = to_categorical(y_train, num_classes=3)
y_test_cat = to_categorical(y_test, num_classes=3)

# 4. Definir red neuronal
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([
    Dense(16, activation='relu', input_shape=(4,)),
    Dense(8, activation='relu'),
    Dense(3, activation='softmax')  # 3 clases
])

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

# 5. Entrenar
history = model.fit(X_train, y_train_cat,
                    validation_split=0.2,
                    epochs=100, batch_size=8, verbose=1)

# 6. Evaluar en test
loss, acc = model.evaluate(X_test, y_test_cat, verbose=0)
print(f"\nAccuracy en test: {acc:.4f}")

# 7. Probar algunas predicciones
import numpy as np
for i in range(5):
    xi = X_test[i:i+1]
    yi = np.argmax(y_test_cat[i])
    pred = np.argmax(model.predict(xi))
    print(f"Muestras {i}: real={iris.target_names[yi]}, predicci n={iris.
        ↪ target_names[pred]}")
```

El ejercicio demuestró que una red neuronal simple, con dos capas ocultas y softmax en la salida, es capaz de aprender la clasificación del iris con un 96,67 % de precisión, mostrando un rendimiento comparable al de métodos evolutivos como MEP.

El código se encuentra disponible en el siguiente link: `https://colab.research.google.com/drive/1_OPxL0Kkag8JgvfSXVUMLLTUMyZ9mZ38?usp=sharing`

2. **Con base en la librería tensorflow, descargue el data set fashion MNIST. Haga una clasificación de prendas de vestir.**

   En este fragmento, hemos descargado el conjunto de datos Fashion MNIST desde TensorFlow y preparado las imágenes para su uso, ajustando sus valores entre 0 y 1 y adaptándolas al formato necesario para nuestro modelo. A continuación, convertimos las etiquetas originales en un formato que permite distinguir las diez categorías de ropa. Después, construimos una red neuronal que aprende a reconocer patrones en las imágenes mediante varias etapas de procesamiento y reduce los posibles errores con técnicas de regularización. Seguidamente, entrenamos esta red durante quince ciclos, reservando una parte de los datos para validar su desempeño mientras aprende. Por último, evaluamos su precisión sobre datos nuevos y mostramos ejemplos comparando las predicciones del modelo con las etiquetas reales.

```
!pip install pandas numpy scikit-learn tensorflow

# 1. Importar librer as necesarias
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import (
    Dense, Conv2D, MaxPooling2D, Flatten, Dropout
)
from tensorflow.keras.utils import to_categorical

# 2. Descargar el dataset Fashion MNIST desde TensorFlow
#    (60 000  im genes de entrenamiento, 10 000  de prueba; cada imagen 28 28
    ↪ en escala de grises)
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.fashion_mnist.load_data()

# 3. Preprocesamiento
#    a) Normalizar los p xeles a rango [0, 1]
x_train = x_train.astype('float32') / 255.0
x_test  = x_test.astype('float32')  / 255.0

#    b) A adir canal de color para Conv2D: (28,28)     (28,28,1)
x_train = x_train[..., tf.newaxis]
x_test  = x_test[..., tf.newaxis]

#    c) Codificar las etiquetas en  o n e hot  (10 clases)
y_train_cat = to_categorical(y_train, num_classes=10)
y_test_cat  = to_categorical(y_test,  num_classes=10)

# 4. Definir la arquitectura de la red neuronal convolucional
model = Sequential([
    Conv2D(32, kernel_size=(3,3), activation='relu', input_shape=(28,28,1)),
    MaxPooling2D(pool_size=(2,2)),
    Conv2D(64, kernel_size=(3,3), activation='relu'),
    MaxPooling2D(pool_size=(2,2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')  # 10 clases de ropa
])
```

```
# 5. Compilar el modelo
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# 6. Entrenar la red
history = model.fit(
    x_train, y_train_cat,
    epochs=15,
    batch_size=128,
    validation_split=0.1,
    verbose=1
)

# 7. Evaluar en el conjunto de prueba
loss, acc = model.evaluate(x_test, y_test_cat, verbose=0)
print(f'\nPrecisin en test: {acc*100:.2f}%')

