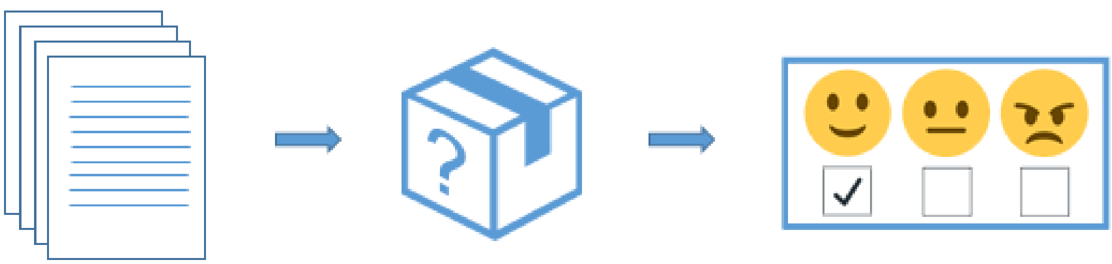
5.2D - DeepFakeComments Generator

August 10, 2023

Welcome to your assignment this week!

To better understand the adverse use of AI, in this assignment, we will look at a Natural Language Processing use case.

Natural Language Pocessing (NLP) is a branch of Artificial Intelligence (AI) that helps computers to understand, to interpret and to manipulate natural (i.e. human) language. Imagine NLP-powered machines as black boxes that are capable of understanding and evaluating the context of the input documents (i.e. collection of words), outputting meaningful results that depend on the task the machine is designed for.



Documents are fed into magic NLP model capable to get, for instance, the sentiment of the original content

In this notebook, you will implement a model that uses an LSTM to generate fake tweets and comments. You will also be able to try it to generate your own fake text.

You will learn to: - Apply an LSTM to generate fake comments. - Generate your own fake text with deep learning.

Run the following cell to load the packages you will need.

[1]:
```
import time
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout,␣
↪Bidirectional
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import regularizers
import tensorflow.keras.utils as ku
import keras.backend as K
```

```
import matplotlib.pyplot as plt
import numpy as np
```

1 Build the model

Let's define a tokenizer and read the data from disk.

[2]:
```
tokenizer = Tokenizer(filters='"#$%&()*+-/:;<=>@[\\]^_`{|}~\t\n')
data = open('covid19_fake.txt').read().replace(".", " . ").replace(",", " , ").
↪replace("?", " ? ").replace("!", " ! ")
```

Now, let's splits the data into tweets where each line of the input file is a fake tweets.

We also extract the vocabulary of the data.

[3]:
```
corpus = data.lower().split("\n")
tokenizer.fit_on_texts(corpus)
total_words = len(tokenizer.word_index) + 1
```

You've loaded: - `corpus`: an array where each entry is a fake post. - `tokenizer`: which is the object that we will use to vectorize our dataset. This object also contains our word index. - `total_words`: is the total number of words in the vacabulary.

[4]:
```
print("Example of fake tweets: ",corpus[:2])
print("Size of the vocabulary = ", total_words)
index = [(k, v) for k, v in tokenizer.word_index.items()]
print("Example of our word index = ", index[0:10])
```

```
Example of fake tweets:  ['there is already a vaccine to treat covid19 . ',
'cleaning hands do not help to prevent covid19 . ']
Size of the vocabulary =  1257
Example of our word index =  [('.', 1), ('the', 2), ('covid19', 3), ('in', 4),
('to', 5), ('a', 6), ('of', 7), (',', 8), ('coronavirus', 9), ('and', 10)]
```

The next step aims to generate the training set of n_grams sequences.

[5]:
```
input_sequences = []
for line in corpus:
    token_list = tokenizer.texts_to_sequences([line])[0]
    for i in range(1, len(token_list)):
        n_gram_sequence = token_list[:i+1]
        input_sequences.append(n_gram_sequence)
```

You've create: - `input_sequences`: which is a list of n_grams sequences.

[6]:
```
sample = 20
reverse_word_map = dict(map(reversed, tokenizer.word_index.items()))
print("The entry ",sample," in 'input_sequences' is: ")
print(input_sequences[sample])
print(" and it corresponds to:")
```

```
for i in input_sequences[sample]:
    print(reverse_word_map[i], end=' ')
```

```
The entry  20  in 'input_sequences' is:
[2, 3, 12, 187, 34, 188]
 and it corresponds to:
the covid19 is same as sars
```

Next, we padd our training set to the max length in order to be able to make a batch processing.

[7]:
```
max_sequence_len = max([len(x) for x in input_sequences])
input_sequences = np.array(pad_sequences(input_sequences,
 maxlen=max_sequence_len, padding='pre'))
```

Run the following to see the containt of the padded 'input_sequences' object.

[8]:
```
reverse_word_map = dict(map(reversed, tokenizer.word_index.items()))
print("The entry ",sample," in 'input_sequences' is: ")
print(input_sequences[sample])
print(" and it corresponds to:")
print("[", end=' ')
for i in input_sequences[sample]:
    if i in reverse_word_map:
        print(reverse_word_map[i], end=' ')
    else:
        print("__", end=' ')
print("]")
```

```
The entry  20  in 'input_sequences' is:
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   2   3  12 187  34 188]
 and it corresponds to:
[ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __
__ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __
__ __ the covid19 is same as sars ]
```

Given a sentence like **"the covid19 is same as"**, we want to design a model that can predict the next word – in the case the word **"sars"**.

Therefore, the next code prepares our input and output to our model consequently.

[9]:
```
input_to_model, label = input_sequences[:,:-1],input_sequences[:,-1]
```

[10]:
```
print("The entry ",sample," in 'input_sequences' is: ")
print(input_sequences[sample])
print(", it corresponds to the following input to our model:")
```

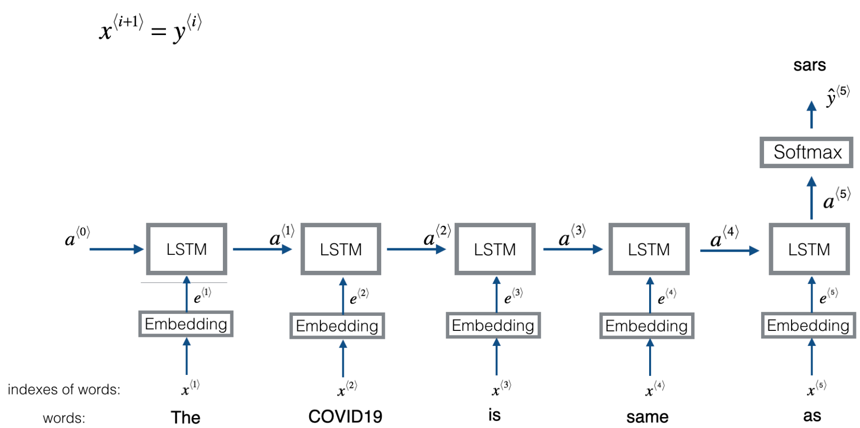
```python
print(input_to_model[sample])
print(" and the following output: ", label[sample])
```

```
The entry  20  in 'input_sequences' is:
[  0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
   0    2    3   12  187   34  188]
, it corresponds to the following input to our model:
[  0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
   0    2    3   12  187   34]
 and the following output:  188
```

Finally, we convert our label to categorical labels for being processed by our model.

[11]:
```python
label = ku.to_categorical(label, num_classes=total_words)
```

Here is the architecture of the model we will use:



Task 1: Implement `deep_fake_comment_model()`. You will need to carry out 5 steps:

1. Create a sequencial model using the `Sequential` class
2. Add an embedding layer to the model using the `Embedding` class of size 128
3. Add an LSTM layer to the model using the `LSTM` class of size 128
4. Add a Dense layer to the model using the `Dense` class with a `softmax` activation
5. Set a `categorical_crossentropy` loss function to the model and optimize `accuracy`.

