```
!pip install contractions
!pip install nltk
!pip install autocorrect
```

```
Requirement already satisfied: contractions in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: textsearch>=0.0.21 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: anyascii in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: pyahocorasick in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: nltk in /usr/local/lib/python3.7/dist-packages (3.2.5)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from nltk
Requirement already satisfied: autocorrect in /usr/local/lib/python3.7/dist-packages (2
```

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
import os
import array
import torch #pytorch
import torch.nn as nn #for our model class
import torch.nn.functional as F
from collections import Counter #counting the unique numbers
import string # from some string manipulation tasks
from torch.utils.data import TensorDataset, DataLoader #data prep
import nltk # natural language toolkit
import re # regex
from string import punctuation # solving punctuation problems
from nltk.corpus import stopwords # stop words in sentences
from nltk.stem import WordNetLemmatizer # For stemming the sentence
from nltk.stem import SnowballStemmer # For stemming the sentence
from nltk.tokenize import word_tokenize
from contractions import contractions_dict # to solve contractions
from autocorrect import Speller #correcting the spellings
import pylab as pl
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow import keras
from tensorflow.keras.preprocessing.text import Tokenizer
from keras.utils.vis_utils import plot_model
from tensorflow.keras import layers
from keras.layers import Dense
from keras.models import Sequential
import IPython.display as display
from PIL import Image
nltk.download('stopwords')
nltk.download('punkt')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
```

```
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

```
# seeing what each line looks like

with open('amazon_cells_labelled.txt') as f:
  lines = f.readlines() #create list of all lines in file
  print(lines)
```

```
['So there is no way for me to plug it in here in the US unless I go by a converter.\t0
```

```
lines[0]
```

```
def split(list_a, chunk_size):

  for i in range(0, len(list_a), chunk_size):
    yield list_a[i:i + chunk_size]

chunk_size = 1
new_lines = list(split(lines, chunk_size))
print(new_lines)
```

```
[['So there is no way for me to plug it in here in the US unless I go by a converter.\t
```

```
test_lines = [i.split('\n', 1)[0] for i in lines]
test_lines = [w.replace('\t', ' ') for w in test_lines]
val = test_lines[0]
val = val[-1:]
val
```

```
x = test_lines[0]
x = x[:-2]
test_lines[0] = x
test_lines[0]
```

```
print(test_lines[0], val)
```

```
So there is no way for me to plug it in here in the US unless I go by a converter. 0
```

```
am_df = pd.DataFrame(columns = ['review', 'pos_neg'])
print(am_df)
```

```
Empty DataFrame
Columns: [review, pos_neg]
Index: []
```

```
test_lines = [i.split('\n', 1)[0] for i in lines]
test_lines = [w.replace('\t', ' ') for w in test_lines]
arr = []

for i,s in enumerate(test_lines):
  val = test_lines[i]
  y = val[-1:]
  int(y)
  arr.append(y)
  x = test_lines[i]
  x = x[:-2]
  test_lines[i] = x
  #print(test_lines[i], val)

am_df['review'] = test_lines
am_df['pos_neg'] = arr
#am_df['pos/neg'] = val
print(am_df.head(10))
```

