Assignment -4
SMS SPAM CLASSIFICATION

Assignment Date	21 October 2022
Student Name	NARMADHA.R
Team ID	IBM-29269-1662616381
Maximum Marks	2 Marks

Import the Dataset

from google.colab import files
uploaded = files.upload()

Saving spam.csv to spam.csv

Import required libraries

import csv
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
STOPWORDS = set(stopwords.words('english'))

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.

[nltk_data]   Unzipping corpora/stopwords.zip.

Import dataset

import io
dataset = pd.read_csv(io.BytesIO(uploaded['spam.csv']), encoding ="ISO-8859-1")

dataset

```
       v1                                    v2 Unnamed: 2  \
0      ham  Go until jurong point, crazy.. Available only ...      NaN
1      ham                      Ok lar... Joking wif u oni...      NaN
2     spam  Free entry in 2 a wkly comp to win FA Cup fina...      NaN
3      ham  U dun say so early hor... U c already then say...      NaN
4      ham  Nah I don't think he goes to usf, he lives aro...      NaN
...    ...                                                ...      ...
5567  spam  This is the 2nd time we have tried 2 contact u...      NaN
5568   ham              Will Ì_ b going to esplanade fr home?      NaN
5569   ham  Pity, * was in mood for that. So...any other s...      NaN
5570   ham  The guy did some bitching but I acted like i'd...      NaN
5571   ham                      Rofl. Its true to its name      NaN

     Unnamed: 3 Unnamed: 4
0           NaN        NaN
1           NaN        NaN
2           NaN        NaN
3           NaN        NaN
4           NaN        NaN
...         ...        ...
5567        NaN        NaN
5568        NaN        NaN
5569        NaN        NaN
5570        NaN        NaN
5571        NaN        NaN

[5572 rows x 5 columns]

vocab_size =5000
embedding_dim =64
max_length =200
trunc_type ='post'
padding_type ='post'
oov_tok ="
training_portion =.8
```

Read the dataset and do pre-processing.

To remove the stop words.

```python
articles = []
labels = []

with open("spam.csv", 'r', encoding ="ISO-8859-1") as dataset:
    reader = csv.reader(dataset, delimiter=',')
    next(reader)
    for row in reader:
        labels.append(row[0])
        article = row[1]
        for word in STOPWORDS:
            token =' '+ word +' '
            article = article.replace(token, ' ')
            article = article.replace(' ', ' ')
        articles.append(article)
print(len(labels))
print(len(articles))
```

```
5572
5572
```

Train the model

```python
train_size = int(len(articles) * training_portion)
train_articles = articles[0: train_size]
train_labels = labels[0: train_size]
validation_articles = articles[train_size:]
validation_labels = labels[train_size:]
print(train_size)
print(len(train_articles))
print(len(train_labels))
print(len(validation_articles))
print(len(validation_labels))
```

```
4457
4457
4457
1115
1115
```

```python
tokenizer = Tokenizer(num_words = vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(train_articles)
word_index = tokenizer.word_index
dict(list(word_index.items())[0:10])
```

```
{'': 1,
 'i': 2,
 'u': 3,
 'call': 4,
 'you': 5,
 '2': 6,
 'get': 7,
 "i'm": 8,
 'ur': 9,
 'now': 10}
```

Training data to Sequences

```python
train_sequences = tokenizer.texts_to_sequences(train_articles)
print(train_sequences[10])
```

```
[8, 190, 37, 201, 30, 260, 293, 991, 222, 53, 153, 3815, 423, 46]
```

Train neural network for NLP

```python
train_padded = pad_sequences(train_sequences, maxlen=max_length, padding=padding_type, truncating=trunc_type)
print(len(train_sequences[0]))
print(len(train_padded[0]))

print(len(train_sequences[1]))
print(len(train_padded[1]))

print(len(train_sequences[10]))
print(len(train_padded[10]))
```

```
16
200
6
200
14
200
```

```
print(train_padded[10])
```

```
[   8  190   37  201   30  260  293  991  222   53  153 3815  423   46
    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    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    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    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    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    0    0
    0    0    0    0]
```

```
validation_sequences = tokenizer.texts_to_sequences(validation_articles)
validation_padded = pad_sequences(validation_sequences, maxlen=max_length, padding=padding_type,
truncating=trunc_type)

print(len(validation_sequences))
print(validation_padded.shape)
```

```
1115
(1115, 200)
```

```
label_tokenizer = Tokenizer()
label_tokenizer.fit_on_texts(labels)

training_label_seq = np.array(label_tokenizer.texts_to_sequences(train_labels))
validation_label_seq = np.array(label_tokenizer.texts_to_sequences(validation_labels))
print(training_label_seq[0])
print(training_label_seq[1])
print(training_label_seq[2])
print(training_label_seq.shape)

print(validation_label_seq[0])
print(validation_label_seq[1])
```

```python
print(validation_label_seq[2])
print(validation_label_seq.shape)
```