[12]:
```python
#TASK 1
# deep_fake_comment_model

def deep_fake_comment_model():
    model = Sequential()
```

```
    model.add(Embedding(input_dim=total_words, output_dim=128,␣
 ↪input_length=max_sequence_len - 1))
    model.add(LSTM(128))
    model.add(Dense(total_words, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])

    return model

#Print details of the model.
model = deep_fake_comment_model()
```

[13]:
```
model.summary()
```

```
Model: "sequential"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 60, 128)           160896

 lstm (LSTM)                 (None, 128)               131584

 dense (Dense)               (None, 1257)              162153

=================================================================
Total params: 454633 (1.73 MB)
Trainable params: 454633 (1.73 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Now, let's start our training.

[14]:
```
history = model.fit(input_to_model, label, epochs=200, batch_size=32, verbose=1)
```

```
Epoch 1/200
126/126 [==============================] - 7s 39ms/step - loss: 6.3848 -
accuracy: 0.0667
Epoch 2/200
126/126 [==============================] - 5s 41ms/step - loss: 5.8770 -
accuracy: 0.0717
Epoch 3/200
126/126 [==============================] - 5s 39ms/step - loss: 5.7410 -
accuracy: 0.0854
Epoch 4/200
126/126 [==============================] - 5s 38ms/step - loss: 5.6039 -
accuracy: 0.1109
Epoch 5/200
126/126 [==============================] - 5s 36ms/step - loss: 5.4357 -
accuracy: 0.1251
```

```
Epoch 6/200
126/126 [==============================] - 4s 35ms/step - loss: 5.2372 -
accuracy: 0.1417
Epoch 7/200
126/126 [==============================] - 4s 35ms/step - loss: 5.0509 -
accuracy: 0.1558
Epoch 8/200
126/126 [==============================] - 4s 35ms/step - loss: 4.8756 -
accuracy: 0.1680
Epoch 9/200
126/126 [==============================] - 4s 36ms/step - loss: 4.7062 -
accuracy: 0.1784
Epoch 10/200
126/126 [==============================] - 4s 36ms/step - loss: 4.5490 -
accuracy: 0.1866
Epoch 11/200
126/126 [==============================] - 4s 35ms/step - loss: 4.3935 -
accuracy: 0.2005
Epoch 12/200
126/126 [==============================] - 4s 35ms/step - loss: 4.2370 -
accuracy: 0.2154
Epoch 13/200
126/126 [==============================] - 5s 36ms/step - loss: 4.0808 -
accuracy: 0.2303
Epoch 14/200
126/126 [==============================] - 4s 35ms/step - loss: 3.9296 -
accuracy: 0.2444
Epoch 15/200
126/126 [==============================] - 4s 35ms/step - loss: 3.7727 -
accuracy: 0.2682
Epoch 16/200
126/126 [==============================] - 4s 35ms/step - loss: 3.6193 -
accuracy: 0.2881
Epoch 17/200
126/126 [==============================] - 4s 35ms/step - loss: 3.4687 -
accuracy: 0.3072
Epoch 18/200
126/126 [==============================] - 4s 35ms/step - loss: 3.3193 -
accuracy: 0.3251
Epoch 19/200
126/126 [==============================] - 4s 35ms/step - loss: 3.1680 -
accuracy: 0.3496
Epoch 20/200
126/126 [==============================] - 4s 35ms/step - loss: 3.0207 -
accuracy: 0.3725
Epoch 21/200
126/126 [==============================] - 4s 36ms/step - loss: 2.8724 -
accuracy: 0.3923
```

```
Epoch 22/200
126/126 [==============================] - 4s 35ms/step - loss: 2.7249 -
accuracy: 0.4201
Epoch 23/200
126/126 [==============================] - 4s 35ms/step - loss: 2.5843 -
accuracy: 0.4538
Epoch 24/200
126/126 [==============================] - 4s 35ms/step - loss: 2.4458 -
accuracy: 0.4814
Epoch 25/200
126/126 [==============================] - 4s 35ms/step - loss: 2.3067 -
accuracy: 0.5154
Epoch 26/200
126/126 [==============================] - 4s 35ms/step - loss: 2.1753 -
accuracy: 0.5506
Epoch 27/200
126/126 [==============================] - 4s 35ms/step - loss: 2.0457 -
accuracy: 0.5792
Epoch 28/200
126/126 [==============================] - 4s 35ms/step - loss: 1.9243 -
accuracy: 0.6159
Epoch 29/200
126/126 [==============================] - 4s 35ms/step - loss: 1.8047 -
accuracy: 0.6486
Epoch 30/200
126/126 [==============================] - 4s 35ms/step - loss: 1.6927 -
accuracy: 0.6759
Epoch 31/200
126/126 [==============================] - 4s 35ms/step - loss: 1.5847 -
accuracy: 0.7062
Epoch 32/200
126/126 [==============================] - 4s 35ms/step - loss: 1.4817 -
accuracy: 0.7313
Epoch 33/200
126/126 [==============================] - 4s 35ms/step - loss: 1.3885 -
accuracy: 0.7536
Epoch 34/200
126/126 [==============================] - 4s 35ms/step - loss: 1.2975 -
accuracy: 0.7737
Epoch 35/200
126/126 [==============================] - 4s 35ms/step - loss: 1.2139 -
accuracy: 0.7906
Epoch 36/200
126/126 [==============================] - 4s 35ms/step - loss: 1.1370 -
accuracy: 0.8087
Epoch 37/200
126/126 [==============================] - 4s 36ms/step - loss: 1.0651 -
accuracy: 0.8114
```

```
Epoch 38/200
126/126 [==============================] - 4s 35ms/step - loss: 0.9966 -
accuracy: 0.8357
Epoch 39/200
126/126 [==============================] - 4s 35ms/step - loss: 0.9352 -
accuracy: 0.8422
Epoch 40/200
126/126 [==============================] - 4s 35ms/step - loss: 0.8777 -
accuracy: 0.8521
Epoch 41/200
126/126 [==============================] - 4s 35ms/step - loss: 0.8228 -
accuracy: 0.8623
Epoch 42/200
126/126 [==============================] - 4s 35ms/step - loss: 0.7729 -
accuracy: 0.8722
Epoch 43/200
126/126 [==============================] - 4s 35ms/step - loss: 0.7294 -
accuracy: 0.8804
Epoch 44/200
126/126 [==============================] - 4s 35ms/step - loss: 0.6844 -
accuracy: 0.8881
Epoch 45/200
126/126 [==============================] - 4s 36ms/step - loss: 0.6456 -
accuracy: 0.8938
Epoch 46/200
126/126 [==============================] - 4s 35ms/step - loss: 0.6085 -
accuracy: 0.8983
Epoch 47/200
126/126 [==============================] - 4s 35ms/step - loss: 0.5751 -
accuracy: 0.9079
Epoch 48/200
126/126 [==============================] - 4s 35ms/step - loss: 0.5424 -
accuracy: 0.9094
Epoch 49/200
126/126 [==============================] - 4s 35ms/step - loss: 0.5144 -
accuracy: 0.9129
Epoch 50/200
126/126 [==============================] - 4s 35ms/step - loss: 0.4882 -
accuracy: 0.9179
Epoch 51/200
126/126 [==============================] - 4s 35ms/step - loss: 0.4605 -
accuracy: 0.9233
Epoch 52/200
126/126 [==============================] - 4s 35ms/step - loss: 0.4398 -
accuracy: 0.9243
Epoch 53/200
126/126 [==============================] - 4s 35ms/step - loss: 0.4168 -
accuracy: 0.9278
```

```
Epoch 54/200
126/126 [==============================] - 4s 36ms/step - loss: 0.3981 -
accuracy: 0.9318
Epoch 55/200
126/126 [==============================] - 4s 35ms/step - loss: 0.3765 -
accuracy: 0.9337
Epoch 56/200
126/126 [==============================] - 4s 35ms/step - loss: 0.3608 -
accuracy: 0.9362
Epoch 57/200
126/126 [==============================] - 4s 35ms/step - loss: 0.3432 -
accuracy: 0.9385
Epoch 58/200
126/126 [==============================] - 4s 35ms/step - loss: 0.3290 -
accuracy: 0.9395
Epoch 59/200
126/126 [==============================] - 4s 35ms/step - loss: 0.3134 -
accuracy: 0.9385
Epoch 60/200
126/126 [==============================] - 4s 35ms/step - loss: 0.3003 -
accuracy: 0.9414
Epoch 61/200
126/126 [==============================] - 4s 35ms/step - loss: 0.2905 -
accuracy: 0.9400
Epoch 62/200
126/126 [==============================] - 4s 35ms/step - loss: 0.2782 -
accuracy: 0.9414
Epoch 63/200
126/126 [==============================] - 4s 35ms/step - loss: 0.2671 -
accuracy: 0.9457
Epoch 64/200
126/126 [==============================] - 5s 36ms/step - loss: 0.2578 -
accuracy: 0.9447
Epoch 65/200
126/126 [==============================] - 4s 36ms/step - loss: 0.2499 -
accuracy: 0.9452
Epoch 66/200
126/126 [==============================] - 4s 35ms/step - loss: 0.2420 -
accuracy: 0.9464
Epoch 67/200
126/126 [==============================] - 4s 36ms/step - loss: 0.2357 -
accuracy: 0.9471
Epoch 68/200
126/126 [==============================] - 4s 35ms/step - loss: 0.2280 -
accuracy: 0.9484
Epoch 69/200
126/126 [==============================] - 4s 35ms/step - loss: 0.2200 -
accuracy: 0.9449
```

