```
import pandas as pd
import numpy as np
import collections
import re
```

```
f=open("/content/t1.txt")
doc1=f.read()
f.close()
```

```
doc1
```

```
'Ram raced to the grocery store. Ram went inside but realized he forgot his wallet. Ram raced back home to grab it. Once Ram found i
```

```
l_doc1 = re.sub(r"[^a-zA-Z0-9]", " ", doc1.lower()).split()
```

```
l_doc1
```

```
['ram',
 'raced',
 'to',
 'the',
 'grocery',
 'store',
 'ram',
 'went',
 'inside',
 'but',
 'realized',
 'he',
 'forgot',
 'his',
 'wallet',
 'ram',
 'raced',
 'back',
 'home',
 'to',
 'grab',
 'it',
 'once',
 'ram',
 'found',
 'it',
 'ram',
 'raced',
 'to',
 'the',
 'car',
 'again',
 'and',
 'drove',
 'back',
 'to',
 'the',
 'grocery',
 'store']
```

```
l=l_doc1
```

```
l
```

```
['ram',
 'raced',
 'to',
 'the',
 'grocery',
 'store',
 'ram',
 'went',
 'inside',
 'but',
 'realized',
 'he',
 'forgot',
 'his',
 'wallet',
 'ram',
 'raced',
 'back',
 'home',
```

```
 'to',
 'grab',
 'it',
 'once',
 'ram',
 'found',
 'it',
 'ram',
 'raced',
 'to',
 'the',
 'car',
 'again',
 'and',
 'drove',
 'back',
 'to',
 'the',
 'grocery',
 'store']
```

```
wordset=set(l)
```

```
wordset
```

```
{'again',
 'and',
 'back',
 'but',
 'car',
 'drove',
 'forgot',
 'found',
 'grab',
 'grocery',
 'he',
 'his',
 'home',
 'inside',
 'it',
 'once',
 'raced',
 'ram',
 'realized',
 'store',
 'the',
 'to',
 'wallet',
 'went'}
```

```
def calculateBOW(wordset,l_doc):
  tf_diz = dict.fromkeys(wordset,0)
  for word in l_doc:
      tf_diz[word]=l_doc.count(word)
  return tf_diz
```

```
bow1 = calculateBOW(wordset,l_doc1)
df_bow = pd.DataFrame([bow1])
df_bow.head()
```

	grab	forgot	back	once	grocery	found	home	store	but	wallet	...	to	again	and	went	realized	it	raced	his	he	drove
0	1	1	2	1	2	1	1	2	1	1	...	4	1	1	1	1	2	3	1	1	1

1 rows × 24 columns

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
```

```
X = vectorizer.fit_transform([doc1])
df_bow_sklearn = pd.DataFrame(X.toarray(),columns=vectorizer.get_feature_names_out())
df_bow_sklearn.head()
```

	again	and	back	but	car	drove	forgot	found	grab	grocery	...	it	once	raced	ram	realized	store	the	to	wallet	went
0	1	1	2	1	1	1	1	1	1	2	...	2	1	3	5	1	2	3	4	1	1

1 rows × 24 columns

```
print(vectorizer.get_feature_names_out())
```

```
['again' 'and' 'back' 'but' 'car' 'drove' 'forgot' 'found' 'grab'
 'grocery' 'he' 'his' 'home' 'inside' 'it' 'once' 'raced' 'ram' 'realized'
 'store' 'the' 'to' 'wallet' 'went']
```

```
import nltk
nltk.download('punkt')
import re
import numpy as np

f= open("/content/t1.txt")
text=f.read()
f.close()
dataset = nltk.sent_tokenize(text)
for i in range(len(dataset)):
    dataset[i] = dataset[i].lower()
    dataset[i] = re.sub(r'\W', ' ', dataset[i])
    dataset[i] = re.sub(r'\s+', ' ', dataset[i])
```

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