	review	pos_neg
0	So there is no way for me to plug it in here i...	0
1	Good case, Excellent value.	1
2	Great for the jawbone.	1
3	Tied to charger for conversations lasting more...	0
4	The mic is great.	1
5	I have to jiggle the plug to get it to line up...	0
6	If you have several dozen or several hundred c...	0
7	If you are Razr owner...you must have this!	1
8	Needless to say, I wasted my money.	0
9	What a waste of money and time!.	0

```
am_df.to_csv('am_df.csv')
```

```
with open('imdb_labelled.txt') as f:
  lines = f.readlines() #create list of all lines in file

chunk_size = 1
new_lines = list(split(lines, chunk_size))
imdb_df = pd.DataFrame(columns = ['review', 'pos_neg'])
test_lines = [i.split('\n', 1)[0] for i in lines]
test_lines = [w.replace('\t', ' ') for w in test_lines]
arr = []
```

```
for i,s in enumerate(test_lines):
  val = test_lines[i]
  y = val[-1:]
  int(y)
  arr.append(y)
  x = test_lines[i]
  x = x[:-2]
  test_lines[i] = x
  #print(test_lines[i], val)

imdb_df['review'] = test_lines
imdb_df['pos_neg'] = arr
imdb_df.to_csv('imdb_df.csv')
print(imdb_df.head(10))
```

```
                                              review pos_neg
0  A very, very, very slow-moving, aimless movie ...       0
1  Not sure who was more lost - the flat characte...       0
2  Attempting artiness with black & white and cle...       0
3       Very little music or anything to speak of.         0
4  The best scene in the movie was when Gerardo i...       1
5  The rest of the movie lacks art, charm, meanin...       0
6                                Wasted two hours.         0
7  Saw the movie today and thought it was a good ...       1
8                               A bit predictable.         0
9  Loved the casting of Jimmy Buffet as the scien...       1
```

```
with open('yelp_labelled.txt') as f:
  lines = f.readlines() #create list of all lines in file

chunk_size = 1
new_lines = list(split(lines, chunk_size))
yelp_df = pd.DataFrame(columns = ['review', 'pos_neg'])
test_lines = [i.split('\n', 1)[0] for i in lines]
test_lines = [w.replace('\t', ' ') for w in test_lines]
arr = []
for i,s in enumerate(test_lines):
  val = test_lines[i]
  y = val[-1:]
  int(y)
  arr.append(y)
  x = test_lines[i]
  x = x[:-2]
  test_lines[i] = x
  #print(test_lines[i], val)

yelp_df['review'] = test_lines
yelp_df['pos_neg'] = arr
yelp_df.to_csv('yelp_df.csv')
print(yelp_df.head(10))
```

```
                                              review pos_neg
0                          Wow... Loved this place.       1
1                                Crust is not good.       0
2          Not tasty and the texture was just nasty.       0
3  Stopped by during the late May bank holiday of...       1
4  The selection on the menu was great and so wer...       1
5     Now I am getting angry and I want my damn pho.       0
6              Honeslty it didn't taste THAT fresh.)       0
7  The potatoes were like rubber and you could te...       0
8                          The fries were great too.       1
9                                   A great touch.       1
```

```
combo = [am_df, imdb_df, yelp_df]
combo_df = pd.concat(combo)
combo_df.to_csv('combo_df.csv')
```

```
print(combo_df.shape)
print(combo_df.isnull().sum())
```

```
(3000, 2)
review     0
pos_neg    0
dtype: int64
```

```
#from gensim.parsing.preprocessing import remove_stopwords
from nltk.corpus import stopwords
from collections import Counter
from itertools import islice

stop = stopwords.words('english') # stop words in english
print(type(stop))

tokenizer = Tokenizer(num_words = 3000)
tokenizer.fit_on_texts(combo_df['review'])
word_index = tokenizer.word_index
print(word_index) # dict type

for i in stop:
  if i in word_index:
    word_index.pop(i)

#word_index = word_index.pop(stop)
print(word_index)
print(len(word_index))

def take(n, iterable):
    "Return first n items of the iterable as a list"
    return list(islice(iterable, n))

# grabbing top 1000 words to tokenize
```

```
n = 1000
word_index = take(n, word_index.items())
print(word_index)
print(len(word_index))
```

```
<class 'list'>
{'the': 1, 'and': 2, 'i': 3, 'a': 4, 'is': 5, 'it': 6, 'to': 7, 'this': 8, 'of': 9, 'wa
{'good': 18, 'great': 20, 'movie': 24, 'phone': 28, 'film': 29, 'one': 32, 'food': 35,
5127
[('good', 18), ('great', 20), ('movie', 24), ('phone', 28), ('film', 29), ('one', 32),
1000
```

```
def count_word(file_name):
  with open(file_name) as f:
    return Counter(f.read().split())
```

```
combo_df['review'] = combo_df['review'].apply(lambda x: ' '.join([word for word in x.split()
combo_df.to_csv('combo_df.csv')

print(combo_df.head())
print(count_word('combo_df.csv'))
```

```
                                          review pos_neg
0              So way plug US unless I go converter.       0
1                        Good case, Excellent value.       1
2                                     Great jawbone.       1
3  Tied charger conversations lasting 45 minutes....       0
4                                     The mic great.       1
Counter({'I': 453, 'good': 151, 'movie': 129, 'like': 120, 'one': 113, 'phone': 108, 'f
```

```
print(combo_df.iloc[1125][0])
```

```
10/10
```

```
combo_df = pd.read_csv('combo_df.csv')
combo_df = combo_df.drop(columns=['Unnamed: 0'])
print(combo_df)
```

```
                                          review  pos_neg
0              So way plug US unless I go converter.        0
1                        Good case, Excellent value.        1
2                                     Great jawbone.        1
3     Tied charger conversations lasting 45 minutes....        0
4                                     The mic great.        1
...                                              ...      ...
2995               I think food flavor texture lacking.        0
```

```
2996                          Appetite instantly gone.       0
2997                Overall I impressed would go back.       0
2998  The whole experience underwhelming, I think we...       0
2999  Then, I wasted enough life there, poured salt ...       0

[3000 rows x 2 columns]
```

```
sentences = combo_df['review'].tolist()
sequences = tokenizer.texts_to_sequences(sentences)
padded = pad_sequences(sequences)
print(word_index)
print(sequences)
print(padded)
print(type(padded))
np.savetxt("token.csv", padded, delimiter=",") # print padded/tokenized data into file
```

