Assignment -4
SMS SPAM Classification

Assignment Date	31 October 2022
Student Name	Aravind.S
Student Roll Number	610519104005
Maximum Marks	2 Marks

Import the dataset

Input:

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

output :

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
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]   Package stopwords is already up-to-date!
```

import dataset

```
import io
dataset = pd.read_csv(io.BytesIO(uploaded['spam.csv']))
```

Input:

dataset

output :

v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9	...	Unnamed: 28	Unnamed: 29	Unnamed: 30	Unnamed: 31	Unnamed: 32
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
5568	ham	Will �_ b going to esplanade fr home?	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
5569	ham	Pity, * was in mood for that. So...any other s...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
5570	ham	The guy did some bitching but I acted like i'd...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
5571	ham	Rofl. Its true to its name	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN

5572 rows × 38 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.

Input:

```
articles = []
labels = []

with open("spam.csv", 'r') 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))
```

output :

5572
5572

Train the model.

Input:

```
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))
```

output :

4457
4457
4457
1115
1115

Input:

```
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])
```

output :

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

Traning data to Sequences.

Input:

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

Output:

[8, 187, 38, 200, 29, 259, 290, 1080, 225, 53, 153, 3760, 458, 45]

Train neural network for NLP.

Input:

```
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]))
```

Output:

16
200
6
200
14
200

Input:

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

Output:

```
[   8  187   38  200   29  259  290 1080  225   53  153 3760  458   45
    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]
```

Input:

```
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)
```

Output:

1115
(1115, 200)

Input:

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

Output:

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

Input:

```
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])
```

Output:

**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.

Input:

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

    # Add an Embedding layer expecting input vocab of size 5000, and output embedding dimension of
size 64 we set at the top

    tf.keras.layers.Embedding(vocab_size, embedding_dim),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(embedding_dim)),

    #tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    # use ReLU in place of tanh function since they are very good alternatives of each other.

    tf.keras.layers.Dense(embedding_dim, activation='relu'),

    # Add a Dense layer with 6 units and softmax activation.
    # When we have multiple outputs, softmax convert outputs layers into a probability distribution.

    tf.keras.layers.Dense(6, activation='softmax')
])
model.summary()
```

Output

Model: "sequential"

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, None, 64)          320000

 bidirectional (Bidirectiona  (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
_________________________________________________________________
```

Input:

```
print(set(labels))
```

Output:

{'ham', 'spam'}

Input:

