Goal: Explore PyTorch library for deep learning model creation and explore nlp techniques for Sent

Dataset:https://www.kaggle.com/nicapotato/womens-ecommerce-clothing-reviews

```
import pandas as pd
import numpy as np
import torch
from google.colab import drive
```

```
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

```
dataset = pd.read_csv("/content/drive/My Drive/pytorch_tutorials/PyTorch Sentiment Analysi
```

```
dataset.head()
```

	Unnamed: 0	Clothing ID	Age	Title	Review Text	Ratin
0	0	767	33	NaN	Absolutely wonderful - silky and sexy and comf...	
1	1	1080	34	NaN	Love this dress! it's sooo pretty. i happene...	
2	2	1077	60	Some major design flaws	I had such high hopes for this dress and reall...	
3	3	1049	50	My favorite buy!	I love, love, love this jumpsuit. it's fun, fl...	
4	4	847	47	Flattering shirt	This shirt is very flattering to all due to th...	

As we are concerned about only text data analysis,we'll ignore all other columns and keep 'Review

```
dataset = dataset[['Review Text','Recommended IND']]
```

```
dataset =  dataset.rename(columns={'Review Text':'review','Recommended IND':'recommended'}
```

```
dataset.head()
```

	review	recommended
0	Absolutely wonderful - silky and sexy and comf...	1
1	Love this dress! it's sooo pretty. i happene...	1
2	I had such high hopes for this dress and reall...	0
3	I love, love, love this jumpsuit. it's fun, fl...	1
4	This shirt is very flattering to all due to th...	1

```
dataset.shape
```

```
(23486, 2)
```

```
dataset.review.isnull().sum()
```

```
845
```

```
dataset = dataset.dropna(axis=0, subset=['review'])
```

There are multiple ways to convert Text to numbers/vectors, we"ll stick to basics and explore worc
excercise is checking PyTorch for Deep Learning model building. Here is the link to basic word em
https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Shuffle the data and then split it, keeping 20% aside for testing
X_train, X_test, y_train, y_test = train_test_split(dataset['review'], dataset['recommende

vectorizer = CountVectorizer(lowercase=True)
vectorizer.fit(dataset['review'])

X_train_vec = vectorizer.transform(X_train)
X_test_vec = vectorizer.transform(X_test)


print(X_train_vec.shape)
print(X_test_vec.shape)
```

```
(18112, 14145)
(4529, 14145)
```

```
classifier = LogisticRegression()
classifier.fit(X_train_vec, y_train)

print("Score:", classifier.score(X_test_vec, y_test))
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:432: FutureWa
  FutureWarning)
Score: 0.8904835504526386
```

We now know that vocabulary size is 14145.

```
'''Now let's convert dataset to Tensors'''
X_train_tensor = torch.from_numpy(X_train_vec.todense()).float()
X_test_tensor = torch.from_numpy(X_test_vec.todense()).float()
Y_train_tensor = torch.from_numpy(np.array(y_train))
Y_test_tensor = torch.from_numpy(np.array(y_test))
```

```
'''Now let's build basic network and try to get prediction without training model'''

class Network(torch.nn.Module):
  def __init__(self,vocab_size,out_classes):
    super().__init__()
    self.linear = torch.nn.Linear(vocab_size,out_classes)

  def forward(self,x):
    return self.linear(x)

#Declare dimensions
VOCAB_SIZE = 14145
OUT_CLASSES = 1

#Initialize model
model = Network(VOCAB_SIZE,OUT_CLASSES)

#Prediction without training
'''CountVectorizer has given sparse matrix,first we have to convert it to Dense matrix.'''
pred = model(X_train_tensor[1])

print(pred)
```

```
tensor([-0.0228], grad_fn=<AddBackward0>)
```

```
'''Now let's do the training,But before we do the training we have to create batches of tr
```

```
"Now let's do the training,But before we do the training we have to create batches of
```

```
'''We'll use DataLoader class for batching but it requires TensorDataset'''
from torch.utils.data import Dataset, TensorDataset


train_data = TensorDataset(X_train_tensor, Y_train_tensor)
```

```
'''TensorDataset creates list of tuples with each record containing feature_tuple and labl
train_data[0]
```

```
(tensor([0., 0., 0.,  ..., 0., 0., 0.]), tensor(1))
```

```
'''Let's use DataLoader'''
from torch.utils.data import DataLoader
train_loader = DataLoader(train_data,batch_size=16, shuffle=True)


'''This object/train_loader is iterable'''
next(iter(train_loader))
```

```
[tensor([[0., 0., 0.,  ..., 0., 0., 0.],
         [0., 0., 0.,  ..., 0., 0., 0.],
         [0., 0., 0.,  ..., 0., 0., 0.],
         ...,
         [0., 0., 0.,  ..., 0., 0., 0.],
         [0., 0., 0.,  ..., 0., 0., 0.],
         [0., 0., 0.,  ..., 0., 0., 0.]]),
 tensor([0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])]
```

```
'''Now let's do the training'''
'''But before that we have to update our Network,because earlier we've created basic liner
either 0 or 1,so let's add sigmoid layer and also few hidden layers'''
class Network(torch.nn.Module):
  def __init__(self,vocab_size,hidden_units,num_classes):
    super().__init__()
    self.fc1 = torch.nn.Linear(vocab_size,hidden_units)
    self.fc2 = torch.nn.Linear(hidden_units,num_classes)
    self.output = torch.nn.Sigmoid()

  def forward(self,x):
    fc1 = self.fc1(x)
    fc2 = self.fc2(fc1)
    output = self.output(fc2)
    return output[:, -1]




NUM_EPOCHS = 5
VOCAB_SIZE = 14145
HIDDEN_UNITS = 3
OUT_CLASSES = 1
LEARNING_RATE = 0.001

#Initialize model
model = Network(VOCAB_SIZE,HIDDEN_UNITS,OUT_CLASSES)
print(model)
#Initialize optimizer
optimizer =torch.optim.SGD(model.parameters(), lr=LEARNING_RATE)
#Initialize loss function
loss_fun = torch.nn.BCELoss()

for i in range(NUM_EPOCHS):
  for x_batch,y_batch in train_loader:
    model.train()
    y_pred = model(x_batch)
    loss = loss_fun(y_pred,y_batch.float())
    loss.backward()
    loss.backward()
    optimizer.step()
    optimizer.zero_grad()

  print('After {} epoch training loss is {}'.format(i,loss.item()))
```

```
Network(
  (fc1): Linear(in_features=14145, out_features=3, bias=True)
  (fc2): Linear(in_features=3, out_features=1, bias=True)
  (output): Sigmoid()
)
After 0 epoch training loss is 0.3262462913990021
After 1 epoch training loss is 0.560406506061554
After 2 epoch training loss is 0.3553839921951294
After 3 epoch training loss is 0.6066318154335022
After 4 epoch training loss is 0.33251193165779114
```

sigmoid(0.999)) tensor(0.3135) ``` Args: ```html weight (Tensor, optional): a manual rescaling weight given to the