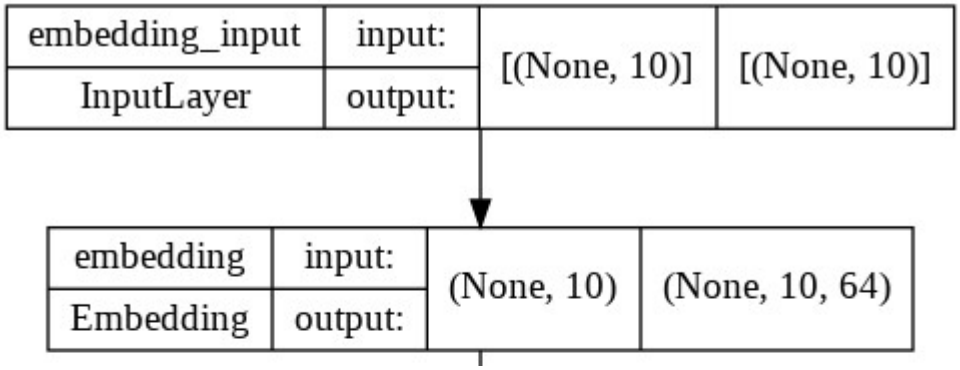
```
[('good', 18), ('great', 20), ('movie', 24), ('phone', 28), ('film', 29), ('one', 32),
[[116, 370, 186, 578, 75, 2267], [18, 157, 91, 526], [20, 1093], [2268, 241, 1094, 1095
[[   0    0    0 ...  578   75 2267]
 [   0    0    0 ...  157   91  526]
 [   0    0    0 ...    0   20 1093]
 ...
 [   0    0    0 ...   54   75   69]
 [   0    0    0 ...  368  299   40]
 [   0    0    0 ...  396  627  488]]
<class 'numpy.ndarray'>
```

```
vocab_size = 1000 # number of words in vocab

model_flatten = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, 64, input_length=10),
    tf.keras.layers.Flatten(),
    #tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(6, activation = 'relu'), # rectified linear unit activation functic
    tf.keras.layers.Dense(1, activation = 'sigmoid') # sigmoid activation function
])

model_global = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, 64, input_length=10),
    #tf.keras.layers.Flatten(),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(6, activation = 'relu'), # rectified linear unit activation functic
    tf.keras.layers.Dense(1, activation = 'sigmoid') # sigmoid activation function
])

model_flatten.summary()
model_global.summary()
```

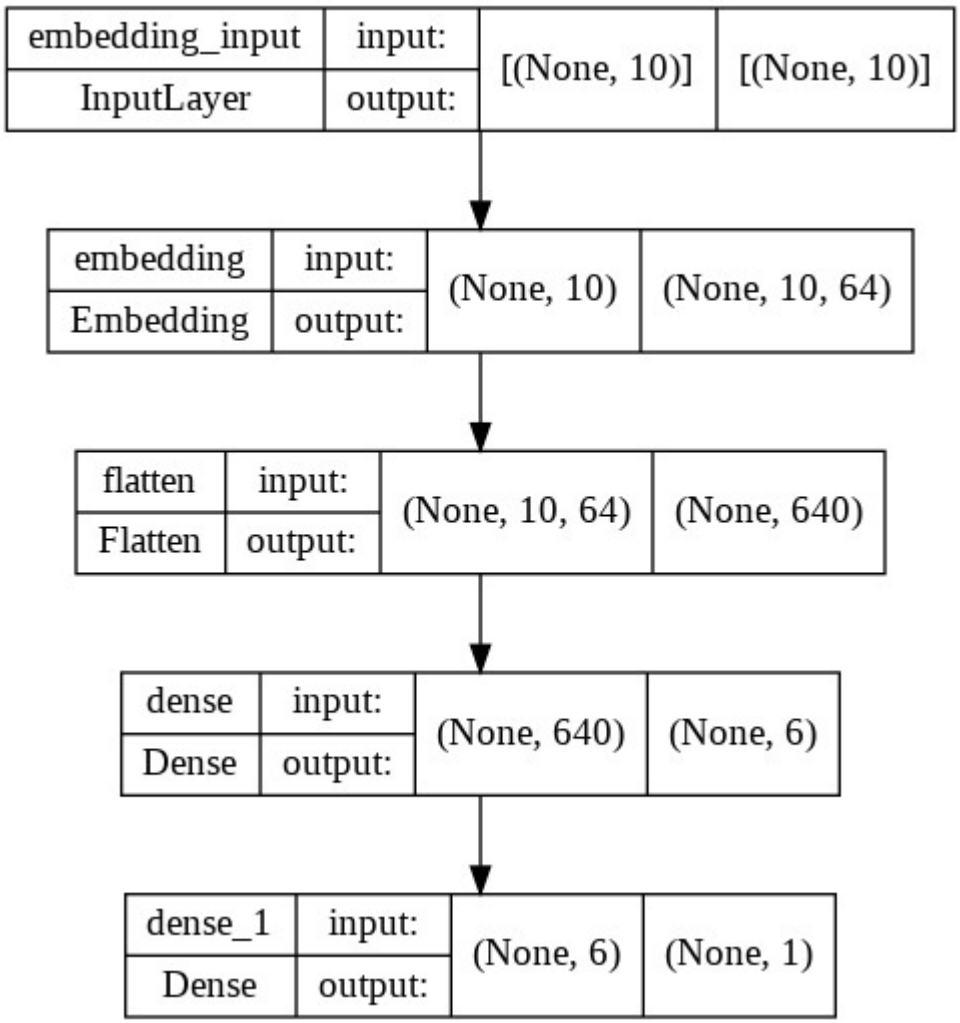
```
Model: "sequential"
_________________________________________________________________
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 10, 64)            64000

 flatten (Flatten)           (None, 640)               0

 dense (Dense)               (None, 6)                 3846

 dense_1 (Dense)             (None, 1)                 7

=================================================================
Total params: 67,853
Trainable params: 67,853
Non-trainable params: 0
_________________________________________________________________
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 10, 64)            64000

 global_average_pooling1d (G  (None, 64)               0
 lobalAveragePooling1D)

 dense_2 (Dense)             (None, 6)                 390

 dense_3 (Dense)             (None, 1)                 7

=================================================================
Total params: 64,397
Trainable params: 64,397
Non-trainable params: 0
_________________________________________________________________
```

