Ejercicios: Redes Neuronales y Deep Learning

Ejercicio 1: Clasificación Binaria con Perceptrón Multicapa (MLP)

Contexto: Clasificación de correos electrónicos como spam o no spam.

Conceptos clave:

- - Perceptrón multicapa (MLP)
- - Funciones de activación (ReLU, Sigmoid)
- - Función de pérdida binaria (binary_crossentropy)

Descripción del desarrollo:

Modelo MLP con TensorFlow/Keras usando datos simulados con sklearn.

Ejercicio 1

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

X, y = make_classification(n_samples=1000, n_features=20, n_classes=2, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

model = Sequential([
   Dense(16, activation='relu', input_shape=(20,)),
   Dense(8, activation='relu'),
   Dense(1, activation='sigmoid')
])
model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10, validation_split=0.2)
```

Preguntas para reforzar el aprendizaje:

- - ¿Por qué se usa la función sigmoid en la salida?
- - ¿Cómo afecta la arquitectura (número de capas y neuronas) al rendimiento?

Ejercicio 2: Clasificación de Imágenes con CNN (MNIST)

Contexto: Reconocimiento automático de dígitos manuscritos para sistemas bancarios.

Conceptos clave:

- - Redes convolucionales
- - Pooling y flattening
- - Softmax para clasificación multiclase

Descripción del desarrollo:

CNN simple entrenado con el dataset MNIST.

Ejercicio 2

```
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train = X_train.reshape(-1, 28, 28, 1) / 255.0
X_test = X_test.reshape(-1, 28, 28, 1) / 255.0
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

model = Sequential([
   Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
   MaxPooling2D(2,2),
   Flatten(),
   Dense(64, activation='relu'),
   Dense(10, activation='softmax')
])
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=5, validation_split=0.2)
```

Preguntas para reforzar el aprendizaje:

- - ¿Qué ventajas ofrece el uso de convoluciones en vez de capas densas?
- - ¿Qué pasaría si eliminamos la capa de pooling?

Ejercicio 3: Reconocimiento de Patrones Médicos con CNN

Contexto: Clasificación de radiografías pulmonares (neumonía vs normal).

Conceptos clave:

- - CNN en imágenes médicas
- - Clasificación binaria en imágenes
- - Uso de generadores de datos

Descripción del desarrollo:

Uso de ImageDataGenerator para cargar imágenes médicas clasificadas.
https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia

Ejercicio 3

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

train_datagen = ImageDataGenerator(rescale=1./255, horizontal_flip=True, zoom_range=0.2)
train_generator = train_datagen.flow_from_directory(
   'dataset/train', target_size=(150, 150), batch_size=32, class_mode='binary')

model = Sequential([
   Conv2D(32, (3,3), activation='relu', input_shape=(150,150,3)),
   MaxPooling2D(2,2),
   Flatten(),
   Dense(64, activation='relu'),
   Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(train_generator, epochs=10)
```

Preguntas para reforzar el aprendizaje:

- - ¿Qué medidas éticas deben considerarse en el uso de IA médica?
- - ¿Cómo se mejora la precisión usando aumento de datos?

Ejercicio 4: Predicción de Ventas con LSTM

Contexto: Predicción mensual de ventas en un supermercado.

Conceptos clave:

- - Series temporales
- - LSTM para datos secuenciales
- - Normalización de datos

Descripción del desarrollo:

LSTM que predice valores futuros en una serie generada artificialmente.

Ejercicio 4

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense


# Generar datos simulados de ventas (serie temporal)

np.random.seed(0)

data = np.sin(np.linspace(0, 100, 300)) + np.random.normal(0, 0.1, 300)


# Escalar los datos entre 0 y 1

scaler = MinMaxScaler()

scaled_data = scaler.fit_transform(data.reshape(-1, 1))


# Crear ventanas de tiempo (X: 10 pasos anteriores, y: siguiente paso)

X, y = [], []

for i in range(len(scaled_data) - 10):
```

```
    X.append(scaled_data[i:i+10])
    y.append(scaled_data[i+10])
X, y = np.array(X), np.array(y)

# Ajustar la forma para LSTM: (muestras, pasos de tiempo, características)
X = X.reshape((X.shape[0], X.shape[1], 1))

# Crear modelo LSTM
model = Sequential([
    LSTM(50, activation='tanh', input_shape=(10, 1)),
    Dense(1)
])
model.compile(optimizer='adam', loss='mse')
history = model.fit(X, y, epochs=20, batch_size=16, validation_split=0.2)

# Visualizar pérdida
plt.plot(history.history['loss'], label='Entrenamiento')
plt.plot(history.history['val_loss'], label='Validación')
plt.title('Pérdida durante entrenamiento LSTM')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Preguntas para reforzar el aprendizaje:

- - ¿Cuál es la diferencia práctica entre RNN y LSTM?
- - ¿Por qué es importante normalizar series temporales?