```
Epoch 70/200
126/126 [==============================] - 4s 35ms/step - loss: 0.2137 -
accuracy: 0.9491
Epoch 71/200
126/126 [==============================] - 4s 35ms/step - loss: 0.2085 -
accuracy: 0.9471
Epoch 72/200
126/126 [==============================] - 4s 35ms/step - loss: 0.2039 -
accuracy: 0.9459
Epoch 73/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1990 -
accuracy: 0.9467
Epoch 74/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1938 -
accuracy: 0.9471
Epoch 75/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1905 -
accuracy: 0.9504
Epoch 76/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1862 -
accuracy: 0.9494
Epoch 77/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1825 -
accuracy: 0.9501
Epoch 78/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1790 -
accuracy: 0.9489
Epoch 79/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1755 -
accuracy: 0.9484
Epoch 80/200
126/126 [==============================] - 5s 36ms/step - loss: 0.1717 -
accuracy: 0.9491
Epoch 81/200
126/126 [==============================] - 4s 36ms/step - loss: 0.1708 -
accuracy: 0.9484
Epoch 82/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1676 -
accuracy: 0.9489
Epoch 83/200
126/126 [==============================] - 5s 38ms/step - loss: 0.1659 -
accuracy: 0.9506
Epoch 84/200
126/126 [==============================] - 5s 36ms/step - loss: 0.1645 -
accuracy: 0.9491
Epoch 85/200
126/126 [==============================] - 5s 36ms/step - loss: 0.1614 -
accuracy: 0.9491
```

```
Epoch 86/200
126/126 [==============================] - 5s 36ms/step - loss: 0.1600 -
accuracy: 0.9499
Epoch 87/200
126/126 [==============================] - 4s 36ms/step - loss: 0.1572 -
accuracy: 0.9494
Epoch 88/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1564 -
accuracy: 0.9486
Epoch 89/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1546 -
accuracy: 0.9491
Epoch 90/200
126/126 [==============================] - 4s 36ms/step - loss: 0.1522 -
accuracy: 0.9479
Epoch 91/200
126/126 [==============================] - 4s 36ms/step - loss: 0.1545 -
accuracy: 0.9499
Epoch 92/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1592 -
accuracy: 0.9489
Epoch 93/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1547 -
accuracy: 0.9474
Epoch 94/200
126/126 [==============================] - 4s 36ms/step - loss: 0.1507 -
accuracy: 0.9479
Epoch 95/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1474 -
accuracy: 0.9504
Epoch 96/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1454 -
accuracy: 0.9484
Epoch 97/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1448 -
accuracy: 0.9479
Epoch 98/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1435 -
accuracy: 0.9504
Epoch 99/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1430 -
accuracy: 0.9484
Epoch 100/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1420 -
accuracy: 0.9499
Epoch 101/200
126/126 [==============================] - 4s 36ms/step - loss: 0.1405 -
accuracy: 0.9489
```

```
Epoch 102/200
126/126 [==============================] - 5s 37ms/step - loss: 0.1405 -
accuracy: 0.9496
Epoch 103/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1397 -
accuracy: 0.9494
Epoch 104/200
126/126 [==============================] - 5s 36ms/step - loss: 0.1383 -
accuracy: 0.9501
Epoch 105/200
126/126 [==============================] - 5s 36ms/step - loss: 0.1379 -
accuracy: 0.9501
Epoch 106/200
126/126 [==============================] - 4s 36ms/step - loss: 0.1376 -
accuracy: 0.9486
Epoch 107/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1369 -
accuracy: 0.9501
Epoch 108/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1362 -
accuracy: 0.9509
Epoch 109/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1368 -
accuracy: 0.9491
Epoch 110/200
126/126 [==============================] - 4s 36ms/step - loss: 0.1363 -
accuracy: 0.9506
Epoch 111/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1359 -
accuracy: 0.9491
Epoch 112/200
126/126 [==============================] - 5s 36ms/step - loss: 0.1347 -
accuracy: 0.9489
Epoch 113/200
126/126 [==============================] - 5s 36ms/step - loss: 0.1343 -
accuracy: 0.9499
Epoch 114/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1341 -
accuracy: 0.9491
Epoch 115/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1336 -
accuracy: 0.9506
Epoch 116/200
126/126 [==============================] - 5s 36ms/step - loss: 0.1378 -
accuracy: 0.9489
Epoch 117/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1713 -
accuracy: 0.9382
```

```
Epoch 118/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1569 -
accuracy: 0.9462
Epoch 119/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1420 -
accuracy: 0.9481
Epoch 120/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1345 -
accuracy: 0.9526
Epoch 121/200
126/126 [==============================] - 5s 36ms/step - loss: 0.1335 -
accuracy: 0.9486
Epoch 122/200
126/126 [==============================] - 4s 36ms/step - loss: 0.1328 -
accuracy: 0.9499
Epoch 123/200
126/126 [==============================] - 5s 36ms/step - loss: 0.1320 -
accuracy: 0.9501
Epoch 124/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1309 -
accuracy: 0.9501
Epoch 125/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1312 -
accuracy: 0.9496
Epoch 126/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1307 -
accuracy: 0.9484
Epoch 127/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1302 -
accuracy: 0.9499
Epoch 128/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1306 -
accuracy: 0.9519
Epoch 129/200
126/126 [==============================] - 4s 36ms/step - loss: 0.1315 -
accuracy: 0.9494
Epoch 130/200
126/126 [==============================] - 4s 36ms/step - loss: 0.1307 -
accuracy: 0.9491
Epoch 131/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1300 -
accuracy: 0.9506
Epoch 132/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1298 -
accuracy: 0.9511
Epoch 133/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1294 -
accuracy: 0.9504
```

```
Epoch 134/200
126/126 [==============================] - 4s 36ms/step - loss: 0.1294 -
accuracy: 0.9496
Epoch 135/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1295 -
accuracy: 0.9509
Epoch 136/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1289 -
accuracy: 0.9501
Epoch 137/200
126/126 [==============================] - 4s 36ms/step - loss: 0.1287 -
accuracy: 0.9496
Epoch 138/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1289 -
accuracy: 0.9506
Epoch 139/200
126/126 [==============================] - 4s 36ms/step - loss: 0.1287 -
accuracy: 0.9494
Epoch 140/200
126/126 [==============================] - 5s 36ms/step - loss: 0.1281 -
accuracy: 0.9494
Epoch 141/200
126/126 [==============================] - 5s 36ms/step - loss: 0.1283 -
accuracy: 0.9489
Epoch 142/200
126/126 [==============================] - 4s 36ms/step - loss: 0.1281 -
accuracy: 0.9509
Epoch 143/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1293 -
accuracy: 0.9496
Epoch 144/200
126/126 [==============================] - 4s 36ms/step - loss: 0.1500 -
accuracy: 0.9432
Epoch 145/200
126/126 [==============================] - 4s 36ms/step - loss: 0.1565 -
accuracy: 0.9429
Epoch 146/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1406 -
accuracy: 0.9467
Epoch 147/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1333 -
accuracy: 0.9474
Epoch 148/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1293 -
accuracy: 0.9501
Epoch 149/200
126/126 [==============================] - 5s 36ms/step - loss: 0.1290 -
accuracy: 0.9484
```