```
print(dataset)
```

```
['ram raced to the grocery store ', 'ram went inside but realized he forgot his wallet ', 'ram raced back home to grab it ', 'once r
```

```
word2count = {}
for data in dataset:
    words = nltk.word_tokenize(data)
    for word in words:
        if word not in word2count.keys():
            word2count[word] = 1
        else:
            word2count[word] += 1
```

```
word2count
```

```
{'ram': 5,
 'raced': 3,
 'to': 4,
 'the': 3,
 'grocery': 2,
 'store': 2,
 'went': 1,
 'inside': 1,
 'but': 1,
 'realized': 1,
 'he': 1,
 'forgot': 1,
 'his': 1,
 'wallet': 1,
 'back': 2,
 'home': 1,
 'grab': 1,
 'it': 2,
 'once': 1,
 'found': 1,
 'car': 1,
 'again': 1,
 'and': 1,
 'drove': 1}
```

```
words
```

```
['once',
 'ram',
 'found',
 'it',
 'ram',
 'raced',
 'to',
 'the',
 'car',
 'again',
 'and',
 'drove',
 'back',
 'to',
 'the',
 'grocery',
 'store']
```

```
len(words)
```

```
17
```

```
vocab_size = len(wordset)
embed_dim = 10
context_size = 4
```

```
word_to_ix = {word: i for i, word in enumerate(wordset)}
ix_to_word = {i: word for i, word in enumerate(wordset)}
```

```
word_to_ix
```

```
{'grab': 0,
 'forgot': 1,
 'back': 2,
 'once': 3,
 'grocery': 4,
 'found': 5,
 'home': 6,
 'store': 7,
 'but': 8,
 'wallet': 9,
 'car': 10,
 'inside': 11,
 'the': 12,
 'ram': 13,
 'to': 14,
 'again': 15,
 'and': 16,
 'went': 17,
 'realized': 18,
 'it': 19,
 'raced': 20,
 'his': 21,
 'he': 22,
 'drove': 23}
```

```
ix_to_word
```

```
{0: 'grab',
 1: 'forgot',
 2: 'back',
 3: 'once',
 4: 'grocery',
 5: 'found',
 6: 'home',
 7: 'store',
 8: 'but',
 9: 'wallet',
 10: 'car',
 11: 'inside',
 12: 'the',
 13: 'ram',
 14: 'to',
 15: 'again',
 16: 'and',
 17: 'went',
 18: 'realized',
 19: 'it',
 20: 'raced',
 21: 'his',
 22: 'he',
 23: 'drove'}
```

```
data = []
for i in range(2, len(words) - 2):
    context = [words[i - 2], words[i - 1], words[i + 1], words[i + 2]]
    target = words[i]
    data.append((context, target))
print(data[:5])
```

```
[(['once', 'ram', 'it', 'ram'], 'found'), (['ram', 'found', 'ram', 'raced'], 'it'), (['found', 'it', 'raced', 'to'], 'ram'), (['it',
```

