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

```
# Load the IMDB Dataset
max_features = 20000 # Numbers of words to consider as features.
max_len = 100 # Cut texts after this number of words (for padding)
batch_size = 32

print('Loading data.....')
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
print(f'{len(x_train)}, train sequences')
print(f'{len(x_test)}, test sequences')
```

```
Loading data.....
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imd
17464789/17464789 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
25000, train sequences
25000, test sequences
```

```
print('Pad sequences (samples x time)')
x_train = pad_sequences(x_train, maxlen=max_len)
x_test = pad_sequences(x_test, maxlen=max_len)
print(f'x_train shape:, {x_train.shape}')
print(f'x_test shape:, {x_test.shape}')
```

```
Pad sequences (samples x time)
x_train shape:, (25000, 100)
x_test shape:, (25000, 100)
```

```
model = Sequential()
model.add(Embedding(max_features, 128, input_length=max_len))
model.add(LSTM(128, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(128))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/embedding.py:90: UserWa
  warnings.warn(
```

```
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
print(model.summary())
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param
embedding (Embedding)	?	0 (unbuilt
lstm (LSTM)	?	0 (unbuilt
dropout (Dropout)	?	0 (unbuilt
lstm_1 (LSTM)	?	0 (unbuilt
dropout_1 (Dropout)	?	0 (unbuilt
dense (Dense)	?	0 (unbuilt

```
 Total params: 0 (0.00 B)
 Trainable params: 0 (0.00 B)
 Non-trainable params: 0 (0.00 B)
None
```

```
print('Training model...')
history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=10,
                    validation_data=(x_test, y_test))
```

```
Training model...
Epoch 1/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 20s 17ms/step - accuracy: 0.7446 - loss: 0.4962 - val_ac
Epoch 2/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 17s 17ms/step - accuracy: 0.9059 - loss: 0.2470 - val_ac
Epoch 3/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 13s 17ms/step - accuracy: 0.9445 - loss: 0.1560 - val_ac
Epoch 4/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 21s 18ms/step - accuracy: 0.9580 - loss: 0.1147 - val_ac
Epoch 5/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 22s 19ms/step - accuracy: 0.9775 - loss: 0.0693 - val_ac
Epoch 6/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 20s 19ms/step - accuracy: 0.9877 - loss: 0.0406 - val_ac
Epoch 7/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 13s 17ms/step - accuracy: 0.9892 - loss: 0.0338 - val_ac
Epoch 8/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 21s 18ms/step - accuracy: 0.9922 - loss: 0.0253 - val_ac
Epoch 9/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 20s 17ms/step - accuracy: 0.9938 - loss: 0.0197 - val_ac
Epoch 10/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 20s 17ms/step - accuracy: 0.9952 - loss: 0.0172 - val_ac
```

```
score, acc = model.evaluate(x_test, y_test, batch_size=batch_size)
print(f'Test score: {score}')
print(f'Test accuracy: {acc}')
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.8184 - loss: 0.7639
Test score: 0.7495042681694031
```

```
Test accuracy: 0.8212400078773499
```

```
# Decode and display review text along with its predicted sentiment
word_index = imdb.get_word_index()
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])

def decode_review(encoded_review):
    return ' '.join([reverse_word_index.get(i - 3, '?') for i in encoded_review])
```

```
# Select a sample review for prediction
sample_review = x_test[2000]
decode_review = decode_review(sample_review)
```

```
# Predict sentiment using the trained LSTM model
prediction = model.predict(np.expand_dims(sample_review, axis=0))
predicted_sentiment = "Positive" if prediction[0][0] > 0.5 else "Negative"

print("\nSample Review:")
print(f"Review: {decode_review}")
print(f"Predicted Sentiment: {predicted_sentiment}")
print(f"Confidence: {prediction[0][0] * 100:.2f}%")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step

Sample Review:
Review: this one brilliantly you can't help but wonder if he is really out there i re
Predicted Sentiment: Positive
Confidence: 99.32%
```

```
Start coding or generate with AI.
```