```
Epoch 150/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1288 -
accuracy: 0.9489
Epoch 151/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1281 -
accuracy: 0.9504
Epoch 152/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1278 -
accuracy: 0.9501
Epoch 153/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1276 -
accuracy: 0.9511
Epoch 154/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1282 -
accuracy: 0.9504
Epoch 155/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1277 -
accuracy: 0.9501
Epoch 156/200
126/126 [==============================] - 4s 36ms/step - loss: 0.1273 -
accuracy: 0.9504
Epoch 157/200
126/126 [==============================] - 5s 37ms/step - loss: 0.1275 -
accuracy: 0.9504
Epoch 158/200
126/126 [==============================] - 5s 43ms/step - loss: 0.1280 -
accuracy: 0.9509
Epoch 159/200
126/126 [==============================] - 5s 41ms/step - loss: 0.1272 -
accuracy: 0.9484
Epoch 160/200
126/126 [==============================] - 5s 38ms/step - loss: 0.1274 -
accuracy: 0.9486
Epoch 161/200
126/126 [==============================] - 5s 43ms/step - loss: 0.1273 -
accuracy: 0.9486
Epoch 162/200
126/126 [==============================] - 5s 40ms/step - loss: 0.1272 -
accuracy: 0.9481
Epoch 163/200
126/126 [==============================] - 5s 42ms/step - loss: 0.1272 -
accuracy: 0.9496
Epoch 164/200
126/126 [==============================] - 5s 39ms/step - loss: 0.1275 -
accuracy: 0.9501
Epoch 165/200
126/126 [==============================] - 5s 38ms/step - loss: 0.1273 -
accuracy: 0.9514
```

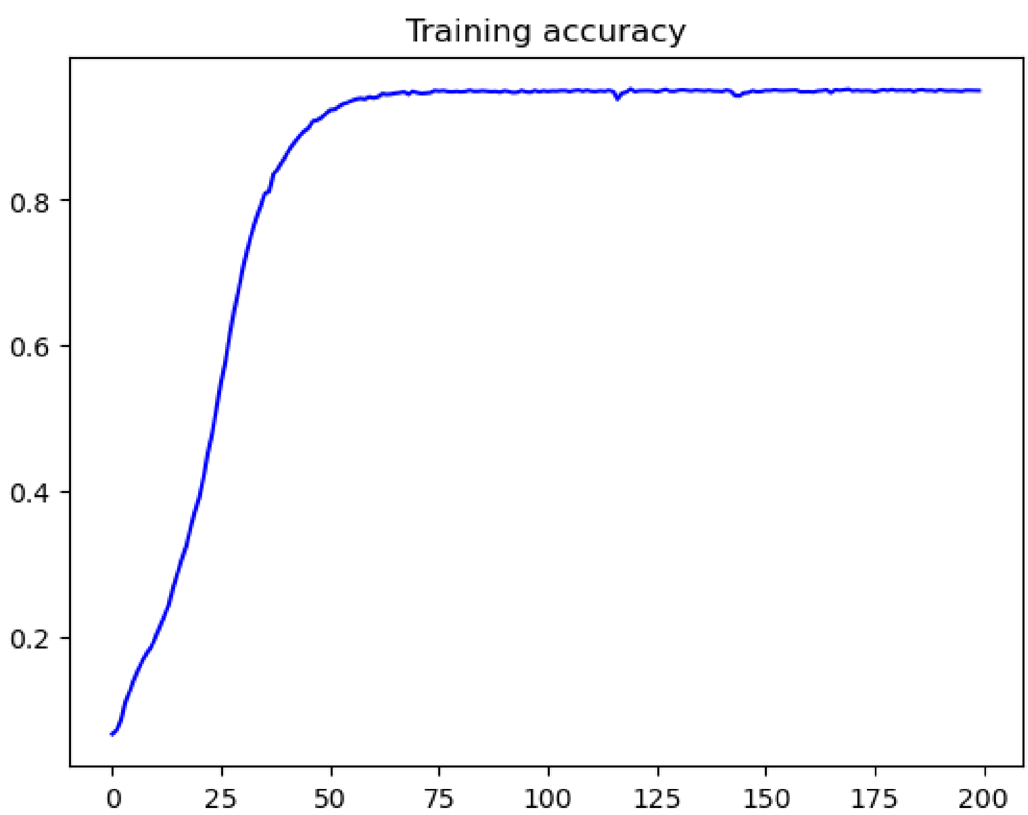
```
Epoch 166/200
126/126 [==============================] - 5s 36ms/step - loss: 0.1275 -
accuracy: 0.9471
Epoch 167/200
126/126 [==============================] - 5s 37ms/step - loss: 0.1270 -
accuracy: 0.9514
Epoch 168/200
126/126 [==============================] - 5s 38ms/step - loss: 0.1262 -
accuracy: 0.9504
Epoch 169/200
126/126 [==============================] - 5s 37ms/step - loss: 0.1266 -
accuracy: 0.9514
Epoch 170/200
126/126 [==============================] - 5s 36ms/step - loss: 0.1260 -
accuracy: 0.9521
Epoch 171/200
126/126 [==============================] - 5s 36ms/step - loss: 0.1267 -
accuracy: 0.9496
Epoch 172/200
126/126 [==============================] - 5s 37ms/step - loss: 0.1263 -
accuracy: 0.9506
Epoch 173/200
126/126 [==============================] - 5s 36ms/step - loss: 0.1265 -
accuracy: 0.9496
Epoch 174/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1262 -
accuracy: 0.9501
Epoch 175/200
126/126 [==============================] - 5s 36ms/step - loss: 0.1258 -
accuracy: 0.9499
Epoch 176/200
126/126 [==============================] - 5s 37ms/step - loss: 0.1262 -
accuracy: 0.9484
Epoch 177/200
126/126 [==============================] - 5s 36ms/step - loss: 0.1260 -
accuracy: 0.9499
Epoch 178/200
126/126 [==============================] - 5s 36ms/step - loss: 0.1260 -
accuracy: 0.9514
Epoch 179/200
126/126 [==============================] - 5s 36ms/step - loss: 0.1259 -
accuracy: 0.9501
Epoch 180/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1258 -
accuracy: 0.9516
Epoch 181/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1256 -
accuracy: 0.9501
```

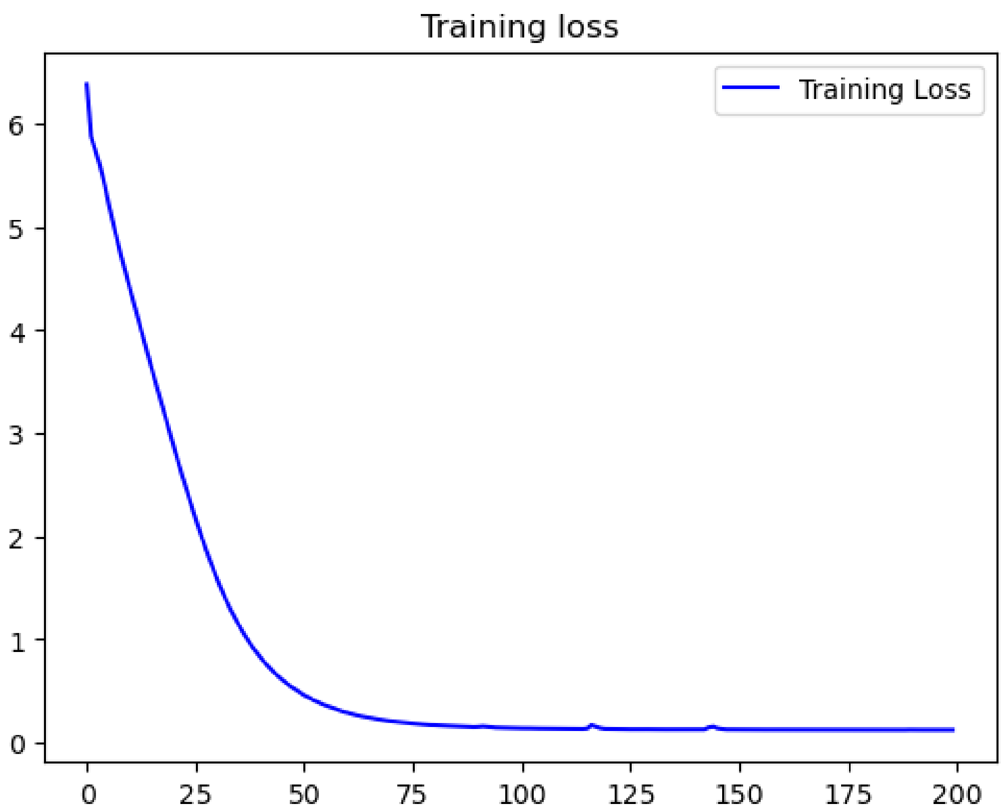
```
Epoch 182/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1257 -
accuracy: 0.9506
Epoch 183/200
126/126 [==============================] - 5s 36ms/step - loss: 0.1257 -
accuracy: 0.9501
Epoch 184/200
126/126 [==============================] - 4s 36ms/step - loss: 0.1255 -
accuracy: 0.9509
Epoch 185/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1253 -
accuracy: 0.9489
Epoch 186/200
126/126 [==============================] - 5s 36ms/step - loss: 0.1255 -
accuracy: 0.9509
Epoch 187/200
126/126 [==============================] - 5s 36ms/step - loss: 0.1252 -
accuracy: 0.9514
Epoch 188/200
126/126 [==============================] - 5s 36ms/step - loss: 0.1253 -
accuracy: 0.9501
Epoch 189/200
126/126 [==============================] - 5s 36ms/step - loss: 0.1252 -
accuracy: 0.9504
Epoch 190/200
126/126 [==============================] - 5s 36ms/step - loss: 0.1263 -
accuracy: 0.9491
Epoch 191/200
126/126 [==============================] - 5s 36ms/step - loss: 0.1254 -
accuracy: 0.9514
Epoch 192/200
126/126 [==============================] - 5s 36ms/step - loss: 0.1252 -
accuracy: 0.9501
Epoch 193/200
126/126 [==============================] - 5s 36ms/step - loss: 0.1254 -
accuracy: 0.9496
Epoch 194/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1251 -
accuracy: 0.9501
Epoch 195/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1258 -
accuracy: 0.9496
Epoch 196/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1251 -
accuracy: 0.9491
Epoch 197/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1246 -
accuracy: 0.9506
```