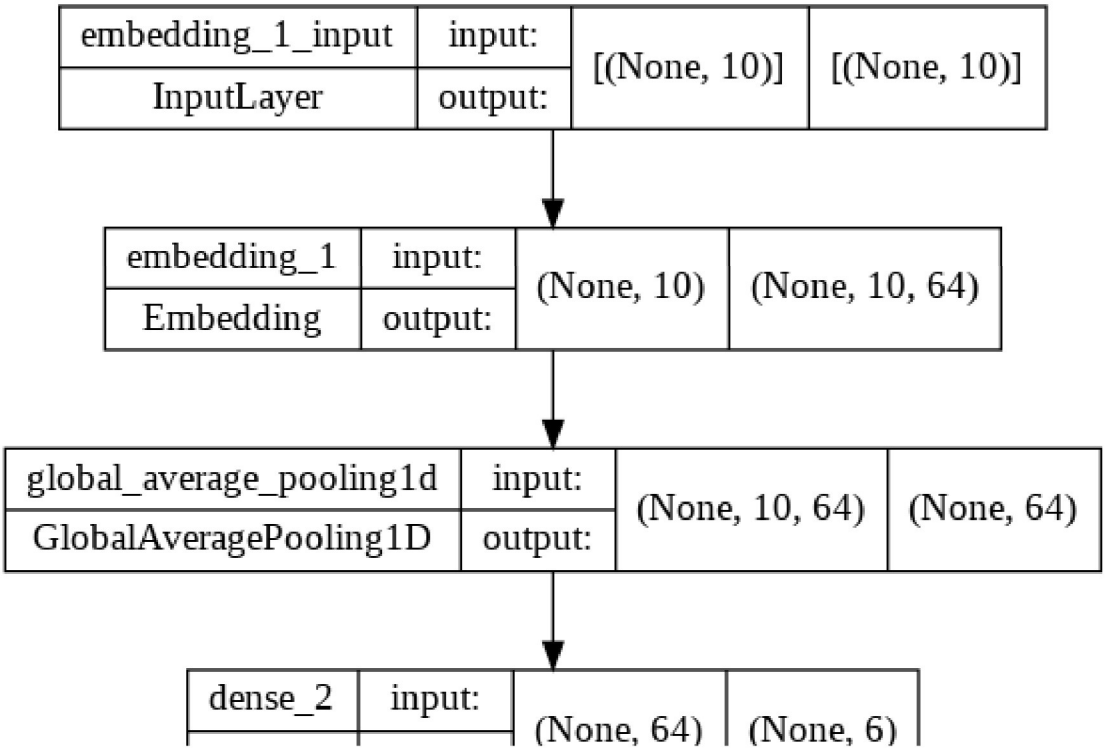
```
plot_model(model_flatten, show_shapes=True, to_file='flatten_model.png', show_layer_names=Tru
```



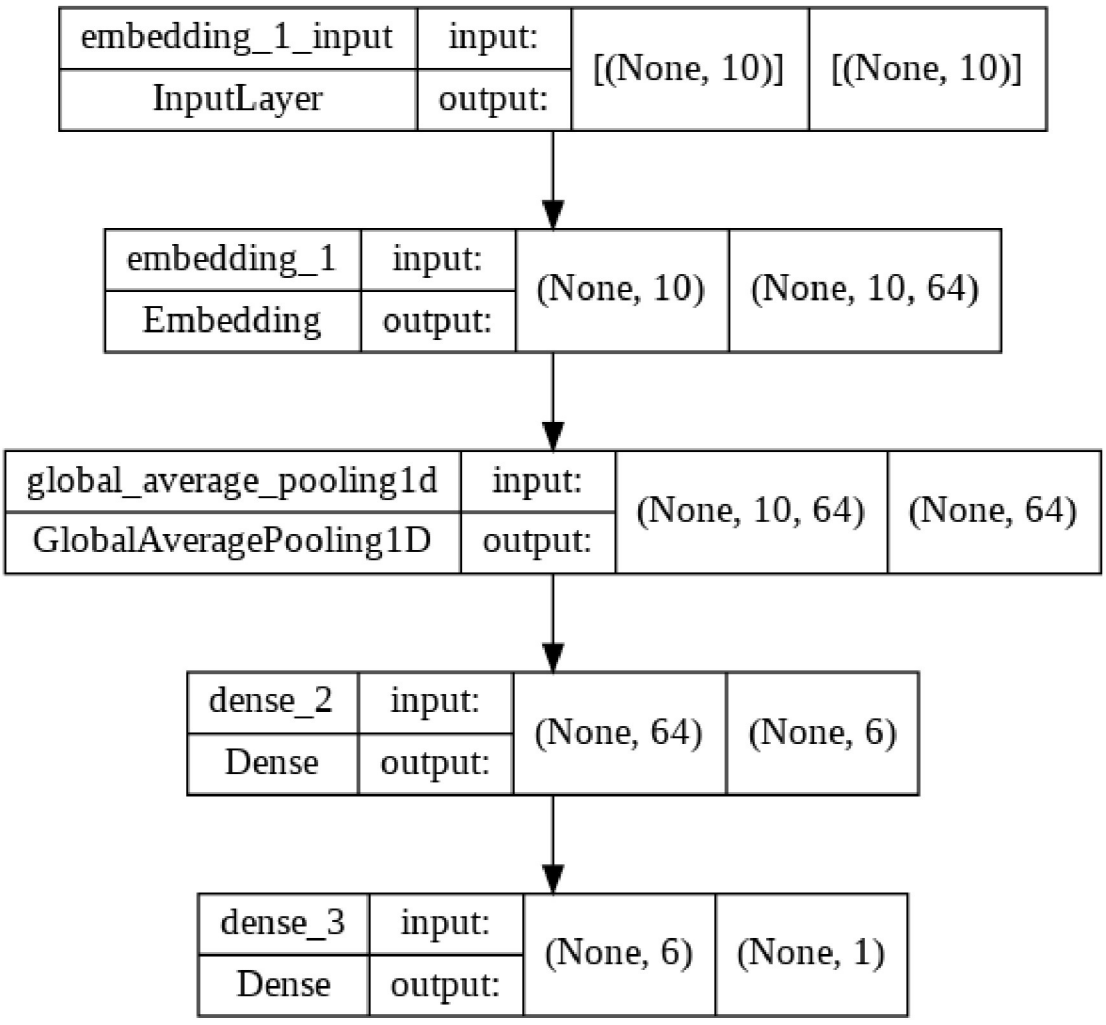

```
display.display(Image.open('flatten_model.png'))
```




```
plot_model(model_global, show_shapes=True, to_file='global_model.png', show_layer_names=True)
```




```
display.display(Image.open('global_model.png'))
```



```
X = combo_df['review']
y = combo_df['pos_neg']
print(X.shape, y.shape)

combo_test_df = combo_df.sample(frac=0.067)
combo_train_df = combo_df.sample(frac=0.267)
combo_test_df.to_csv('combo_test_df.csv')
combo_train_df.to_csv('combo_train_df.csv')
# need to print these into separate CSV files

print(combo_test_df.shape)
print(combo_train_df.shape)
```

```
(3000,) (3000,)
(201, 2)
(801, 2)
```