```
embeddings =  np.random.random_sample((vocab_size, embed_dim))
```

```
embeddings
```

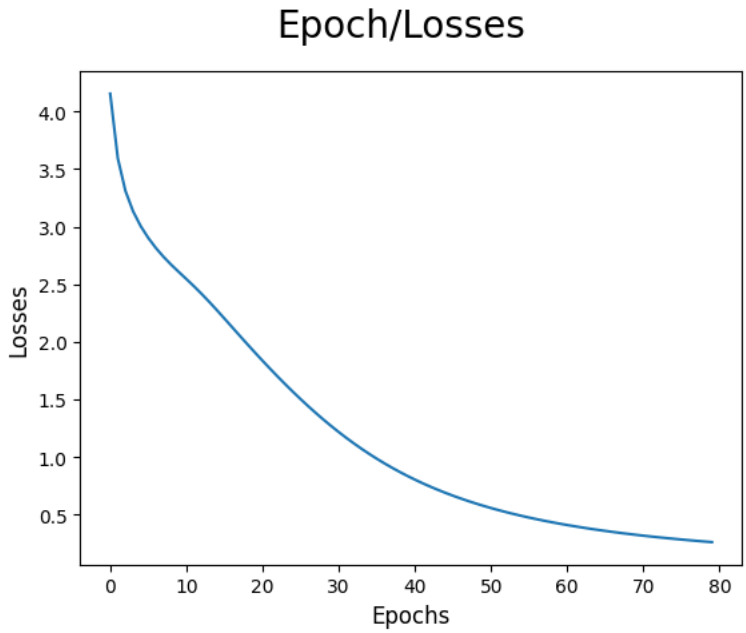
```
array([[0.44619563, 0.25702298, 0.7518889 , 0.73853142, 0.57449211,
        0.30262447, 0.92523433, 0.38430385, 0.4562259 , 0.75882258],
       [0.52011513, 0.7730353 , 0.36422371, 0.47108297, 0.05824687,
```

```
               0.43890918, 0.42171732, 0.44300873, 0.60486869, 0.93904089],
              [0.45100347, 0.53229724, 0.45374736, 0.09665696, 0.64853423,
               0.63109602, 0.71956318, 0.14929909, 0.33765372, 0.86861381],
              [0.59124721, 0.59263217, 0.73436609, 0.14570862, 0.34798821,
               0.31051236, 0.88409035, 0.98616619, 0.49003688, 0.46411356],
              [0.41437672, 0.89490502, 0.22041231, 0.45718606, 0.95413923,
               0.83846712, 0.36267988, 0.88901859, 0.5096098 , 0.34734505],
              [0.96534183, 0.45169581, 0.65292208, 0.18797838, 0.14906916,
               0.73109873, 0.20059192, 0.19416452, 0.82101352, 0.54894649],
              [0.96356839, 0.47852012, 0.93640607, 0.14485411, 0.75934024,
               0.37459702, 0.6565173 , 0.11547805, 0.44173435, 0.16985169],
              [0.23825436, 0.50297833, 0.45098152, 0.23713531, 0.81460485,
               0.34561123, 0.83485963, 0.7564537 , 0.4333513 , 0.64257242],
              [0.74031561, 0.43931661, 0.68054216, 0.47952197, 0.17599287,
               0.68978054, 0.64852779, 0.92518552, 0.11224424, 0.37659268],
              [0.46263426, 0.58980843, 0.4554626 , 0.18816074, 0.67859148,
               0.48600932, 0.82669461, 0.87522279, 0.36579631, 0.0376356 ],
              [0.31205298, 0.20633113, 0.94720568, 0.7695564 , 0.20527688,
               0.11229255, 0.17642284, 0.7373097 , 0.54998012, 0.01505024],
              [0.57028249, 0.73244989, 0.47772479, 0.69906242, 0.49655278,
               0.01509625, 0.86876073, 0.37406779, 0.60449393, 0.67610544],
              [0.42814752, 0.99692868, 0.15282719, 0.29760397, 0.05812538,
               0.69828524, 0.27105112, 0.77670222, 0.50473681, 0.69782225],
              [0.0020591 , 0.30014897, 0.58627841, 0.20217407, 0.22696359,
               0.00232041, 0.53828606, 0.89238554, 0.2325952 , 0.74327745],
              [0.02624089, 0.84492391, 0.16906366, 0.40076313, 0.12830342,
               0.35736856, 0.43311182, 0.68225284, 0.12135127, 0.16973905],
              [0.3862919 , 0.40839397, 0.98464128, 0.50631648, 0.01795265,
               0.47647633, 0.69289079, 0.11628582, 0.6398843 , 0.13748846],
              [0.64672849, 0.31351084, 0.47070746, 0.8031109 , 0.79682419,
               0.01449928, 0.1838197 , 0.6074353 , 0.58015147, 0.51500164],
              [0.31017752, 0.73430395, 0.01421099, 0.0300171 , 0.40350006,
               0.48564454, 0.97388568, 0.83534021, 0.83366876, 0.87521493],
              [0.61438115, 0.45468773, 0.6582586 , 0.27425476, 0.79394284,
               0.64227   , 0.97190338, 0.63587733, 0.13647462, 0.25905946],
              [0.83140933, 0.66308629, 0.95699422, 0.15377122, 0.65865988,
               0.06079501, 0.05908668, 0.14262918, 0.70838583, 0.27075892],
              [0.16375743, 0.41747094, 0.308228  , 0.52027049, 0.11526933,
               0.05313022, 0.10904407, 0.25745716, 0.74868685, 0.80799496],
              [0.75023775, 0.12556828, 0.45224744, 0.29429752, 0.99436853,
               0.11691266, 0.56728085, 0.01546555, 0.39780569, 0.45527147],
              [0.83067649, 0.26993769, 0.73606826, 0.03811081, 0.69662855,
               0.18700984, 0.77334426, 0.07335874, 0.84333739, 0.92604314],
              [0.91798576, 0.40091081, 0.4954483 , 0.28913135, 0.63459297,
               0.0391565 , 0.93199013, 0.28387055, 0.62856399, 0.81596679]])
```