```
Epoch 198/200
126/126 [==============================] - 4s 35ms/step - loss: 0.1253 -
accuracy: 0.9504
Epoch 199/200
126/126 [==============================] - 5s 36ms/step - loss: 0.1252 -
accuracy: 0.9499
Epoch 200/200
126/126 [==============================] - 5s 36ms/step - loss: 0.1250 -
accuracy: 0.9501
```

Let's plot details of our training.

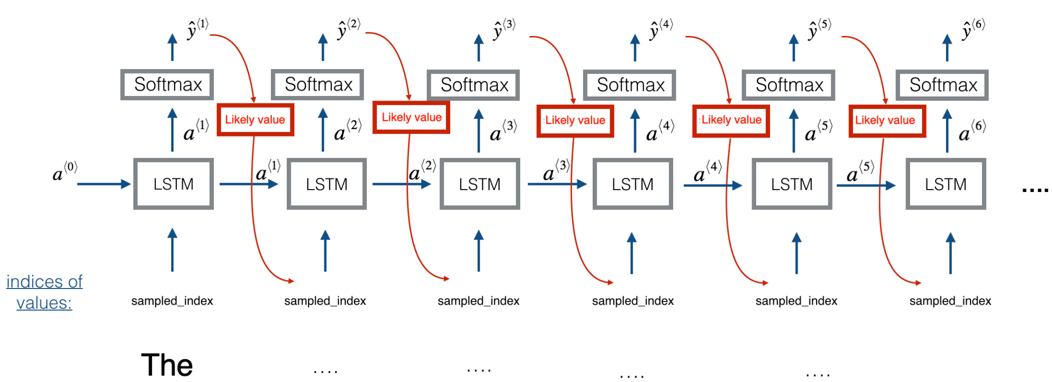
[15]:
```
acc = history.history['accuracy']
loss = history.history['loss']
epochs = range(len(acc))
plt.plot(epochs, acc, 'b', label='Training accuracy')
plt.title('Training accuracy')
plt.figure()
plt.plot(epochs, loss, 'b', label='Training Loss')
plt.title('Training loss')
plt.legend()
plt.show()
```





2 Generating fake comments

To generate fake tweets, we use the below architecture:



The idea is to give one or more starting token(s) to our model, and generate the next tokens until we generate ..

At each step, we select the token with the highest probability as our next token and generate the next one similartly using `model.predict_classes()`.

Note: The model takes as input the activation `a` from the previous state of the LSTM and the token chosen, forward propagate by one step, and get a new output activation `a`. The new activation `a` can then be used to generate the output, using the `dense` layer with `softmax` activation as before.

Task 2: Implement `generate()`.

---

[16]:
```
#TASK 2
# Implement the generate() function

def generate(seed_text):
    generated_text = seed_text.lower()
    seed_sequence = tokenizer.texts_to_sequences([seed_text])[0]
    num_tokens_to_generate = 20
    for _ in range(num_tokens_to_generate):
        input_sequence = pad_sequences([seed_sequence],␣
 ↪maxlen=max_sequence_len-1, padding='pre')
        predicted_probabilities = model.predict(input_sequence, verbose=0)[0]
        predicted_token_index = np.argmax(predicted_probabilities)
        predicted_token = tokenizer.index_word[predicted_token_index]

        generated_text += ' ' + predicted_token
        seed_sequence.append(predicted_token_index)

        if predicted_token == '.':
            break

    return generated_text
```

Let's test it:

[17]:
```
print(generate("COVID19 virus"))
print(generate("COVID19 is the"))
print(generate("The usa is"))
print(generate("The new virus"))
print(generate("China has"))
```

```
covid19 virus survives on surfaces for 7 days .
covid19 is the deadliest virus known to humans .
the usa is a cup of hot water can save your life .
the new virus that the covid19 can be transmitted through the eyes .
china has just announced that the â€˜biological' lab in wuhan where the covid19
virus was created was â€˜funded' by president barak sp
```

Let's test it in an interactive mode:

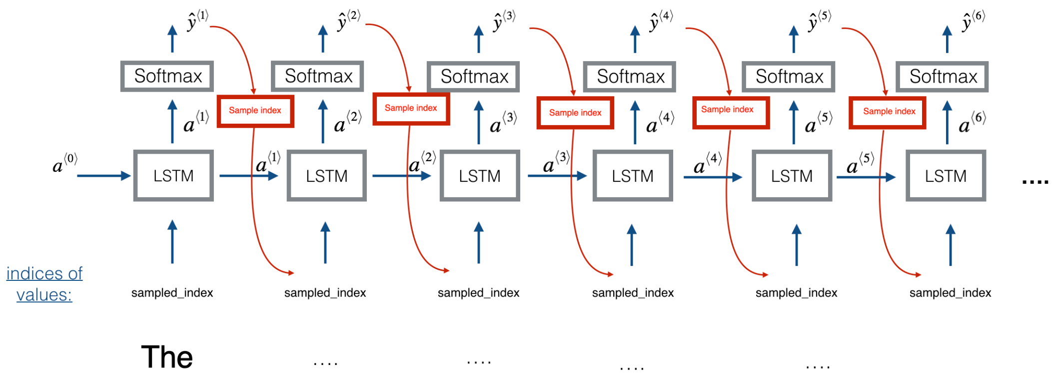
[19]:
```
usr_input = input("Write the beginning of your tweet, the algorithm machine␣
 ↪will complete it. Your input is: ")
for w in generate(usr_input).split():
```

```
    print(w, end =" ")
    time.sleep(0.4)
```

```
Write the beginning of your tweet, the algorithm machine will complete it. Your
input is: India can
india can detain anyone with a fever â€" indefinitely .
```

3 Generating text by sampling

The previous part is generating text by choosing the token with the highest probability. Now, we sill generate text by sampling as shown in the architecture below:



TASK 3: Implement the `generate_sample()` function. To sample a token from the output at each timestep, you need to use the following two functions: - `model.predict_proba()`: To get probabilities from the output layer. - `np.random.choice()`: To sample from the token list using the probaility array of each token.

- Since the "model.predict_proba()" is not availble in the tensorflow version,as in Keras, we will use the method predict_proba and use np.random.choice() as selection function to choose the token instead of np.argmax.

[20]:
```
#TASK 3
# Implement the generate_sample() function

def generate_sample(seed_text):
    generated_text = seed_text.lower()
    seed_sequence = tokenizer.texts_to_sequences([seed_text])[0]
    num_tokens_to_generate = 20


    for _ in range(num_tokens_to_generate):
        input_sequence = pad_sequences([seed_sequence],␣
 ↪maxlen=max_sequence_len-1, padding='pre')
        predicted_probabilities = model.predict(input_sequence, verbose = 0)[0]
```

```
        predicted_token_index = np.random.choice(len(predicted_probabilities),␣
↪p=predicted_probabilities)
        predicted_token = tokenizer.index_word[predicted_token_index]

        generated_text += ' ' + predicted_token
        seed_sequence.append(predicted_token_index)

        if predicted_token == '.':
            break

    return generated_text
```

Let's test it in an interactive mode:

[21]:
```
usr_input = input("Write the beginning of your tweet, the algorithm machine␣
↪will complete it. Your input is: ")
for w in generate_sample(usr_input).split():
    print(w, end =" ")
    time.sleep(0.4)
```

```
Write the beginning of your tweet, the algorithm machine will complete it. Your
input is: Elon musk says
elon musk says the coronavirus was engineered by scientists in a lab .
```

We can see that the Deep Learning Model has generated a fake Elon Musk's tweet related to the Corona Virus.

4 Generate your own text

Below, use you own data to generate content for a different application:

We've created a corpus focused on space related matters, incorporating fabricated tweets and comments. This is intended to train the model, enabling it to produce fake tweets regarding space missions and breakthroughs.

[23]:
```
tokenizer = Tokenizer(filters='"#$%&()*+-/:;<=>@[\\]^_`{|}~\t\n')
data = open('space_fake.txt').read().replace(".", " . ").replace(",", " , ").
↪replace("?", " ? ").replace("!", " ! ")
```

[24]:
```
corpus = data.lower().split("\n")
tokenizer.fit_on_texts(corpus)
total_words = len(tokenizer.word_index) + 1
```

[25]:
```
print("Example of fake tweets: ",corpus[:2])
print("Size of the vocabulary = ", total_words)
index = [(k, v) for k, v in tokenizer.word_index.items()]
print("Example of our word index = ", index[0:10])
```