# 8. Probar algunas predicciones
class_names = [
    'T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
    'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot'
]
import numpy as np

for i in range(5):
    img = x_test[i:i+1]                  # tomar la  i sima  muestra
    pred = model.predict(img)            # obtener probabilidades
    label = np.argmax(pred, axis=1)[0]   # clase predicha
    real  = y_test[i]                    # etiqueta real
    print(f'Muestra {i}: real = {class_names[real]}, pred = {class_names[label]}')



    # 1. Mostrar la historia de entrenamiento
print(" ltima  p rdida de entrenamiento:", history.history['loss'][-1])
print(" ltima  precisi n de entrenamiento:", history.history['accuracy'][-1])
print(" ltima  p rdida de validaci n:", history.history['val_loss'][-1])
print(" ltima  precisi n de validaci n:", history.history['val_accuracy'][-1])

# 2. Evaluar en test y mostrar m tricas detalladas
loss_test, acc_test = model.evaluate(x_test, y_test_cat, verbose=0)
print(f"\nP rdida en test: {loss_test:.4f}")
print(f"Precisi n en test: {acc_test*100:.2f}%")

# 3. Reporte de clasificaci n (usando sklearn)
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np
```

```
# 3.1. Obtener predicciones de clase
y_pred_probs = model.predict(x_test)
y_pred = np.argmax(y_pred_probs, axis=1)
y_true = np.argmax(y_test_cat, axis=1)

# 3.2. Mostrar informe y matriz de confusi n
print("\nReporte de clasificacin:")
print(classification_report(y_true, y_pred, target_names=class_names))

print("Matriz de confusi n:")
print(confusion_matrix(y_true, y_pred))

# 4. Ejemplos de predicci n con su probabilidad
for i in range(5):
    probs = y_pred_probs[i]
    pred = y_pred[i]
    real = y_true[i]
    print(f"\nMuestra {i}:")
    print(f"  Etiqueta real: {class_names[real]}")
    print(f"  Predicci n:    {class_names[pred]}")
    print(f"  Probabilidades: {np.round(probs, 3)}")
```

A continuación se presenta el desempeño de la red neuronal convolucional en la clasificación del conjunto de datos Fashion MNIST sobre el conjunto de prueba (10000 muestras).

Clase	Precisión	Recall	F1-score	Soporte
T-shirt/top	0.89	0.84	0.86	1000
Trouser	1.00	0.97	0.99	1000
Pullover	0.84	0.90	0.87	1000
Dress	0.91	0.92	0.92	1000
Coat	0.88	0.82	0.85	1000
Sandal	0.98	0.98	0.98	1000
Shirt	0.72	0.77	0.74	1000
Sneaker	0.96	0.96	0.96	1000
Bag	0.98	0.98	0.98	1000
Ankle boot	0.96	0.98	0.97	1000
Accuracy			0.91	
Macro avg	0.91	0.91	0.91	10000
Weighted avg	0.91	0.91	0.91	10000

Tabla 1: Reporte de clasificación por clase, exactitud global y promedios.

El modelo alcanzó una precisión global del 91 % en el conjunto de prueba. Se observa un desempeño particularmente alto en las categorías *Trouser*, *Sandal* y *Bag*, con F1-scores superiores al 0.97. La clase con mayor dificultad fue *Shirt*, reflejando un ligero desfase entre precisión y recall. En conjunto, la CNN demostró un rendimiento sólido y equilibrado en todas las categorías.

El código implementado se encuentra en el siguiente link de Google Colaboratory: `https://colab.research.google.com/drive/1EJcUkaD-IeCaOnp2JdCQCEBfiaHGsNEe?usp=sharing`

3. Consiga un data set de cualquier tipo, estudie sus características (features) y su rótulo. Diseñe una red neuronal y haga ejemplos con base en los pesos aprendidos.

Introducción al análisis

En este fragmento de código, hemos cargado el conjunto de datos Wine desde la librería `sklearn`, que contiene 13 características químicas de tres clases de vino. A continuación, mostramos una vista previa de los datos y describimos sus atributos y etiquetas. Tras ello, normalizamos las características mediante estandarización Z-score y dividimos el conjunto en entrenamiento (80 %) y prueba (20 %). A continuación, definimos y entrenamos una red neuronal multicapa (MLP) con dos capas ocultas de 16 y 8 neuronas respectivamente, usando activación ReLU y un máximo de 500 iteraciones. Una vez entrenada, evaluamos su precisión en el conjunto de prueba, obteniendo un valor cercano al 94.44 %. Por último, extraemos manualmente los pesos y biases de la primera capa para calcular la activación de ReLU de varias muestras de prueba y comparamos estos resultados con las probabilidades y clases predichas por el modelo, mostrando así el funcionamiento interno de la red y su nivel de confianza en cada predicción.