```
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)
```

Output:

Epoch 1/10
140/140 - 35s - loss: 0.3508 - accuracy: 0.9114 - val_loss: 0.0527 - val_accuracy: 0.9812 - 35s/epoch - 247ms/step
Epoch 2/10
140/140 - 29s - loss: 0.0354 - accuracy: 0.9904 - val_loss: 0.0352 - val_accuracy: 0.9874 - 29s/epoch - 205ms/step
Epoch 3/10
140/140 - 29s - loss: 0.0168 - accuracy: 0.9962 - val_loss: 0.0334 - val_accuracy: 0.9901 - 29s/epoch - 205ms/step
Epoch 4/10
140/140 - 29s - loss: 0.0066 - accuracy: 0.9984 - val_loss: 0.0477 - val_accuracy: 0.9892 - 29s/epoch - 205ms/step

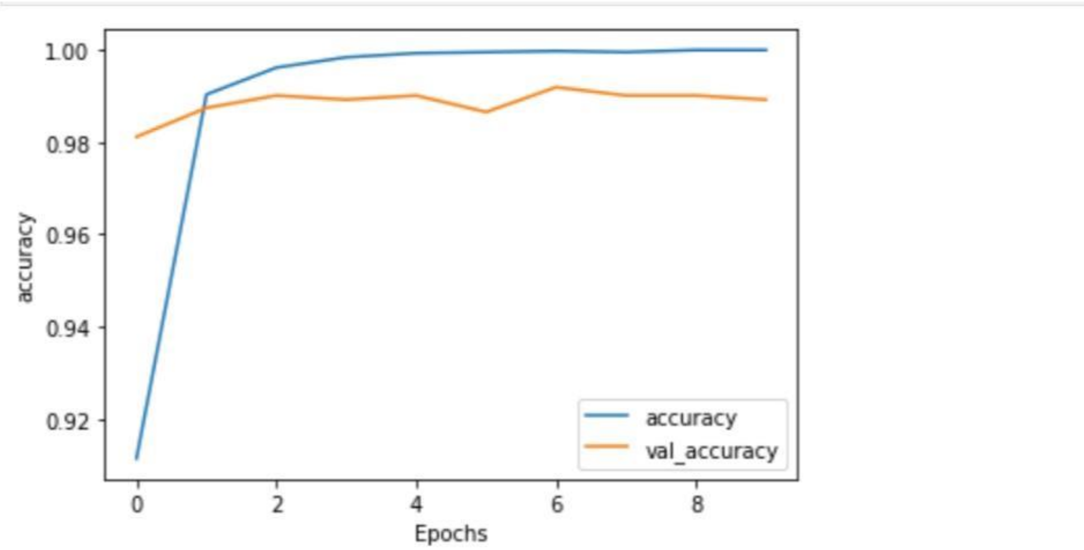
Epoch 5/10
140/140 - 30s - loss: 0.0042 - accuracy: 0.9993 - val_loss: 0.0415 - val_accuracy: 0.9901 - 30s/epoch - 214ms/step
Epoch 6/10
140/140 - 30s - loss: 0.0026 - accuracy: 0.9996 - val_loss: 0.0650 - val_accuracy: 0.9865 - 30s/epoch - 215ms/step
Epoch 7/10
140/140 - 29s - loss: 0.0015 - accuracy: 0.9998 - val_loss: 0.0573 - val_accuracy: 0.9919 - 29s/epoch - 204ms/step
Epoch 8/10
140/140 - 29s - loss: 9.5079e-04 - accuracy: 0.9996 - val_loss: 0.0646 - val_accuracy: 0.9901 - 29s/epoch - 205ms/step
Epoch 9/10
140/140 - 29s - loss: 3.1964e-04 - accuracy: 1.0000 - val_loss: 0.0618 - val_accuracy: 0.9901 - 29s/epoch - 207ms/step
Epoch 10/10
140/140 - 29s - loss: 1.9858e-04 - accuracy: 1.0000 - val_loss: 0.0654 - val_accuracy: 0.9892 - 29s/epoch - 205ms/step

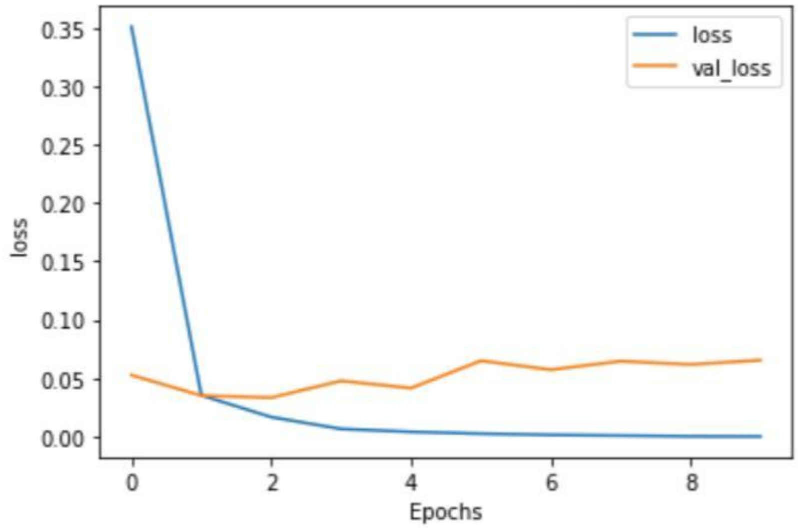
Input:

```
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")
```

Output:




loss
val_loss
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
loss
0
2
4
6
8
Epochs