```
Example of fake tweets:  ['space travel has always captured the imagination of
```

```
humanity . ', 'exploring the cosmos is a remarkable feat of human engineering
and ambition . ']
Size of the vocabulary =  1048
Example of our word index =  [('.', 1), ('the', 2), ('space', 3), ('of', 4),
('a', 5), ('to', 6), ('and', 7), ('concept', 8), (',', 9), ('for', 10)]
```

[26]:
```
input_sequences = []
for line in corpus:
    token_list = tokenizer.texts_to_sequences([line])[0]
    for i in range(1, len(token_list)):
        n_gram_sequence = token_list[:i+1]
        input_sequences.append(n_gram_sequence)
```

[27]:
```
sample = 20
reverse_word_map = dict(map(reversed, tokenizer.word_index.items()))
print("The entry ",sample," in 'input_sequences' is: ")
print(input_sequences[sample])
print(" and it corresponds to:")
for i in input_sequences[sample]:
    print(reverse_word_map[i], end=' ')
```

```
The entry  20  in 'input_sequences' is:
[390, 2, 41, 13, 5, 391, 392, 4, 76, 135, 7, 393, 1]
 and it corresponds to:
exploring the cosmos is a remarkable feat of human engineering and ambition .
```

[28]:
```
max_sequence_len = max([len(x) for x in input_sequences])
input_sequences = np.array(pad_sequences(input_sequences,␣
 ↪maxlen=max_sequence_len, padding='pre'))
```

[29]:
```
reverse_word_map = dict(map(reversed, tokenizer.word_index.items()))
print("The entry ",sample," in 'input_sequences' is: ")
print(input_sequences[sample])
print(" and it corresponds to:")
print("[", end=' ')
for i in input_sequences[sample]:
    if i in reverse_word_map:
        print(reverse_word_map[i], end=' ')
    else:
        print("__", end=' ')
print("]")
```

```
The entry  20  in 'input_sequences' is:
[  0   0   0   0   0   0 390   2  41  13   5 391 392   4  76 135   7 393
   1]
 and it corresponds to:
[ __ __ __ __ __ __ exploring the cosmos is a remarkable feat of human
```

```
engineering and ambition . ]
```

[30]:
```python
input_to_model, label = input_sequences[:,:-1],input_sequences[:,-1]
```

[31]:
```python
print("The entry ",sample," in 'input_sequences' is: ")
print(input_sequences[sample])
print(", it corresponds to the following input to our model:")
print(input_to_model[sample])
print(" and the following output: ", label[sample])
```

```
The entry  20  in 'input_sequences' is:
[  0   0   0   0   0   0 390   2  41  13   5 391 392   4  76 135   7 393
   1]
, it corresponds to the following input to our model:
[  0   0   0   0   0   0 390   2  41  13   5 391 392   4  76 135   7 393]
 and the following output:  1
```

[32]:
```python
label = ku.to_categorical(label, num_classes=total_words)
```

[33]:
```python
#TASK 1
# deep_fake_comment_model

def deep_fake_comment_model():
    model = Sequential()
    model.add(Embedding(input_dim=total_words, output_dim=128,␣
 ↪input_length=max_sequence_len - 1))
    model.add(LSTM(128))
    model.add(Dense(total_words, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])

    return model

#Print details of the model.
model = deep_fake_comment_model()
```

[34]:
```python
history = model.fit(input_to_model, label, epochs=200, batch_size=32, verbose=1)
```

```
Epoch 1/200
95/95 [==============================] - 4s 16ms/step - loss: 6.3484 - accuracy:
0.0865
Epoch 2/200
95/95 [==============================] - 1s 15ms/step - loss: 5.7949 - accuracy:
0.0931
Epoch 3/200
95/95 [==============================] - 1s 15ms/step - loss: 5.5087 - accuracy:
0.1323
Epoch 4/200
95/95 [==============================] - 1s 15ms/step - loss: 5.2665 - accuracy:
```

```
0.1612
Epoch 5/200
95/95 [==============================] - 1s 16ms/step - loss: 5.0484 - accuracy:
0.1767
Epoch 6/200
95/95 [==============================] - 1s 15ms/step - loss: 4.8610 - accuracy:
0.1958
Epoch 7/200
95/95 [==============================] - 1s 15ms/step - loss: 4.6851 - accuracy:
0.2122
Epoch 8/200
95/95 [==============================] - 1s 15ms/step - loss: 4.5064 - accuracy:
0.2300
Epoch 9/200
95/95 [==============================] - 1s 15ms/step - loss: 4.3294 - accuracy:
0.2425
Epoch 10/200
95/95 [==============================] - 1s 15ms/step - loss: 4.1530 - accuracy:
0.2521
Epoch 11/200
95/95 [==============================] - 1s 15ms/step - loss: 3.9676 - accuracy:
0.2757
Epoch 12/200
95/95 [==============================] - 1s 15ms/step - loss: 3.7935 - accuracy:
0.2869
Epoch 13/200
95/95 [==============================] - 1s 15ms/step - loss: 3.6307 - accuracy:
0.2985
Epoch 14/200
95/95 [==============================] - 1s 15ms/step - loss: 3.4688 - accuracy:
0.3162
Epoch 15/200
95/95 [==============================] - 1s 15ms/step - loss: 3.3098 - accuracy:
0.3310
Epoch 16/200
95/95 [==============================] - 1s 15ms/step - loss: 3.1497 - accuracy:
0.3521
Epoch 17/200
95/95 [==============================] - 1s 15ms/step - loss: 3.0012 - accuracy:
0.3787
Epoch 18/200
95/95 [==============================] - 1s 15ms/step - loss: 2.8550 - accuracy:
0.4021
Epoch 19/200
95/95 [==============================] - 1s 15ms/step - loss: 2.7117 - accuracy:
0.4268
Epoch 20/200
95/95 [==============================] - 1s 15ms/step - loss: 2.5686 - accuracy:
```

```
0.4584
Epoch 21/200
95/95 [==============================] - 1s 15ms/step - loss: 2.4248 - accuracy:
0.4952
Epoch 22/200
95/95 [==============================] - 1s 15ms/step - loss: 2.2944 - accuracy:
0.5334
Epoch 23/200
95/95 [==============================] - 1s 15ms/step - loss: 2.1571 - accuracy:
0.5762
Epoch 24/200
95/95 [==============================] - 1s 15ms/step - loss: 2.0335 - accuracy:
0.5972
Epoch 25/200
95/95 [==============================] - 1s 15ms/step - loss: 1.9071 - accuracy:
0.6364
Epoch 26/200
95/95 [==============================] - 1s 15ms/step - loss: 1.7890 - accuracy:
0.6660
Epoch 27/200
95/95 [==============================] - 1s 15ms/step - loss: 1.6778 - accuracy:
0.6986
Epoch 28/200
95/95 [==============================] - 1s 15ms/step - loss: 1.5701 - accuracy:
0.7233
Epoch 29/200
95/95 [==============================] - 1s 15ms/step - loss: 1.4692 - accuracy:
0.7486
Epoch 30/200
95/95 [==============================] - 1s 15ms/step - loss: 1.3751 - accuracy:
0.7674
Epoch 31/200
95/95 [==============================] - 1s 15ms/step - loss: 1.2853 - accuracy:
0.7874
Epoch 32/200
95/95 [==============================] - 1s 15ms/step - loss: 1.2030 - accuracy:
0.7976
Epoch 33/200
95/95 [==============================] - 1s 15ms/step - loss: 1.1265 - accuracy:
0.8164
Epoch 34/200
95/95 [==============================] - 1s 15ms/step - loss: 1.0579 - accuracy:
0.8263
Epoch 35/200
95/95 [==============================] - 1s 15ms/step - loss: 0.9950 - accuracy:
0.8378
Epoch 36/200
95/95 [==============================] - 1s 15ms/step - loss: 0.9357 - accuracy:
```