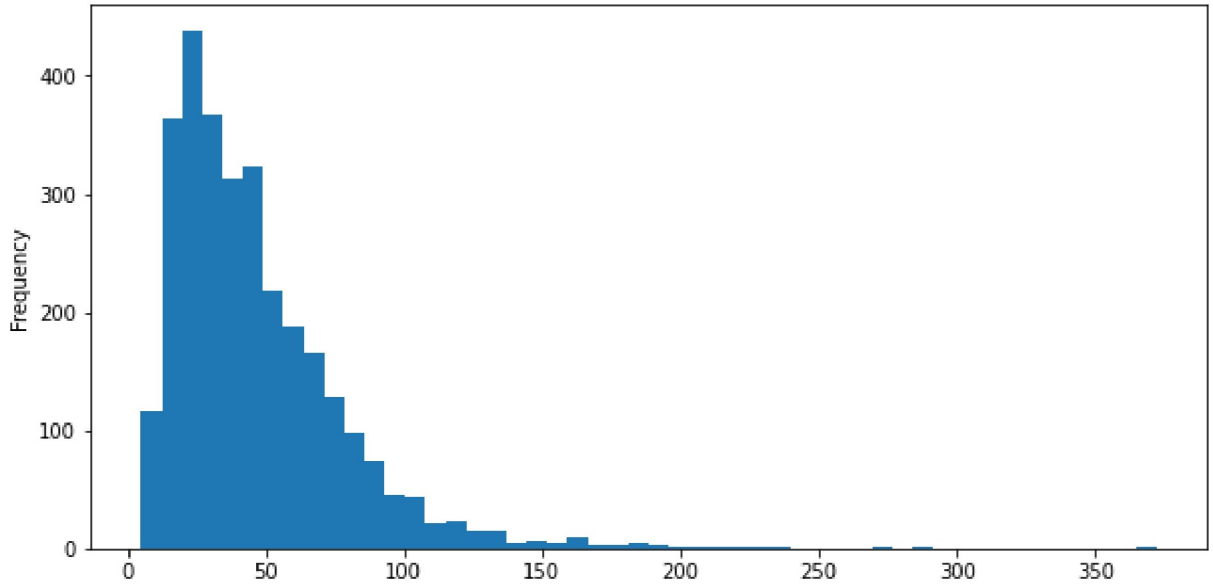
```
print(X.head)
print(y.head)
```

```
<bound method NDFrame.head of 0                    So way plug US unless I go converter.
1                              Good case, Excellent value.
2                                           Great jawbone.
3       Tied charger conversations lasting 45 minutes....
4                                           The mic great.
                               ...
2995                I think food flavor texture lacking.
2996                             Appetite instantly gone.
2997                  Overall I impressed would go back.
2998    The whole experience underwhelming, I think we...
2999    Then, I wasted enough life there, poured salt ...
Name: review, Length: 3000, dtype: object>
<bound method NDFrame.head of 0       0
1       1
2       1
3       0
4       1
       ..
2995    0
2996    0
2997    0
2998    0
2999    0
Name: pos_neg, Length: 3000, dtype: int64>
```

```
combo_df['length'] = combo_df['review'].apply(len)
combo_df['length'].plot(kind = 'hist', bins = 50, figsize = (10,5))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0198dddad0>
```


```
display.display(Image.open('length.png'))
```




```
ax = combo_df.hist(column = 'length', by = 'pos_neg', bins = 50, figsize = (10,10))
pl.suptitle('length via pos_neg')
```