```
import pandas as pd
import numpy as np
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier

# 1. Cargar y explorar el dataset Wine
wine = load_wine()
df = pd.DataFrame(wine.data, columns=wine.feature_names)
df['target'] = wine.target

# Mostrar las primeras filas
print("Vista previa del dataset Wine:")
print(df.head())

# 2. Caractersticas y r tulo
print("\nCaractersticas (features):", wine.feature_names)
print("R tulos (target names):", wine.target_names)

# 3. Preprocesamiento
X = wine.data
y = wine.target

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y
)

# 4. Definir y entrenar la red neuronal (MLPClassifier de scikit-learn)
model = MLPClassifier(hidden_layer_sizes=(16, 8), activation='relu', max_iter=500,
    ↪ random_state=42)
```

```
model.fit(X_train, y_train)

# 5. Evaluacin
acc = model.score(X_test, y_test)
print(f"\nPrecisin en test: {acc*100:.2f}%")

# 6. Ejemplos basados en pesos aprendidos para varias muestras
weights = model.coefs_[0]
biases = model.intercepts_[0]

print("\nEjemplos de salidas de la primera capa y predicciones:")
for sample_idx in range(5):
    x_sample = X_test[sample_idx]
    # Salida manual de la primera capa con ReLU
    first_layer_output = np.maximum(0, np.dot(x_sample, weights) + biases)
    # Prediccin de probabilidades y clase
    probs = model.predict_proba(x_sample.reshape(1, -1))[0]
    pred_class = wine.target_names[np.argmax(probs)]
    real_class = wine.target_names[y_test[sample_idx]]

    print(f"\nMuestra {sample_idx}:")
    print(f"  Salida capa 1 (5 primeros valores): {np.round(first_layer_output[:5],
        ↪ 3)}")
    print(f"  Probabilidades: {np.round(probs, 3)}")
    print(f"  Clase real: {real_class}, Clase predicha: {pred_class}")
```

El conjunto de 13 características describe propiedades químicas del vino (p. ej., contenido de alcohol, acidez, fenoles), y las tres etiquetas (`class_0`, `class_1`, `class_2`) corresponden a las tres variedades. El modelo MLP logró una precisión de 94.44 % en el conjunto de prueba, lo que indica que aprendió eficazmente los patrones distintivos de cada clase.

A continuación se analizan cinco ejemplos de prueba:

- **Muestra 0:** Las primeras cinco activaciones de la capa 1 son altas para las primeras dos neuronas, lo que lleva a una probabilidad de [1,0, 0,0, 0,0] y acierto en `class_0`.
- **Muestra 1:** La activación favorece neuronas asociadas a `class_1`, resultando en probabilidades [0,008, 0,551, 0,440]. El modelo predice `class_1` (0.551) aunque la etiqueta real era `class_2`, reflejando un único error.
- **Muestra 2:** La salida de la capa 1 en las dos primeras neuronas es alta, generando [0,998, 0,002, 0,0] y acierto en `class_0`.
- **Muestra 3:** Las activaciones indican clara preferencia por `class_1`, con probabilidades [0,014, 0,986, 0,0] y acierto en `class_1`.
- **Muestra 4:** La red muestra alta confianza en `class_1` ([0,037, 0,961, 0,002]) y acierta correctamente.

En resumen, las activaciones internas tras la función ReLU se traducen directamente en vectores de probabilidad mediante softmax, lo que permite interpretar el nivel de confianza del modelo y detectar tanto aciertos como el único caso de clasificación errónea.