```
0.8463
Epoch 37/200
95/95 [==============================] - 1s 15ms/step - loss: 0.8866 - accuracy:
0.8532
Epoch 38/200
95/95 [==============================] - 1s 15ms/step - loss: 0.8353 - accuracy:
0.8605
Epoch 39/200
95/95 [==============================] - 1s 15ms/step - loss: 0.7927 - accuracy:
0.8651
Epoch 40/200
95/95 [==============================] - 1s 15ms/step - loss: 0.7539 - accuracy:
0.8700
Epoch 41/200
95/95 [==============================] - 1s 15ms/step - loss: 0.7178 - accuracy:
0.8710
Epoch 42/200
95/95 [==============================] - 1s 15ms/step - loss: 0.6842 - accuracy:
0.8786
Epoch 43/200
95/95 [==============================] - 1s 15ms/step - loss: 0.6537 - accuracy:
0.8789
Epoch 44/200
95/95 [==============================] - 1s 15ms/step - loss: 0.6258 - accuracy:
0.8835
Epoch 45/200
95/95 [==============================] - 1s 15ms/step - loss: 0.6013 - accuracy:
0.8832
Epoch 46/200
95/95 [==============================] - 1s 15ms/step - loss: 0.5782 - accuracy:
0.8894
Epoch 47/200
95/95 [==============================] - 1s 15ms/step - loss: 0.5571 - accuracy:
0.8868
Epoch 48/200
95/95 [==============================] - 1s 15ms/step - loss: 0.5379 - accuracy:
0.8888
Epoch 49/200
95/95 [==============================] - 1s 15ms/step - loss: 0.5219 - accuracy:
0.8914
Epoch 50/200
95/95 [==============================] - 1s 15ms/step - loss: 0.5060 - accuracy:
0.8921
Epoch 51/200
95/95 [==============================] - 1s 15ms/step - loss: 0.4917 - accuracy:
0.8927
Epoch 52/200
95/95 [==============================] - 1s 15ms/step - loss: 0.4772 - accuracy:
```

```
0.8957
Epoch 53/200
95/95 [==============================] - 1s 15ms/step - loss: 0.4670 - accuracy:
0.8954
Epoch 54/200
95/95 [==============================] - 1s 15ms/step - loss: 0.4545 - accuracy:
0.8973
Epoch 55/200
95/95 [==============================] - 1s 15ms/step - loss: 0.4439 - accuracy:
0.8944
Epoch 56/200
95/95 [==============================] - 1s 15ms/step - loss: 0.4344 - accuracy:
0.8957
Epoch 57/200
95/95 [==============================] - 1s 15ms/step - loss: 0.4246 - accuracy:
0.8973
Epoch 58/200
95/95 [==============================] - 1s 15ms/step - loss: 0.4190 - accuracy:
0.8947
Epoch 59/200
95/95 [==============================] - 1s 15ms/step - loss: 0.4096 - accuracy:
0.8983
Epoch 60/200
95/95 [==============================] - 1s 15ms/step - loss: 0.4018 - accuracy:
0.8963
Epoch 61/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3962 - accuracy:
0.8963
Epoch 62/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3910 - accuracy:
0.8996
Epoch 63/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3852 - accuracy:
0.8963
Epoch 64/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3799 - accuracy:
0.8970
Epoch 65/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3761 - accuracy:
0.8970
Epoch 66/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3717 - accuracy:
0.9000
Epoch 67/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3678 - accuracy:
0.8944
Epoch 68/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3661 - accuracy:
```

```
0.8937
Epoch 69/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3614 - accuracy:
0.8970
Epoch 70/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3587 - accuracy:
0.8967
Epoch 71/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3554 - accuracy:
0.8983
Epoch 72/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3515 - accuracy:
0.8980
Epoch 73/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3487 - accuracy:
0.8980
Epoch 74/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3465 - accuracy:
0.8980
Epoch 75/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3444 - accuracy:
0.8993
Epoch 76/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3435 - accuracy:
0.8987
Epoch 77/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3394 - accuracy:
0.8993
Epoch 78/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3388 - accuracy:
0.8993
Epoch 79/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3372 - accuracy:
0.8967
Epoch 80/200
95/95 [==============================] - 2s 16ms/step - loss: 0.3365 - accuracy:
0.8970
Epoch 81/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3584 - accuracy:
0.8937
Epoch 82/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3701 - accuracy:
0.8914
Epoch 83/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3526 - accuracy:
0.8924
Epoch 84/200
95/95 [==============================] - 2s 16ms/step - loss: 0.3376 - accuracy:
```

```
0.8990
Epoch 85/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3299 - accuracy:
0.8990
Epoch 86/200
95/95 [==============================] - 2s 16ms/step - loss: 0.3282 - accuracy:
0.8996
Epoch 87/200
95/95 [==============================] - 1s 16ms/step - loss: 0.3258 - accuracy:
0.8980
Epoch 88/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3255 - accuracy:
0.8990
Epoch 89/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3255 - accuracy:
0.8967
Epoch 90/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3251 - accuracy:
0.8963
Epoch 91/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3230 - accuracy:
0.8977
Epoch 92/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3219 - accuracy:
0.8993
Epoch 93/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3200 - accuracy:
0.8973
Epoch 94/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3204 - accuracy:
0.8990
Epoch 95/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3192 - accuracy:
0.8987
Epoch 96/200
95/95 [==============================] - 1s 16ms/step - loss: 0.3189 - accuracy:
0.8963
Epoch 97/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3183 - accuracy:
0.8973
Epoch 98/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3180 - accuracy:
0.8950
Epoch 99/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3160 - accuracy:
0.8993
Epoch 100/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3152 - accuracy:
```

```
0.8983
Epoch 101/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3153 - accuracy:
0.8980
Epoch 102/200
95/95 [==============================] - 2s 16ms/step - loss: 0.3142 - accuracy:
0.9006
Epoch 103/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3145 - accuracy:
0.8983
Epoch 104/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3132 - accuracy:
0.8980
Epoch 105/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3131 - accuracy:
0.8973
Epoch 106/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3123 - accuracy:
0.8977
Epoch 107/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3137 - accuracy:
0.8977
Epoch 108/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3120 - accuracy:
0.8993
Epoch 109/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3118 - accuracy:
0.8983
Epoch 110/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3105 - accuracy:
0.8996
Epoch 111/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3111 - accuracy:
0.8980
Epoch 112/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3103 - accuracy:
0.8977
Epoch 113/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3107 - accuracy:
0.8970
Epoch 114/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3090 - accuracy:
0.8983
Epoch 115/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3095 - accuracy:
0.9006
Epoch 116/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3098 - accuracy:
```

```
0.9003
Epoch 117/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3099 - accuracy:
0.8990
Epoch 118/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3100 - accuracy:
0.8960
Epoch 119/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3109 - accuracy:
0.8970
Epoch 120/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3086 - accuracy:
0.8993
Epoch 121/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3083 - accuracy:
0.8963
Epoch 122/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3072 - accuracy:
0.8977
Epoch 123/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3073 - accuracy:
0.8980
Epoch 124/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3063 - accuracy:
0.8987
Epoch 125/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3078 - accuracy:
0.8996
Epoch 126/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3077 - accuracy:
0.8960
Epoch 127/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3085 - accuracy:
0.8960
Epoch 128/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3298 - accuracy:
0.8944
Epoch 129/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3414 - accuracy:
0.8911
Epoch 130/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3151 - accuracy:
0.8983
Epoch 131/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3108 - accuracy:
0.8983
Epoch 132/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3069 - accuracy:
```