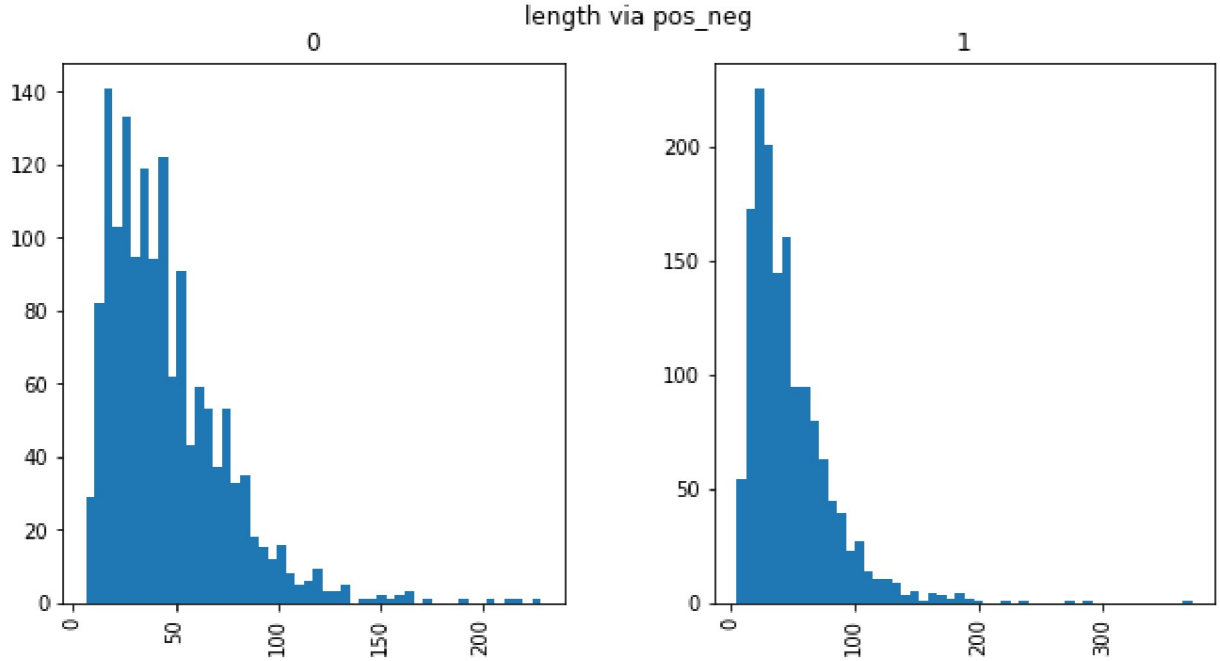
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0198dddad0>
```


```
#display.display(Image.open('length.png'))
```

```
0 50 100 150 200 250 300 350
```

```
ax = combo_df.hist(column = 'length', by = 'pos_neg', bins = 50, figsize = (10,5))
pl.suptitle('length via pos_neg')
```

```
Text(0.5, 0.98, 'length via pos_neg')
```




```
#display.display(Image.open('length pos neg.png'))
```

```
# X is review, y is pos_neg
print(X.head(), y.head())
print(X.shape, y.shape)
```

```
0                So way plug US unless I go converter.
1                          Good case, Excellent value.
2                                       Great jawbone.
3    Tied charger conversations lasting 45 minutes....
```

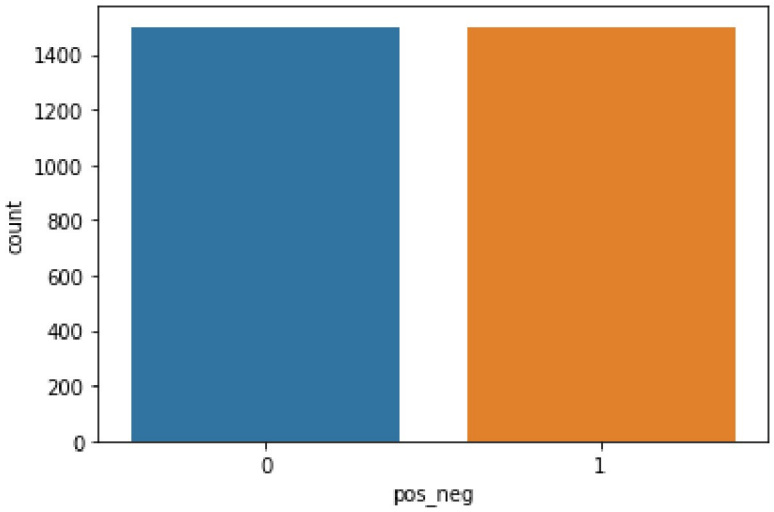
```
4                                   The mic great.
Name: review, dtype: object 0    0
1    1
2    1
3    0
4    1
Name: pos_neg, dtype: int64
(3000,) (3000,)
```

```
X_train,x_test,y_train,y_test = train_test_split(X,y,stratify=y)
print(X_train.shape, y_train.shape)
print(X_train.head(), y_train.head())

#combo_test_df.to_csv('combo_test_df.csv')
#combo_train_df.to_csv('combo_train_df.csv')
```

```
(2250,) (2250,)
1999      All insult one's intelligence huge waste money.
1477                            The directing sloppy best.
2651                Great place relax awesome burger beer.
2666                                  The staff attentive.
744     If like loud buzzing override conversations, p...
Name: review, dtype: object 1999    0
1477    0
2651    1
2666    1
744     0
Name: pos_neg, dtype: int64
```

```
sns.countplot(combo_df['pos_neg'])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass t
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f01985bdf90>
```




