import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from keras.models import Sequential

from keras.layers import \*

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/letter-recognition/letter-recognition.data"

df = pd.read\_csv(url, header=None)

# Preprocess data

X = df.iloc[:, 1:]

X = X / 15.0

y = df.iloc[:, 0]

y = pd.get\_dummies(y)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Define neural network architecture

mmodel = Sequential()

mmodel.add(Dense(64, input\_dim=16, activation='relu'))

mmodel.add(Dense(32, activation='relu'))

mmodel.add(Dense(26, activation='softmax'))

# Compile model

mmodel.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train model

history = mmodel.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test), epochs=100, batch\_size=32)

# Evaluate model

test\_loss, test\_acc = mmodel.evaluate(X\_test, y\_test)

print("Test accuracy:", test\_acc)

# Visualize results

acc = history.history['accuracy']

val\_acc = history.history['val\_accuracy']

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs = range(1, len(acc) + 1)

# Train model

history = mmodel.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test), epochs=100, batch\_size=32)

# Evaluate model

test\_loss, test\_acc = mmodel.evaluate(X\_test, y\_test)

print("Test accuracy:", test\_acc)

# Visualize results

acc = history.history['accuracy']

val\_acc = history.history['val\_accuracy']

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs = range(1, len(acc) + 1)

plt.figure(figsize=(10, 5))

# Plot training and validation Accuracy

plt.plot(epochs, acc, 'bo', label='Training acc')

plt.plot(epochs, val\_acc, 'b', label='Validation acc')

# Plot training and validation loss

plt.plot(epochs, loss, 'bo', label='Training loss',color='red')

plt.plot(epochs, val\_loss, 'b', label='Validation loss',color='red')

plt.title('Training and validation')

plt.xlabel('Epochs')

plt.ylabel('Accuracy/Loss')

plt.legend()

plt.show()

# Make predictions on test data

y\_pred = mmodel.predict(X\_test)

y\_pred = np.argmax(y\_pred, axis=1)

y\_true = np.argmax(np.array(y\_test), axis=1)

# Calculate prediction accuracy

test\_acc = np.mean(y\_pred == y\_true)

print("Test accuracy:", test\_acc)

# Create a confusion matrix

conf\_mat = np.zeros((26, 26), dtype=np.int)

for i in range(len(y\_true)):

  conf\_mat[y\_true[i], y\_pred[i]] += 1

# Visualize prediction accuracy

plt.figure(figsize=(10, 10))

plt.imshow(conf\_mat, cmap='Oranges')

plt.xticks(range(26), [chr(ord('A')+i) for i in range(26)])

plt.yticks(range(26), [chr(ord('A')+i) for i in range(26)])

plt.xlabel('Predicted label')

plt.ylabel('Actual label')

plt.title('Confusion matrix')

for i in range(26):

    for j in range(26):

        plt.text(j, i, conf\_mat[i, j], ha='center', va='center', color='white' if conf\_mat[i, j] > len(y\_true)\*0.05 else 'black')

plt.show()

from tensorflow.keras.datasets import imdb

from tensorflow.keras.preprocessing.sequence import pad\_sequences

# Load the dataset

(X\_train, y\_train), (X\_test, y\_test) = imdb.load\_data(num\_words=10000)

# Pad the sequences to the same length

maxlen = 100

X\_train = pad\_sequences(X\_train, maxlen=maxlen)

X\_test = pad\_sequences(X\_test, maxlen=maxlen)

bmodel = Sequential([

    Embedding(input\_dim=10000, output\_dim=32, input\_length=maxlen),

    Conv1D(32, 7, activation='relu'),

    MaxPooling1D(5),

    Conv1D(32, 7, activation='relu'),

    GlobalMaxPooling1D(),

    Dense(1, activation='sigmoid')

])

bmodel = Sequential([

    Embedding(input\_dim=10000, output\_dim=32, input\_length=maxlen),

    Conv1D(32, 7, activation='relu'),

    MaxPooling1D(5),

    Conv1D(32, 7, activation='relu'),

    GlobalMaxPooling1D(),

    Dense(1, activation='sigmoid')

])

bmodel.compile(

  optimizer='rmsprop',

  loss='binary\_crossentropy',

  metrics=['acc']

)

history = bmodel.fit(

  X\_train, y\_train,

  epochs=16,

  batch\_size=128,

  validation\_split=0.2

)

score = bmodel.evaluate(X\_test, y\_test)

print("Test loss:", score[0])

print("Test accuracy:", score[1])

acc = history.history['acc']

val\_acc = history.history['val\_acc']

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs = range(1, len(acc) + 1)

# Plot the training and validation accuracy

# Plot the training and validation loss

plt.plot(epochs, acc, 'bo', label='Training acc')

plt.plot(epochs, val\_acc, 'b', label='Validation acc')

plt.plot(epochs, loss, 'bo', label='Training loss', color='red')

plt.plot(epochs, val\_loss, 'b', label='Validation Loss', color='red')

plt.title('Training and validation accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()