```
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
from keras.datasets import imdb
import tensorflow as tf
from keras import regularizers
from keras.layers import Bidirectional, GlobalMaxPool1D

tf.random.set_seed(133) # For reproducability and for consistency with the CNN part of the project.

top_words = 10000

(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words = top_words)

max_words = 500
embed_size = 128

x_train = sequence.pad_sequences(x_train, maxlen = 500)
x_test = sequence.pad_sequences(x_test, maxlen = 500)

x_val = x_train[:10000] #setting aside 20% of the x_train for our validation dataset.
partial_x_train = x_train[10000:] # to retrain the model.

y_val = y_train[:10000] # setting aside 20% of the y_train for our validation dataset.
partial_y_train = y_train[10000:]

# Note that the x_train, x_test, y_train, and y_test objects are being imported.
# This is because the IMDB reviews dataset actually has an even 50% split for training and testing datasets.
# It is up to us to build a model capable of high accuracy, but we MUST use validation datasets.
# Some datasets wills have validation datasets available for us to use, that is not the case here.
```

```
from tensorflow.keras.layers import LSTM, Dropout, Embedding

model = Sequential()
model.add(Embedding(top_words, embed_size, input_length = 500))
model.add(Bidirectional(LSTM(units = 128, activation = "tanh")))
model.add(Dropout(0.35))
model.add(Dense(64, activation = "relu"))
model.add(Dense(32, activation = "relu"))
model.add(Dense(units = 1, activation = "sigmoid"))
model.compile(optimizer = "adam",
              loss = "binary_crossentropy",
              metrics = ["accuracy"])

model_tr_2 = model.fit(partial_x_train, partial_y_train, epochs = 10,
                       batch_size = 128, validation_data = (x_val, y_val))

loss = model_tr_2.history["loss"]
val_loss = model_tr_2.history["val_loss"]
epochs = range(1, len(loss) + 1)

import matplotlib.pyplot as plt

plt.plot(epochs, loss, "bo", label = "Training Loss")
plt.plot(epochs, val_loss, "k", label = "Validation Loss")
plt.title("Training & Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()

plt.clf

acc = model_tr_2.history["accuracy"]
val_acc = model_tr_2.history["val_accuracy"]
epochs_f = range(1, len(loss) + 1)
```

```
plt.plot(epochs, acc, "bo", label = "Training Accuracy")
plt.plot(epochs, val_acc, "k", label = "Training & Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()

plt.clf

model.fit(x_train, y_train, epochs = 4, batch_size = 128)
results = model.evaluate(x_test, y_test) # an overall accuracy of 92%
```

```
782/782 [==============================] - 17s 21ms/step - loss: 0.3434 - accuracy: 0.9200
```

In the above chunk, the only thing that we've accomplished of note is the significance of the `validation_split = 0.2` option in the `model.fit()` function. Doing so dramatically increased the performance of the overall model.

I will continue on in this manner, using the same value for the split, but with increased units in the LSTM in the following chunks. The above chunk, LSTM - 1, used 128 units, LSTM - 2 will use 256, and LSTM - 3 will use 512. All with the same values for everything else, other than the dense layer(s).

```
model = Sequential()
model.add(Embedding(top_words, embed_size, input_length = 500))
model.add(Bidirectional(LSTM(units = 256, activation = "tanh")))
model.add(Dropout(0.5))
model.add(Dense(128, activation = "relu"))
model.add(Dense(64, activation = "relu"))
model.add(Dense(32, activation = "relu"))
model.add(Dense(1, activation = "sigmoid"))
model.compile(optimizer = "adam",
              loss = "binary_crossentropy",
              metrics = ["accuracy"])

model_tr_3 = model.fit(partial_x_train, partial_y_train, epochs = 20,
                       batch_size = 256, validation_data = (x_val, y_val))
```

```
loss = model_tr_3.history["loss"]
val_loss = model_tr_3.history["val_loss"]
epochs = range(1, len(loss) + 1)

import matplotlib.pyplot as plt

plt.plot(epochs, loss, "bo", label = "Training Loss")
plt.plot(epochs, val_loss, "k", label = "Validation Loss")
plt.title("Training & Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()

plt.clf

acc = model_tr_3.history["accuracy"]
val_acc = model_tr_3.history["val_accuracy"]
plt.plot(epochs, acc, "bo", label = "Training Accuracy")
plt.plot(epochs, val_acc, "k", label = "Training & Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()

plt.clf

model_tr_3_f = model.fit(partial_x_train, partial_y_train, epochs = 4,
                         batch_size = 256, validation_data = (x_val, y_val))

model.fit(x_train, y_train, epochs = 4, batch_size = 256)

results = model.evaluate(x_test, y_test) # accuracy of 85.15%

     782/782 [==============================] - 19s 24ms/step - loss: 0.5816 - accuracy: 0.8515


model = Sequential()
```

```
model.add(Embedding(top_words, embed_size, input_length = 500))
model.add(Bidirectional(LSTM(units = 512, activation = "tanh")))
model.add(Dropout(0.5))
model.add(Dense(256, activation = "relu"))
model.add(Dense(128, activation = "relu"))
model.add(Dense(1, activation = "sigmoid"))
model.compile(optimizer = "adam",
              loss = "binary_crossentropy",
              metrics = ["accuracy"])

model_tr_4 = model.fit(partial_x_train, partial_y_train, epochs = 20,
                       batch_size = 128, validation_data = (x_val, y_val))

loss = model_tr_4.history["loss"]
val_loss = model_tr_4.history["val_loss"]
epochs = range(1, len(loss) + 1)

plt.plot(epochs, loss, "bo", label = "Training Loss")
plt.plot(epochs, val_loss, "k", label = "Validation Loss")
plt.title("Training & Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()

plt.clf


acc = model_tr_4.history["accuracy"]
val_acc = model_tr_4.history["val_accuracy"]
plt.plot(epochs, acc, "bo", label = "Training Accuracy")
plt.plot(epochs, val_acc, "k", label = "Training & Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()

plt.clf
```

```
Epoch 1/20
118/118 [==============================] - 53s 430ms/step - loss: 0.5853 - accuracy: 0.6751 - val_loss: 0.4778 - val
Epoch 2/20
118/118 [==============================] - 50s 420ms/step - loss: 0.4099 - accuracy: 0.8250 - val_loss: 0.3675 - val
Epoch 3/20
118/118 [==============================] - 50s 422ms/step - loss: 0.2744 - accuracy: 0.8924 - val_loss: 0.4204 - val
Epoch 4/20
118/118 [==============================] - 50s 421ms/step - loss: 0.1807 - accuracy: 0.9348 - val_loss: 0.4054 - val
Epoch 5/20
118/118 [==============================] - 50s 421ms/step - loss: 0.1279 - accuracy: 0.9537 - val_loss: 0.4391 - val
Epoch 6/20
109/118 [==========================>...] - ETA: 3s - loss: 0.0931 - accuracy: 0.9664
```