ASSIGNMENT 8

# IMDb Review Classification: Feedforward, CNN, RNN, LSTM

In this task, we are going to do sentiment classification on a movie review dataset. We are going to build a feedforward net, a convolutional neural net, a recurrent net and combine one or more of them to understand performance of each of them. A sentence can be thought of as a sequence of words that collectively represent meaning. Individual words impact the meaning. Thus, the context matters; words that occur earlier in the sentence influence the sentence's structure and meaning in the latter part of the sentence (e.g., Jose asked Anqi if she were going to the library today). Likewise, words that occur later in a sentence can affect the meaning of earlier words (e.g., Apple is an interesting company). As we have seen in lecture, if we wish to make use of a full sentence's context in both directions, then we should use a bi-directional RNN (e.g., Bi-LSTM). For the purpose of this tutorial, we are going to

# restrict ourselves to only uni-directional RNNs.

!pip install --upgrade tensorflow import pandas as pd

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, Dense, LSTM, SimpleRNN, Conv1D,

MaxPooling1D, Flatten, Dropout import numpy as np np.random.seed(1) vocabulary\_size = 10000

max\_review\_length = 500

train\_data = pd.read\_csv("train.csv") test\_data = pd.read\_csv("test.csv")

X\_train = train\_data['review'] # Change 'review' to the name of your text column

y\_train = train\_data['sentiment'].map({'positive': 1, 'negative': 0}) # Map sentiment to 1/0

X\_test = test\_data['review']

y\_test = test\_data['sentiment'].map({'positive': 1, 'negative': 0})

tokenizer = Tokenizer(num\_words=vocabulary\_size) tokenizer.fit\_on\_texts(X\_train)

X\_train\_sequences = tokenizer.texts\_to\_sequences(X\_train)

X\_test\_sequences = tokenizer.texts\_to\_sequences(X\_test)

X\_train\_padded = pad\_sequences(X\_train\_sequences, maxlen=max\_review\_length) X\_test\_padded = pad\_sequences(X\_test\_sequences, maxlen=max\_review\_length)

def build\_feedforward\_model(): model = Sequential()

model.add(Embedding(vocabulary\_size, 32, input\_length=max\_review\_length)) model.add(Flatten())

model.add(Dense(250, activation='relu')) model.add(Dropout(0.5)) model.add(Dense(1, activation='sigmoid'))

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

return model

def build\_cnn\_model(): model = Sequential()

model.add(Embedding(vocabulary\_size, 32, input\_length=max\_review\_length)) model.add(Conv1D(32, kernel\_size=3, activation='relu')) model.add(MaxPooling1D(pool\_size=2))

model.add(Flatten())

model.add(Dense(250, activation='relu')) model.add(Dropout(0.5)) model.add(Dense(1, activation='sigmoid'))

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

return model

def build\_rnn\_model(): model = Sequential()

model.add(Embedding(vocabulary\_size, 32, input\_length=max\_review\_length)) model.add(LSTM(100, dropout=0.2, recurrent\_dropout=0.2)) model.add(Dense(1, activation='sigmoid'))

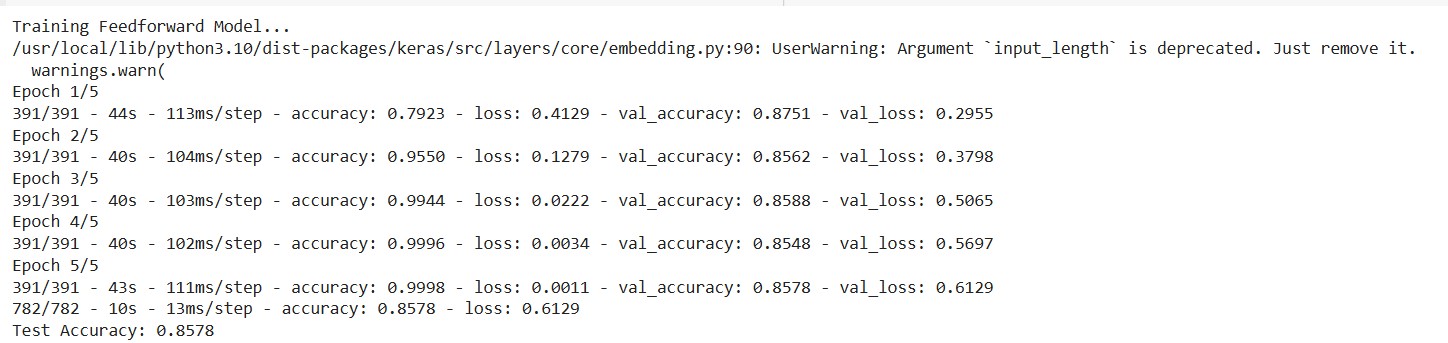
model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

return model

max\_features = vocabulary\_size maxlen = max\_review\_length

def build\_lstm\_model():

model = Sequential()



model.add(Embedding(max\_features, 128, input\_length=maxlen)) model.add(LSTM(128)) # LSTM layer

model.add(Dense(1, activation='sigmoid'))