```
0.9000
Epoch 133/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3051 - accuracy:
0.8963
Epoch 134/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3060 - accuracy:
0.8996
Epoch 135/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3060 - accuracy:
0.8980
Epoch 136/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3053 - accuracy:
0.8990
Epoch 137/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3048 - accuracy:
0.8963
Epoch 138/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3048 - accuracy:
0.8993
Epoch 139/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3045 - accuracy:
0.8977
Epoch 140/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3051 - accuracy:
0.8973
Epoch 141/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3036 - accuracy:
0.8983
Epoch 142/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3036 - accuracy:
0.8947
Epoch 143/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3041 - accuracy:
0.9003
Epoch 144/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3039 - accuracy:
0.8983
Epoch 145/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3045 - accuracy:
0.9000
Epoch 146/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3034 - accuracy:
0.8967
Epoch 147/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3039 - accuracy:
0.9006
Epoch 148/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3039 - accuracy:
```

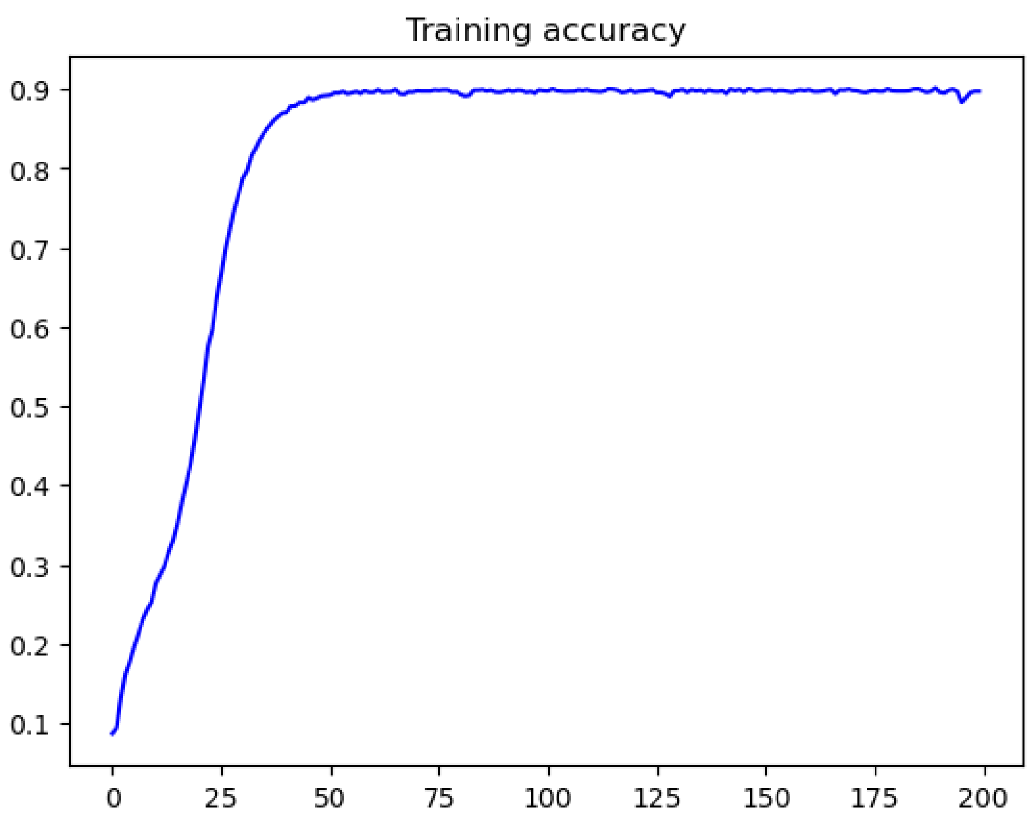
```
0.8993
Epoch 149/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3036 - accuracy:
0.8973
Epoch 150/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3031 - accuracy:
0.8987
Epoch 151/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3033 - accuracy:
0.8993
Epoch 152/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3025 - accuracy:
0.8996
Epoch 153/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3027 - accuracy:
0.8973
Epoch 154/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3019 - accuracy:
0.8987
Epoch 155/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3024 - accuracy:
0.8987
Epoch 156/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3020 - accuracy:
0.8977
Epoch 157/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3027 - accuracy:
0.8967
Epoch 158/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3028 - accuracy:
0.8983
Epoch 159/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3022 - accuracy:
0.8993
Epoch 160/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3024 - accuracy:
0.8983
Epoch 161/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3033 - accuracy:
0.8996
Epoch 162/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3023 - accuracy:
0.8973
Epoch 163/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3022 - accuracy:
0.8973
Epoch 164/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3021 - accuracy:
```

```
0.8980
Epoch 165/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3021 - accuracy:
0.8990
Epoch 166/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3012 - accuracy:
0.9000
Epoch 167/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3022 - accuracy:
0.8944
Epoch 168/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3014 - accuracy:
0.8993
Epoch 169/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3018 - accuracy:
0.8990
Epoch 170/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3014 - accuracy:
0.9003
Epoch 171/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3017 - accuracy:
0.8987
Epoch 172/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3017 - accuracy:
0.8983
Epoch 173/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3013 - accuracy:
0.8967
Epoch 174/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3020 - accuracy:
0.8960
Epoch 175/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3015 - accuracy:
0.8980
Epoch 176/200
95/95 [==============================] - 1s 16ms/step - loss: 0.3006 - accuracy:
0.8987
Epoch 177/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3027 - accuracy:
0.8977
Epoch 178/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3015 - accuracy:
0.8977
Epoch 179/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3018 - accuracy:
0.9006
Epoch 180/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3012 - accuracy:
```

```
0.8983
Epoch 181/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3013 - accuracy:
0.8980
Epoch 182/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3011 - accuracy:
0.8980
Epoch 183/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3013 - accuracy:
0.8980
Epoch 184/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3014 - accuracy:
0.8983
Epoch 185/200
95/95 [==============================] - 1s 15ms/step - loss: 0.2998 - accuracy:
0.9003
Epoch 186/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3004 - accuracy:
0.9006
Epoch 187/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3014 - accuracy:
0.8983
Epoch 188/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3010 - accuracy:
0.8963
Epoch 189/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3013 - accuracy:
0.8977
Epoch 190/200
95/95 [==============================] - 1s 15ms/step - loss: 0.2995 - accuracy:
0.9019
Epoch 191/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3006 - accuracy:
0.8963
Epoch 192/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3006 - accuracy:
0.8954
Epoch 193/200
95/95 [==============================] - 1s 15ms/step - loss: 0.2997 - accuracy:
0.8987
Epoch 194/200
95/95 [==============================] - 1s 15ms/step - loss: 0.3009 - accuracy:
0.9003
Epoch 195/200
95/95 [==============================] - 1s 16ms/step - loss: 0.3024 - accuracy:
0.8973
Epoch 196/200
95/95 [==============================] - 2s 16ms/step - loss: 0.3632 - accuracy:
```

```
0.8835
Epoch 197/200
95/95 [==============================] - 1s 16ms/step - loss: 0.3371 - accuracy:
0.8894
Epoch 198/200
95/95 [==============================] - 1s 16ms/step - loss: 0.3057 - accuracy:
0.8960
Epoch 199/200
95/95 [==============================] - 2s 16ms/step - loss: 0.3031 - accuracy:
0.8977
Epoch 200/200
95/95 [==============================] - 1s 16ms/step - loss: 0.3011 - accuracy:
0.8977
```

[35]:
```python
acc = history.history['accuracy']
loss = history.history['loss']
epochs = range(len(acc))
plt.plot(epochs, acc, 'b', label='Training accuracy')
plt.title('Training accuracy')
plt.figure()
plt.plot(epochs, loss, 'b', label='Training Loss')
plt.title('Training loss')
plt.legend()
plt.show()
```



[36]:
```
#TASK 2
# Implement the generate() function

def generate(seed_text):
    generated_text = seed_text.lower()
    seed_sequence = tokenizer.texts_to_sequences([seed_text])[0]
    num_tokens_to_generate = 20
    for _ in range(num_tokens_to_generate):
        input_sequence = pad_sequences([seed_sequence],␣
↪maxlen=max_sequence_len-1, padding='pre')
        predicted_probabilities = model.predict(input_sequence, verbose=0)[0]
        predicted_token_index = np.argmax(predicted_probabilities)
        predicted_token = tokenizer.index_word[predicted_token_index]

        generated_text += ' ' + predicted_token
        seed_sequence.append(predicted_token_index)

        if predicted_token == '.':
```

```
            break

    return generated_text
```

[38]:
```
print(generate("India has"))
print(generate("Space Exploration is"))
print(generate("NASA is known"))
print(generate("Rockets habe been"))
print(generate("Asteroids can"))
```

```
india has starshot envisions sending spacecraft to nearby stars .
space exploration is emerging as a potential industry , offering civilians a
chance to experience space .
nasa is known to be responsible for the red planet's terraforming process .
rockets habe been mining is being explored as a way to obtain valuable resources
for astronauts .
asteroids can disrupt black hole sing alongs with surprise visits .
```

[39]:
```
#TASK 3
# Implement the generate_sample() function

def generate_sample(seed_text):
    generated_text = seed_text.lower()
    seed_sequence = tokenizer.texts_to_sequences([seed_text])[0]
    num_tokens_to_generate = 20


    for _ in range(num_tokens_to_generate):
        input_sequence = pad_sequences([seed_sequence],␣
↪maxlen=max_sequence_len-1, padding='pre')
        predicted_probabilities = model.predict(input_sequence, verbose = 0)[0]
        predicted_token_index = np.random.choice(len(predicted_probabilities),␣
↪p=predicted_probabilities)
        predicted_token = tokenizer.index_word[predicted_token_index]

        generated_text += ' ' + predicted_token
        seed_sequence.append(predicted_token_index)

        if predicted_token == '.':
            break

    return generated_text
```

[43]:
```
generate_sample("Aliens have been")
```

[43]:
```
'aliens have been known to challenge black holes to interdimensional chess
matches .'
```

It's evident that the Deep Learning Model has produced a completely imaginative and fake tweet connected to space-related topics.

5 Congratulations!

You've come to the end of this assignment, and have seen how to build a deep learning architecture that generate fake tweets/comments.

Congratulations on finishing this notebook!