```
#display.display(Image.open('pos neg.png'))
```

```
# need to clean up the text
def string_fix(s):
  s = re.sub(r"[^\w\s]", '', s) # remove single characters not in word
  s = re.sub(r"\s+", '', s) # replace whitespace with no space
  s = re.sub(r"\d", '', s) # replaces digits with no space
  return s


# "padded" is the padded combo_df file

train_data = TensorDataset(torch.from_numpy(padded), torch.from_numpy(padded))
valid_data = TensorDataset(torch.from_numpy(padded), torch.from_numpy(padded))

print(type(train_data), type(valid_data))
print(train_data)
print(valid_data)
```

```
<class 'torch.utils.data.dataset.TensorDataset'> <class 'torch.utils.data.dataset.Tenso
<torch.utils.data.dataset.TensorDataset object at 0x7f01984d16d0>
<torch.utils.data.dataset.TensorDataset object at 0x7f0198bf55d0>
```

```
# commented out neural net to avoid long calculation

class SentimentRNN(nn.Module):
    def __init__(self,no_layers,vocab_size,hidden_dim,embedding_dim,drop_prob=0.5):
        super(SentimentRNN,self).__init__()

        self.output_dim = output_dim
        self.hidden_dim = hidden_dim

        self.no_layers = no_layers
        self.vocab_size = vocab_size

        # embedding and LSTM layers
        self.embedding = nn.Embedding(vocab_size, embedding_dim)

        #lstm
        self.lstm = nn.LSTM(input_size=embedding_dim, hidden_size=self.hidden_dim, num_layers

        # dropout layer
        self.dropout = nn.Dropout(0.3)

        # linear and sigmoid layer
        self.fc = nn.Linear(self.hidden_dim, output_dim)
        self.sig = nn.Sigmoid()

    def forward(self,x,hidden):
        batch_size = x.size(0)
        # embeddings and lstm_out
        embeds = self.embedding(x)  # shape: B x S x Feature   since batch = True
```

```
            #print(embeds.shape)  #[50, 500, 1000]
            lstm_out, hidden = self.lstm(embeds, hidden)

            lstm_out = lstm_out.contiguous().view(-1, self.hidden_dim)

            # dropout and fully connected layer
            out = self.dropout(lstm_out)
            out = self.fc(out)

            # sigmoid function
            sig_out = self.sig(out)

            # reshape to be batch_size first
            sig_out = sig_out.view(batch_size, -1)

            sig_out = sig_out[:, -1] # get last batch of labels

            # return last sigmoid output and hidden state
            return sig_out, hidden

        def init_hidden(self, batch_size):
            # Initializes hidden state
            # Create two new tensors with sizes n_layers x batch_size x hidden_dim,
            # initialized to zero, for hidden state and cell state of LSTM
            h0 = torch.zeros((self.no_layers,batch_size,self.hidden_dim)).to(device)
            c0 = torch.zeros((self.no_layers,batch_size,self.hidden_dim)).to(device)
            hidden = (h0,c0)
            return hidden



    is_cuda = torch.cuda.is_available()

    # If we have a GPU available, we'll set our device to GPU. We'll use this device variable lat
    if is_cuda:
        device = torch.device("cuda")
        print("use gpu")
    else:
        device = torch.device("cpu")
        print("use cpu")


         use cpu


    no_layers = 1
    vocab_size = len(padded) + 1 #extra 1 for padding
    embedding_dim = 64
    output_dim = 1
    hidden_dim = 256
```

```
model = SentimentRNN(no_layers,vocab_size,hidden_dim,embedding_dim,drop_prob=0.5)

#moving to gpu
model.to(device)

print(model)
```

```
SentimentRNN(
  (embedding): Embedding(3001, 64)
  (lstm): LSTM(64, 256, batch_first=True)
  (dropout): Dropout(p=0.3, inplace=False)
  (fc): Linear(in_features=256, out_features=1, bias=True)
  (sig): Sigmoid()
)
```

```
# loss and optimization functions
lr=0.001

criterion = nn.BCELoss()

optimizer = torch.optim.Adam(model.parameters(), lr=lr)

# function to predict accuracy
def acc(pred,label):
    pred = torch.round(pred.squeeze())
    return torch.sum(pred == label.squeeze()).item()
```

```
'''
clip = 5
epochs = 5
batch_size = 50
train_loader = DataLoader(train_data, shuffle=True, batch_size=batch_size)
valid_loader = DataLoader(valid_data, shuffle=True, batch_size=batch_size)
labels = combo_df['pos_neg']

valid_loss_min = np.Inf
# train for some number of epochs
epoch_tr_loss,epoch_vl_loss = [],[]
epoch_tr_acc,epoch_vl_acc = [],[]

