```
import numpy as np  # --> Import NumPy library for numerical operations
import pandas as pd  # --> Import Pandas library for data handling (not used in code)
import matplotlib.pyplot as plt  # --> Import Matplotlib for plotting graphs
import seaborn as sns  # --> Import Seaborn for advanced plotting (not used in code)
from sklearn.model_selection import train_test_split  # --> Import train_test_split for data splitting (not used)
from keras.datasets import imdb  # --> Import IMDb dataset from Keras
from keras import models, layers  # --> Import model and layer modules from Keras

MAX_WORDS = 8000  # --> Limit vocabulary to top 8000 words to reduce memory usage
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=MAX_WORDS)  # --> Load IMDb dataset with top 8000 words

train_size = len(X_train) // 4  # --> Use only 25% of training data
test_size = len(X_test) // 4  # --> Use only 25% of testing data
X_train = X_train[:train_size]  # --> Slice training features to 25%
y_train = y_train[:train_size]  # --> Slice training labels to match
X_test = X_test[:test_size]  # --> Slice test features to 25%
y_test = y_test[:test_size]  # --> Slice test labels to match

print(f"Using {len(X_train)} training samples and {len(X_test)} test samples")  # --> Print number of samples used

def vectorize(sequences, dimension=MAX_WORDS):  # --> Function to convert word indices to binary vectors
    results = np.zeros((len(sequences), dimension), dtype=np.float32)  # --> Initialize zero matrix with float32 type
    for i, sequence in enumerate(sequences):  # --> Iterate through each sequence
        results[i, [j for j in sequence if j < dimension]] = 1  # --> Set 1s for word indices within vocab range
    return results  # --> Return the binary matrix
```

```
def batch_vectorize(sequences, batch_size=1000, dimension=MAX_WORDS):  # --> Function to vectorize in smaller batches
    results = []  # --> Initialize empty list to store batch results
    for i in range(0, len(sequences), batch_size):  # --> Loop through batches
        batch = sequences[i:i+batch_size]  # --> Slice batch from data
        results.append(vectorize(batch, dimension))  # --> Vectorize and append batch
    return np.concatenate(results)  # --> Merge all batches into one array

print("Vectorizing training data...")  # --> Print message before vectorizing training data
X_train_vec = batch_vectorize(X_train)  # --> Vectorize training data in batches
print("Vectorizing test data...")  # --> Print message before vectorizing test data
X_test_vec = batch_vectorize(X_test)  # --> Vectorize test data in batches

del X_train, X_test  # --> Delete original non-vectorized data to save memory
import gc  # --> Import garbage collector module
gc.collect()  # --> Force garbage collection to free up memory

model = models.Sequential([  # --> Create a sequential neural network model
    layers.Dense(32, activation='relu', input_shape=(MAX_WORDS,)),  # --> Add dense layer with 32 neurons and ReLU
    layers.Dropout(0.3),  # --> Add dropout layer with 30% rate
    layers.Dense(16, activation='relu'),  # --> Add second dense layer with 16 neurons and ReLU
    layers.Dropout(0.2),  # --> Add dropout layer with 20% rate
    layers.Dense(1, activation='sigmoid')  # --> Add output layer with sigmoid for binary classification
])

model.compile(optimizer='adam',  # --> Compile model using Adam optimizer
              loss='binary_crossentropy',  # --> Use binary crossentropy loss for binary classification
              metrics=['accuracy'])  # --> Track accuracy metric during training

history = model.fit(  # --> Train the model and store training history
  X_train_vec, y_train,  # --> Use vectorized training data and labels
  epochs=5,  # --> Train for 5 epochs
  batch_size=256,  # --> Use batch size of 256
  validation_data=(X_test_vec, y_test),  # --> Use test data for validation
  verbose=1  # --> Show progress during training
)

score = model.evaluate(X_test_vec, y_test, verbose=0)  # --> Evaluate model on test data
print(f'Test loss: {score[0]:.4f}')  # --> Print test loss
print(f'Test accuracy: {score[1]:.4f}')  # --> Print test accuracy

plt.figure(figsize=(12, 5))  # --> Create a figure with specified size for plots

plt.subplot(1, 2, 1)  # --> Create first subplot for accuracy
plt.plot(history.history['accuracy'], label='Train')  # --> Plot training accuracy
plt.plot(history.history['val_accuracy'], label='Test')  # --> Plot validation accuracy
plt.title('Model Accuracy')  # --> Set title for accuracy plot
plt.ylabel('Accuracy')  # --> Label y-axis
plt.xlabel('Epoch')  # --> Label x-axis
plt.legend()  # --> Show legend for plot

plt.subplot(1, 2, 2)  # --> Create second subplot for loss
plt.plot(history.history['loss'], label='Train')  # --> Plot training loss
plt.plot(history.history['val_loss'], label='Test')  # --> Plot validation loss
plt.title('Model Loss')  # --> Set title for loss plot
plt.ylabel('Loss')  # --> Label y-axis
plt.xlabel('Epoch')  # --> Label x-axis
plt.legend()  # --> Show legend for plot

plt.tight_layout()  # --> Adjust layout for better spacing
plt.show()  # --> Display the plots
```