[1]
[1]
[2]
(4457, 1)
[1]
[2]
[1]
(1115, 1)

```python
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])


def decode_article(text):
    return ' '.join([reverse_word_index.get(i, '?') for i in text])
print(decode_article(train_padded[10]))
print('---')
print(train_articles[10])
```

i'm gonna home soon want talk stuff anymore tonight k i've cried enough today ?
? ?
? ?
? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?

I'm gonna home soon want talk stuff anymore tonight, k? I've cried enough today.

To implement LSTM

```python
model = tf.keras.Sequential([

    tf.keras.layers.Embedding(vocab_size, embedding_dim),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(embedding_dim)),
    tf.keras.layers.Dense(embedding_dim, activation='relu'),
    tf.keras.layers.Dense(6, activation='softmax')
])
model.summary()
```

Model: "sequential"

_____

Layer (type)	Output Shape	Param #

```
=================================================================
embedding (Embedding)      (None, None, 64)        320000

bidirectional (Bidirectional  (None, 128)            66048
l)

dense (Dense)              (None, 64)            8256

dense_1 (Dense)            (None, 6)             390

=================================================================
Total params: 394,694
Trainable params: 394,694
Non-trainable params: 0
```

_____

print(set(labels))

{'spam', 'ham'}

model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
num_epochs =10
history = model.fit(train_padded, training_label_seq, epochs=num_epochs, validation_data=(validation_padded, validation_label_seq), verbose=2)

Epoch 1/10
140/140 - 37s - loss: 0.3177 - accuracy: 0.9251 - val_loss: 0.0387 - val_accuracy: 0.9830 - 37s/epoch - 265ms/step
Epoch 2/10
140/140 - 35s - loss: 0.0310 - accuracy: 0.9915 - val_loss: 0.0318 - val_accuracy: 0.9901 - 35s/epoch - 252ms/step
Epoch 3/10
140/140 - 32s - loss: 0.0130 - accuracy: 0.9975 - val_loss: 0.0627 - val_accuracy: 0.9857 - 32s/epoch - 230ms/step
Epoch 4/10
140/140 - 31s - loss: 0.0060 - accuracy: 0.9987 - val_loss: 0.0478 - val_accuracy: 0.9901 - 31s/epoch - 220ms/step
Epoch 5/10
140/140 - 30s - loss: 0.0042 - accuracy: 0.9989 - val_loss: 0.0613 - val_accuracy: 0.9883 - 30s/epoch - 215ms/step
Epoch 6/10
140/140 - 29s - loss: 0.0033 - accuracy: 0.9991 - val_loss: 0.0728 - val_accuracy: 0.9883 - 29s/epoch - 210ms/step
Epoch 7/10
140/140 - 29s - loss: 0.0020 - accuracy: 0.9996 - val_loss: 0.0540 - val_accuracy: 0.9865 - 29s/epoch - 208ms/step
Epoch 8/10

140/140 - 31s - loss: 7.6466e-04 - accuracy: 0.9998 - val_loss: 0.0644 - val_accuracy: 0.9901 - 31s/epoch - 219ms/step
Epoch 9/10
140/140 - 30s - loss: 3.9159e-04 - accuracy: 1.0000 - val_loss: 0.0678 - val_accuracy: 0.9883 - 30s/epoch - 211ms/step
Epoch 10/10
140/140 - 29s - loss: 1.7514e-04 - accuracy: 1.0000 - val_loss: 0.0726 - val_accuracy: 0.9883 - 29s/epoch - 208ms/step

```python
def plot_graphs(history, string):
  plt.plot(history.history[string])
  plt.plot(history.history['val_'+string])
  plt.xlabel("Epochs")
  plt.ylabel(string)
  plt.legend([string, 'val_'+string])
  plt.show()

plot_graphs(history, "accuracy")
plot_graphs(history, "loss")
```