Ejercicio 5: Clasificación de Sentimientos con LSTM

Contexto: Análisis de opiniones de clientes en una tienda online (positivo/negativo).

Conceptos clave:

- - Procesamiento secuencial de texto
- - Tokenización y secuencias
- - Embeddings y redes LSTM

Descripción del desarrollo:

Uso de Tokenizer, pad_sequences y modelo LSTM para texto.

Ejercicio 5

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense

texts = ['Me encantó el producto', 'No me gustó nada', 'Excelente servicio', 'Terrible atención']
labels = [1, 0, 1, 0]

tokenizer = Tokenizer(num_words=1000)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
X = pad_sequences(sequences, maxlen=10)
y = np.array(labels)

model = Sequential([
   Embedding(input_dim=1000, output_dim=16, input_length=10),
   LSTM(32),
   Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X, y, epochs=10)
```

Preguntas para reforzar el aprendizaje:

- - ¿Por qué es importante el preprocesamiento del texto?
- - ¿Qué significa la capa Embedding?

Ejercicio 6: Técnicas de Regularización y Validación

Contexto: Evitar el sobreajuste en la predicción de abandono de clientes.

Conceptos clave:

- - Dropout
- - Early Stopping
- - Batch Normalization

Descripción del desarrollo:

Modelo con regularización usando técnicas combinadas en Keras.

Ejercicio 6

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping

model = Sequential([
   Dense(64, activation='relu', input_shape=(20,)),
   BatchNormalization(),
   Dropout(0.3),
   Dense(32, activation='relu'),
   Dropout(0.3),
   Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
early_stop = EarlyStopping(monitor='val_loss', patience=3)
model.fit(X_train, y_train, validation_split=0.2, epochs=20, callbacks=[early_stop])
```

Preguntas para reforzar el aprendizaje:

- - ¿Qué técnica fue más efectiva para evitar el sobreajuste?
- - ¿Qué significa el parámetro patience en EarlyStopping?

Ejercicio 7: Visualización de Función de Pérdida y Activaciones

Contexto: Comprender el comportamiento interno del modelo durante el entrenamiento.

Conceptos clave:

- - Visualización de activaciones
- - Historial de pérdida y precisión
- - Interpretabilidad de modelos

Descripción del desarrollo:

Ejercicio 7

Descripción de desarrollo

Durante el entrenamiento de redes neuronales, es fundamental monitorear el desempeño del modelo y entender el comportamiento interno de las capas. Este ejercicio permite visualizar cómo aprende un modelo, detectar sobreajuste y observar activaciones intermedias.

Requisitos

- TensorFlow / Keras
- Matplotlib
- Numpy
- Dataset simulado (o puede integrarse con ejercicios anteriores)

Código completo paso a paso

```
import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras.models import Sequential, Model

from tensorflow.keras.layers import Dense, Dropout

from tensorflow.keras.optimizers import Adam

from sklearn.datasets import make_classification

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

# 1. Generar datos simulados

X, y = make_classification(n_samples=500, n_features=20, n_classes=2, random_state=42)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

```
# 2. Normalización
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# 3. Construcción del modelo
model = Sequential([
    Dense(32, activation='relu', input_shape=(20,), name="capa_1"),
    Dropout(0.2),
    Dense(16, activation='relu', name="capa_2"),
    Dense(1, activation='sigmoid', name="salida")
])
model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])
# 4. Entrenamiento y visualización de pérdida
history = model.fit(X_train, y_train, validation_split=0.2, epochs=15)
# Gráfico de pérdida
plt.figure(figsize=(8, 4))
plt.plot(history.history['loss'], label='Pérdida Entrenamiento')
plt.plot(history.history['val_loss'], label='Pérdida Validación')
plt.title('Evolución de la Pérdida')
plt.xlabel('Épocas')
plt.ylabel('Pérdida')
plt.legend()
plt.grid(True)
plt.show()
```