```
def linear(m, theta):
    w = theta
    return m.dot(w)


def log_softmax(x):
    e_x = np.exp(x - np.max(x))
    return np.log(e_x / e_x.sum())
def NLLLoss(logs, targets):
    out = logs[range(len(targets)), targets]
    return -out.sum()/len(out)


import tensorflow as tf
import keras.backend as k
from keras.models import Sequential
from keras.layers import Dense, Embedding, Lambda


def log_softmax_crossentropy_with_logits(logits,target):

    out = np.zeros_like(logits)
    out[np.arange(len(logits)),target] = 1

    softmax = np.exp(logits) / np.exp(logits).sum(axis=-1,keepdims=True)

    return (- out + softmax) / logits.shape[0]


def forward(context_idxs, theta):
    m = embeddings[context_idxs].reshape(1, -1)
    n = linear(m, theta)
    o = log_softmax(n)

    return m, n, o


def backward(preds, theta, target_idxs):
    m, n, o = preds

    dlog = log_softmax_crossentropy_with_logits(n, target_idxs)
```

```
    dw = m.T.dot(dlog)

    return dw


def optimize(theta, grad, lr=0.03):
    theta -= grad * lr
    return theta


theta = np.random.uniform(-1, 1, ( context_size * embed_dim, vocab_size))


epoch_losses = {}

for epoch in range(80):

    losses =  []

    for context, target in data:
        context_idxs = np.array([word_to_ix[w] for w in context])
        preds = forward(context_idxs, theta)

        target_idxs = np.array([word_to_ix[target]])
        loss = NLLLoss(preds[-1], target_idxs)

        losses.append(loss)

        grad = backward(preds, theta, target_idxs)
        theta = optimize(theta, grad, lr=0.03)

        epoch_losses[epoch] = losses


import matplotlib.pyplot as plt
ix = np.arange(0,80)

fig = plt.figure()
fig.suptitle('Epoch/Losses', fontsize=20)
plt.plot(ix,[epoch_losses[i][0] for i in ix])
plt.xlabel('Epochs', fontsize=12)
plt.ylabel('Losses', fontsize=12)
```

```
Text(0, 0.5, 'Losses')
```




```
def predict(words):
    context_idxs = np.array([word_to_ix[w] for w in words])
    preds = forward(context_idxs, theta)
    word = ix_to_word[np.argmax(preds[-1])]
    return word
predict(['found', 'it', 'raced', 'to'])
```

```
'ram'
```

```
def accuracy():
    wrong = 0
```

```
    for context, target in data:
        if(predict(context) != target):
            wrong += 1

    return (1 - (wrong / len(data)))
accuracy()
```

```
1.0
```