for epoch in range(epochs):
    train_losses = []
    train_acc = 0.0
    model.train()
    # initialize hidden state
    h = model.init_hidden(batch_size)
    for inputs, labels in train_loader:
```

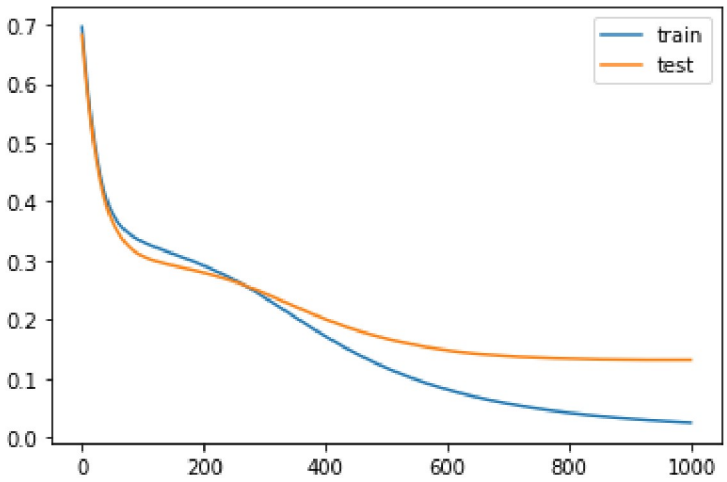
```
        inputs, labels = inputs.to(device), labels.to(device)
        # Creating new variables for the hidden state, otherwise
        # we'd backprop through the entire training history
        h = tuple([each.data for each in h])

        model.zero_grad()
        output,h = model(inputs,h)

        # calculate the loss and perform backprop
        #output = output.unsqueeze(-1) # -1 stands for last here equivalent to 1
        print(type(labels))
        loss = criterion(output.squeeze(), labels)
        loss.backward()
        train_losses.append(loss.item())
        # calculating accuracy
        accuracy = acc(output,labels)
        train_acc += accuracy
        #`clip_grad_norm` helps prevent the exploding gradient problem in RNNs / LSTMs.
        nn.utils.clip_grad_norm_(model.parameters(), clip)
        optimizer.step()



    val_h = model.init_hidden(batch_size)
    val_losses = []
    val_acc = 0.0
    model.eval()
    for inputs, labels in valid_loader:
            val_h = tuple([each.data for each in val_h])

            inputs, labels = inputs.to(device), labels.to(device)

            output, val_h = model(inputs, val_h)
            val_loss = criterion(output.squeeze(), labels.float())

            val_losses.append(val_loss.item())

            accuracy = acc(output,labels)
            val_acc += accuracy

    epoch_train_loss = np.mean(train_losses)
    epoch_val_loss = np.mean(val_losses)
    epoch_train_acc = train_acc/len(train_loader.dataset)
    epoch_val_acc = val_acc/len(valid_loader.dataset)
    epoch_tr_loss.append(epoch_train_loss)
    epoch_vl_loss.append(epoch_val_loss)
    epoch_tr_acc.append(epoch_train_acc)
    epoch_vl_acc.append(epoch_val_acc)
    print(f'Epoch {epoch+1}')
    print(f'train_loss : {epoch_train_loss} val_loss : {epoch_val_loss}')
```

```
        print(f'train_accuracy : {epoch_train_acc*100} val_accuracy : {epoch_val_acc*100}')
        if epoch_val_loss <= valid_loss_min:
            torch.save(model.state_dict(), '../working/state_dict.pt')
            print('Validation loss decreased ({:.6f} --> {:.6f}).  Saving model ...'.format(valic
            valid_loss_min = epoch_val_loss
        print(25*'==')
    '''
```

```
    '''
    #from sklearn.datasets import make_moons
    # generate 2d classification dataset
    # X, y = make_moons(n_samples=1000, noise=0.2, random_state=1) # using
    # split into train and test
    n_train = 30
    trainX, testX = X[:n_train, :], X[n_train:, :]
    trainy, testy = y[:n_train], y[n_train:]
    # define model
    model = Sequential()
    model.add(Dense(500, input_dim=2, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    # fit model
    history = model.fit(trainX, trainy, validation_data=(testX, testy), epochs=1000, verbose=1)
    # evaluate the model
    _, train_acc = model.evaluate(trainX, trainy, verbose=0)
    _, test_acc = model.evaluate(testX, testy, verbose=0)
    print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))
    # plot training history
    plt.plot(history.history['loss'], label='train')
    plt.plot(history.history['val_loss'], label='test')
    plt.legend()
    plt.show()
    '''
```

```
'\n#from sklearn.datasets import make_moons\n# generate 2d classification dataset\n#
X, y = make_moons(n_samples=1000, noise=0.2, random_state=1) # using \n# split into tr
ain and test\nn_train = 30\ntrainX, testX = X[:n_train, :], X[n_train:, :]\ntrainy, te
sty = y[:n_train], y[n_train:]\n# define model\nmodel = Sequential()\nmodel.add(Dense
```

```
# display charts and epochs

display.display(Image.open('test 2.png'))
display.display(Image.open('first 5 epochs.PNG'))
```




```
Epoch 1/1000
1/1 [==============================] - 2s 2s/step - loss: 0.6989 - accuracy: 0.5000 - val_loss: 0.6836 - val_accuracy: 0.7247
Epoch 2/1000
1/1 [==============================] - 0s 76ms/step - loss: 0.6866 - accuracy: 0.6667 - val_loss: 0.6718 - val_accuracy: 0.8371
Epoch 3/1000
1/1 [==============================] - 0s 74ms/step - loss: 0.6747 - accuracy: 0.8000 - val_loss: 0.6602 - val_accuracy: 0.8474
Epoch 4/1000
1/1 [==============================] - 0s 83ms/step - loss: 0.6630 - accuracy: 0.8000 - val_loss: 0.6489 - val_accuracy: 0.8330
Epoch 5/1000
1/1 [==============================] - 0s 73ms/step - loss: 0.6517 - accuracy: 0.8000 - val_loss: 0.6379 - val_accuracy: 0.8278
```