```
# 5. Visualización de activaciones internas (solo para una muestra)
from tensorflow.keras import backend as K
# Crear un modelo que devuelva salidas intermedias
layer_outputs = [layer.output for layer in model.layers if 'Dense' in layer.__class__.__name__]
activation_model = Model(inputs=model.input, outputs=layer_outputs)
# Seleccionar una muestra aleatoria
sample = X_test[0].reshape(1, -1)
# Obtener activaciones
activations = activation_model.predict(sample)
# Mostrar activaciones por capa
for i, activation in enumerate(activations):
    plt.figure(figsize=(6, 1))
    plt.title(f"Activación de la capa {i+1}")
    plt.imshow(activation, aspect='auto', cmap='viridis')
    plt.colorbar()
    plt.tight_layout()
    plt.show()
```

Preguntas para reforzar el aprendizaje

1. ¿Qué parones observas en la evolución de la pérdida? ¿Hay indicios de sobreajuste?
2. ¿Cómo interpretas las activaciones internas? ¿Qué ocurre si la entrada cambia?
3. ¿Cuál es el impacto del Dropout en las activaciones?
4. ¿Cómo podrías visualizar activaciones en una red convolucional?

Ejercicio 8: Comparación de Optimizadores y Funciones de Pérdida

Contexto: Determinar qué combinación produce mejores resultados.

Conceptos clave:

- - Adam vs SGD vs RMSprop
- - Funciones de pérdida: MSE, MAE, Categorical CE
- - Evaluación comparativa

Descripción del desarrollo:

Modelo entrenado múltiples veces cambiando optimizador y función de pérdida.

Ejercicio 8

```
import numpy as np

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from sklearn.datasets import make_classification

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler


# Datos simulados

X, y = make_classification(n_samples=1000, n_features=20, n_classes=2, random_state=42)

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

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)

# Combinaciones a evaluar

optimizers = ['adam', 'sgd', 'rmsprop']

losses = ['binary_crossentropy', 'mean_squared_error', 'mean_absolute_error']

results = []
```

```
# Entrenamiento y evaluación
for opt in optimizers:
    for loss_fn in losses:
        model = Sequential([
            Dense(32, activation='relu', input_shape=(20,)),
            Dense(1, activation='sigmoid')
        ])
        model.compile(optimizer=opt, loss=loss_fn, metrics=['accuracy'])
        history = model.fit(X_train, y_train, validation_split=0.2, epochs=10, verbose=0)
        final_val_acc = history.history['val_accuracy'][-1]
        final_val_loss = history.history['val_loss'][-1]
        print(f"Optimizador: {opt}, Pérdida: {loss_fn}, Val_Acc: {final_val_acc:.4f}, Val_Loss: {final_val_loss:.4f}")
        results.append((opt, loss_fn, final_val_acc, final_val_loss))
# Visualización sugerida (requiere pandas, seaborn)
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
df = pd.DataFrame(results, columns=["Optimizador", "Pérdida", "Precisión", "Pérdida final"])
plt.figure(figsize=(10, 5))
sns.barplot(data=df, x="Optimizador", y="Precisión", hue="Pérdida")
plt.title("Comparación de optimizadores y funciones de pérdida")
plt.ylim(0, 1)
plt.show()
```

Preguntas para reforzar el aprendizaje:

- - ¿Qué combinación fue más rápida en converger?
- - ¿Cuál obtuvo mejor precisión final?

Ejercicio 9: Evaluación Avanzada del Modelo

Contexto: Evaluación de un clasificador multiclase aplicado a imágenes.

Conceptos clave:

- - Matriz de confusión
- - Curvas ROC y AUC
- - Precisión, recall y F1-score

Descripción del desarrollo:

Cálculo de métricas con sklearn.metrics y visualización.

Ejercicio 9

```
import numpy as np

from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, roc_curve

from sklearn.preprocessing import label_binarize

import matplotlib.pyplot as plt

import seaborn as sns


# Supongamos que tenemos 10 clases como en MNIST

n_classes = 10


# Suponiendo que y_test es categorical y y_pred_proba es la predicción por clase

y_pred_proba = model.predict(X_test)  # salida softmax

y_pred = np.argmax(y_pred_proba, axis=1)

y_true = np.argmax(y_test, axis=1)


# Matriz de confusión

conf_matrix = confusion_matrix(y_true, y_pred)

plt.figure(figsize=(8,6))
```

```
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')

plt.title('Matriz de Confusión')

plt.xlabel('Predicción')

plt.ylabel('Real')

plt.show()


# Reporte de métricas

print(classification_report(y_true, y_pred))


# ROC AUC multiclase

# Necesitamos binarizar las etiquetas

y_true_bin = label_binarize(y_true, classes=list(range(n_classes)))

y_pred_bin = y_pred_proba  # ya viene como probabilidades por clase


auc_score = roc_auc_score(y_true_bin, y_pred_bin, average='macro', multi_class='ovr')

print("ROC AUC (macro average):", auc_score)
```

Preguntas para reforzar el aprendizaje:

- - ¿Qué clases son más difíciles de predecir?
- - ¿Cómo interpretar un bajo F1-score?