```
import tensorflow as tf
from tensorflow.keras.preprocessing import sequence
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding, Input, Dropout
from tensorflow.keras.layers import LSTM
from tensorflow.keras.datasets import imdb
import numpy as np
import matplotlib.pyplot as plt
from textblob import TextBlob
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
import tqdm
import pandas as pd


import tensorflow as tf
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

```
Num GPUs Available:  1
```

```
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=50000)

X = np.concatenate((X_train, X_test), axis=0)
y = np.concatenate((y_train, y_test), axis=0)

word_to_id = imdb.get_word_index()
word_to_id = {k:(v) for k,v in word_to_id.items()}
id_to_word = {value:key for key,value in word_to_id.items()}

df = pd.DataFrame(columns=['X', 'Y', 'proportion'])

for i in tqdm.tqdm(range(len(X))):
    positive = 0
    negative = 0

    for word in [id_to_word[id] for id in X[i]]:
        blob = TextBlob(word)
        polarity = blob.sentiment.polarity

        if polarity > 0:
            positive += 1
        else:
            negative += 1

    proportion = positive / negative

    df.loc[len(df)] = [X[i], y[i], proportion]

print(df[:10])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imd
17464789/17464789 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imd
1641221/1641221 [==============================] - 0s 0us/step
100%|██████████| 50000/50000 [22:27<00:00, 37.09it/s]
0  [1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, ...  1    0.038095
1  [1, 194, 1153, 194, 8255, 78, 228, 5, 6, 1463,...  0    0.061798
2  [1, 14, 47, 8, 30, 31, 7, 4, 249, 108, 7, 4, 5...  0    0.052239
3  [1, 4, 18609, 16085, 33, 2804, 4, 2040, 432, 1...  1    0.037736
4  [1, 249, 1323, 7, 61, 113, 10, 10, 13, 1637, 1...  0    0.113636
5  [1, 778, 128, 74, 12, 630, 163, 15, 4, 1766, 7...  0    0.000000
6  [1, 6740, 365, 1234, 5, 1156, 354, 11, 14, 532...  1    0.078947
7  [1, 4, 14906, 716, 4, 65, 7, 4, 689, 4367, 630...  0    0.040741
8  [1, 43, 188, 46, 5, 566, 264, 51, 6, 530, 664,...  1    0.059091
9  [1, 14, 20, 47, 111, 439, 3445, 19, 12, 15, 16...  0    0.083333
```

```
df[:10]
```

	X	Y	proportion
0	[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, ...	1	0.038095
1	[1, 194, 1153, 194, 8255, 78, 228, 5, 6, 1463,...	0	0.061798
2	[1, 14, 47, 8, 30, 31, 7, 4, 249, 108, 7, 4, 5...	0	0.052239
3	[1, 4, 18609, 16085, 33, 2804, 4, 2040, 432, 1...	1	0.037736
4	[1, 249, 1323, 7, 61, 113, 10, 10, 13, 1637, 1...	0	0.113636
5	[1, 778, 128, 74, 12, 630, 163, 15, 4, 1766, 7...	0	0.000000
6	[1, 6740, 365, 1234, 5, 1156, 354, 11, 14, 532...	1	0.078947
7	[1, 4, 14906, 716, 4, 65, 7, 4, 689, 4367, 630...	0	0.040741
8	[1, 43, 188, 46, 5, 566, 264, 51, 6, 530, 664,...	1	0.059091
9	[1, 14, 20, 47, 111, 439, 3445, 19, 12, 15, 16...	0	0.083333

```
X = df['X']
y = df['Y']
proportions = df['proportion']

X = sequence.pad_sequences(X, maxlen=128)

trainSplit = 0.8

proportionsTest = proportions[int(len(proportions) * trainSplit):]
proportions = proportions[:int(len(proportions) * trainSplit)]
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=(1 - trainSplit), rando


proportionInput = Input(shape=(1,))
proportion = Dense(1, activation='sigmoid')(proportionInput)

reviewInput = Input(shape=(128,))
embedding = Embedding(50000, 128, input_length=128)(reviewInput)
lstm = LSTM(64, dropout=0.2, return_sequences=True, kernel_regularizer=tf.keras.regularizer
dropout = Dropout(0.2)(lstm)
lstm = LSTM(64, dropout=0.2, kernel_regularizer=tf.keras.regularizers.l1(0.01))(dropout)

concat = tf.keras.layers.concatenate([lstm, proportion])
dense = Dense(1, activation='sigmoid')(concat)

model = tf.keras.Model(inputs=[reviewInput, proportionInput], outputs=dense)
model.summary()
```

```
Model: "model_8"
__________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==========================================================================================
 input_22 (InputLayer)       [(None, 128)]             0         []

 embedding_10 (Embedding)    (None, 128, 128)          6400000   ['input_22[0][0]'

 lstm_22 (LSTM)              (None, 128, 64)           49408     ['embedding_10[0]

 dropout_14 (Dropout)        (None, 128, 64)           0         ['lstm_22[0][0]']

 input_21 (InputLayer)       [(None, 1)]               0         []

 lstm_23 (LSTM)              (None, 64)                33024     ['dropout_14[0][0

 dense_18 (Dense)            (None, 1)                 2         ['input_21[0][0]'

 concatenate_8 (Concatenate  (None, 65)                0         ['lstm_23[0][0]',
 )                                                               'dense_18[0][0]'

 dense_19 (Dense)            (None, 1)                 66        ['concatenate_8[0

==========================================================================================
Total params: 6482500 (24.73 MB)
Trainable params: 6482500 (24.73 MB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________
```

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

```
model.fit(
    [X_train, np.array(proportions)],
    y_train,
    batch_size=64,
    epochs=5,
    validation_data=([X_test, np.array(proportionsTest)], y_test)
)
```

```
Epoch 1/5
625/625 [==============================] - 56s 82ms/step - loss: 2.8611 - accuracy: 0
Epoch 2/5
625/625 [==============================] - 17s 27ms/step - loss: 0.3664 - accuracy: 0
Epoch 3/5
625/625 [==============================] - 12s 19ms/step - loss: 0.2864 - accuracy: 0
Epoch 4/5
625/625 [==============================] - 12s 19ms/step - loss: 0.2382 - accuracy: 0
Epoch 5/5
625/625 [==============================] - 11s 17ms/step - loss: 0.2012 - accuracy: 0
<keras.src.callbacks.History at 0x7882fcf2fb50>
```

```
loss, acc = model.evaluate([X_test, np.array(proportionsTest)], y_test, batch_size=32)
print('Test Accuracy: %f' % (acc*100))
```

```
313/313 [==============================] - 2s 7ms/step - loss: 0.5137 - accuracy: 0.8
Test Accuracy: 84.979999
```