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

return model

def train\_and\_evaluate\_model(model, X\_train, y\_train, X\_test, y\_test): model.fit(X\_train, y\_train, epochs=5, batch\_size=64,

validation\_data=(X\_test, y\_test), verbose=2)

loss, accuracy = model.evaluate(X\_test, y\_test, verbose=2) print(f"Test Accuracy: {accuracy:.4f}")

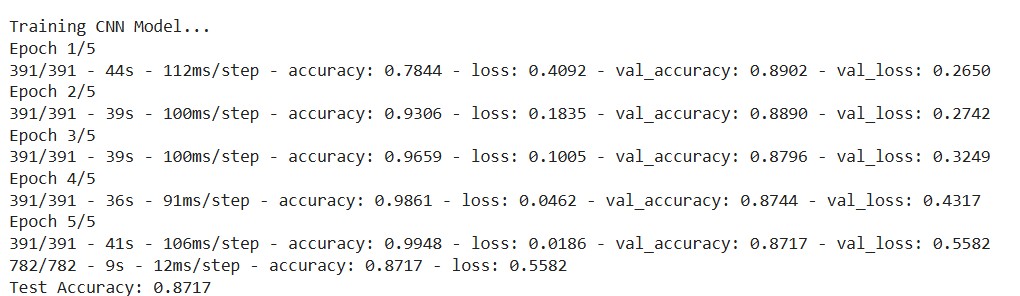
print("Training Feedforward Model...") feedforward\_model = build\_feedforward\_model()

train\_and\_evaluate\_model(feedforward\_model, X\_train\_padded, y\_train, X\_test\_padded, y\_test)

**OUTPUT**

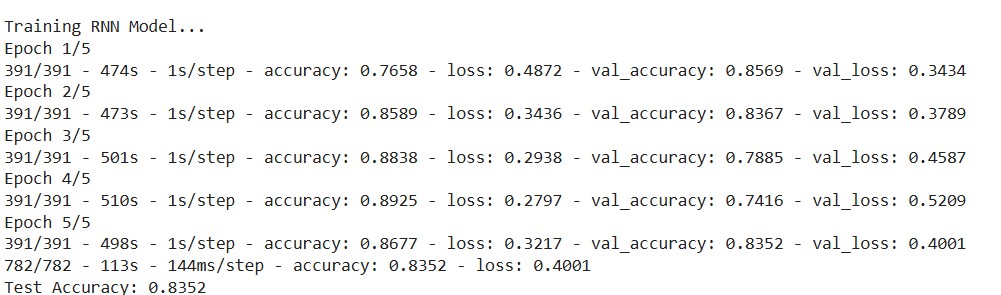
print("\nTraining CNN Model...") cnn\_model = build\_cnn\_model()

train\_and\_evaluate\_model(cnn\_model, X\_train\_padded, y\_train, X\_test\_padded, y\_test)

**OUTPUT**

print("\nTraining RNN Model...") rnn\_model = build\_rnn\_model()

train\_and\_evaluate\_model(rnn\_model, X\_train\_padded, y\_train, X\_test\_padded, y\_test)

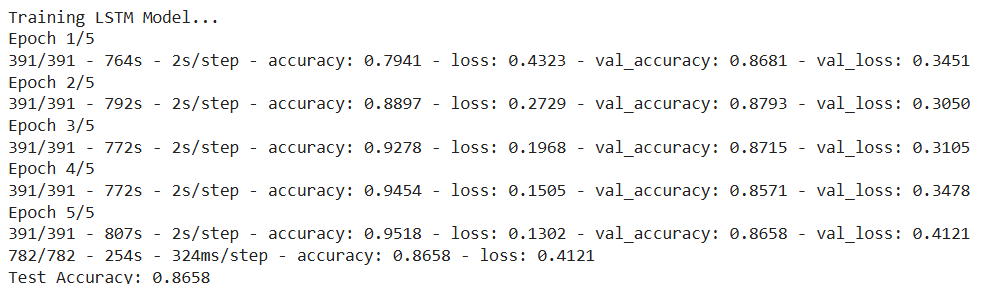


**OUTPUT**

print("\nTraining LSTM Model...") lstm\_model = build\_lstm\_model()

train\_and\_evaluate\_model(lstm\_model, X\_train\_padded, y\_train, X\_test\_padded, y\_test)

**OUTPUT**



***BY: Tanishq Dublish***

***Roll NO.: 102117154  
Group: CS6***