IMDBReviewEx2

```
#import
from keras.datasets import imdb
from keras.preprocessing import sequence
from keras import Sequential
from keras.layers import Embedding, LSTM, Dense, Dropout
import tensorflow as tf
from keras.preprocessing.sequence import pad_sequences

# Load IMDB data and split it into training & testing datasets
vocabulary_size = 5000
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=vocabulary_size)
print('Loaded dataset with {} training samples, {} test samples'.format(len(X_train), len(X_test)))

# Decode the sentences to see the reviews as test
word_index = imdb.get_word_index()
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
decoded_review = ' '.join([reverse_word_index.get(i - 3, '?') for i in X_train[0]])
print(X_train[0])
print(decoded_review)

# Pad the sequences
max_words = 500
X_train = sequence.pad_sequences(X_train, maxlen=max_words)
X_test = sequence.pad_sequences(X_test, maxlen=max_words)
print(X_train[0])

# Initialize the model
embedding_size = 32
model = tf.keras.Sequential()
model.add(tf.keras.layers.Embedding(vocabulary_size, embedding_size, input_length=max_words))
model.add(tf.keras.layers.LSTM(100, return_sequences=True))  # Return sequences for deeper LSTM layers
model.add(tf.keras.layers.Dropout(0.2))  # Adding dropout for regularization
model.add(tf.keras.layers.LSTM(100))
model.add(tf.keras.layers.Dropout(0.2))  # Adding dropout for regularization
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))

# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Split X_train into train and validation datasets
batch_size = 64
X_valid, y_valid = X_train[:batch_size], y_train[:batch_size]
X_train_partial, y_train_partial = X_train[batch_size:], y_train[batch_size:]

# Define a callback for early stopping
class myCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        DESIRED_ACC = 0.94
        if logs.get('val_accuracy') >= DESIRED_ACC:
            print("\nStopping training as validation accuracy reached %.2f!" % DESIRED_ACC)
            self.model.stop_training = True

callbacks = myCallback()

# Fit the model
num_epochs = 20
history = model.fit(X_train_partial, y_train_partial, validation_data=(X_valid, y_valid),
                    batch_size=batch_size, epochs=num_epochs, callbacks=[callbacks])

# Test the model and print the test accuracy score
scores = model.evaluate(X_test, y_test)
print('Test accuracy:', scores[1])
```

```
Epoch 1/20
390/390 [==============================] - 62s 145ms/step - loss: 0.4656 - accuracy: 0.7760 - val_loss: 0.2540 - val_accuracy: 0.
Epoch 2/20
390/390 [==============================] - 31s 81ms/step - loss: 0.3009 - accuracy: 0.8802 - val_loss: 0.3115 - val_accuracy: 0.8
Epoch 3/20
390/390 [==============================] - 22s 58ms/step - loss: 0.3290 - accuracy: 0.8621 - val_loss: 0.6736 - val_accuracy: 0.6
Epoch 4/20
390/390 [==============================] - 18s 47ms/step - loss: 0.6532 - accuracy: 0.5833 - val_loss: 0.6986 - val_accuracy: 0.4
Epoch 5/20
390/390 [==============================] - 19s 50ms/step - loss: 0.4580 - accuracy: 0.7771 - val_loss: 0.2095 - val_accuracy: 0.9
Epoch 6/20
390/390 [==============================] - 19s 48ms/step - loss: 0.2701 - accuracy: 0.8960 - val_loss: 0.2122 - val_accuracy: 0.8
Epoch 7/20
390/390 [==============================] - 18s 45ms/step - loss: 0.2280 - accuracy: 0.9143 - val_loss: 0.1988 - val_accuracy: 0.9
Epoch 8/20
390/390 [==============================] - 18s 47ms/step - loss: 0.2092 - accuracy: 0.9202 - val_loss: 0.1672 - val_accuracy: 0.
Epoch 9/20
390/390 [==============================] - 16s 41ms/step - loss: 0.1871 - accuracy: 0.9312 - val_loss: 0.1952 - val_accuracy: 0.9
Epoch 10/20
390/390 [==============================] - 17s 44ms/step - loss: 0.1749 - accuracy: 0.9360 - val_loss: 0.1799 - val_accuracy: 0.9
Epoch 11/20
390/390 [==============================] - 17s 43ms/step - loss: 0.1603 - accuracy: 0.9427 - val_loss: 0.1950 - val_accuracy: 0.9
Epoch 12/20
390/390 [==============================] - 16s 41ms/step - loss: 0.1421 - accuracy: 0.9502 - val_loss: 0.2386 - val_accuracy: 0.9
Epoch 13/20
390/390 [==============================] - 16s 41ms/step - loss: 0.1244 - accuracy: 0.9593 - val_loss: 0.2205 - val_accuracy: 0.9
Epoch 14/20
390/390 [==============================] - 16s 41ms/step - loss: 0.1065 - accuracy: 0.9660 - val_loss: 0.2581 - val_accuracy: 0.9
Epoch 15/20
390/390 [==============================] - 15s 40ms/step - loss: 0.0921 - accuracy: 0.9722 - val_loss: 0.3507 - val_accuracy: 0.9
Epoch 16/20
390/390 [==============================] - 16s 40ms/step - loss: 0.0753 - accuracy: 0.9786 - val_loss: 0.3434 - val_accuracy: 0.9
Epoch 17/20
390/390 [==============================] - 16s 41ms/step - loss: 0.0676 - accuracy: 0.9808 - val_loss: 0.3732 - val_accuracy: 0.9
Epoch 18/20
390/390 [==============================] - 17s 44ms/step - loss: 0.0638 - accuracy: 0.9824 - val_loss: 0.2970 - val_accuracy: 0.9
Epoch 19/20
390/390 [==============================] - 15s 39ms/step - loss: 0.0505 - accuracy: 0.9880 - val_loss: 0.4462 - val_accuracy: 0.8
Epoch 20/20
390/390 [==============================] - 15s 39ms/step - loss: 0.0513 - accuracy: 0.9869 - val_loss: 0.3978 - val_accuracy: 0.8
782/782 [==============================] - 12s 15ms/step - loss: 0.5335 - accuracy: 0.8609
Test accuracy: 0.8608800172805786
```