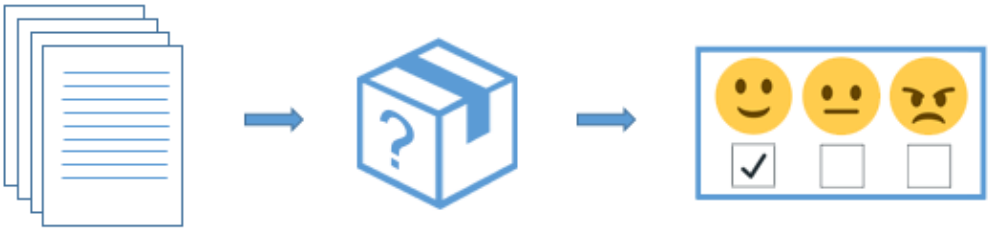
5.2D - DeepFakeComments Generator

August 15, 2022

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
import random
```

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:
```

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

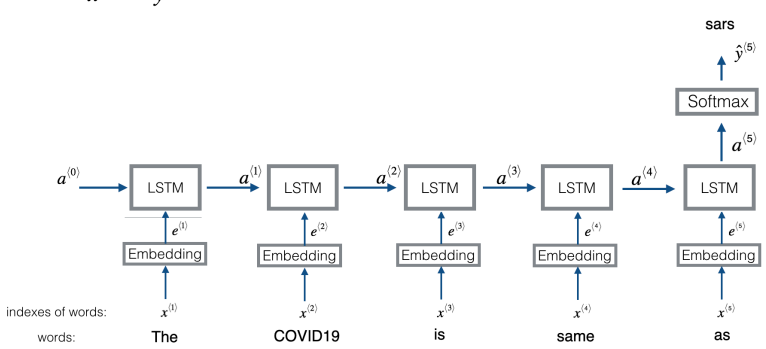
```
The entry  20  in 'input_sequences' is:
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   2   3  12 187  34 188]
, it corresponds 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]:
```
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]:
```
#TASK 1
# deep_fake_comment_model

def deep_fake_comment_model():
    ### START CODE HERE ###
    model = Sequential(name="DeepFakeComment")
    model.add(Embedding(total_words, 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(lr=0.001),␣
 ↪metrics=['accuracy'])
    ### END CODE HERE ###
    return model
#Print details of the model.
model = deep_fake_comment_model()
model.summary()
```

```
Model: "DeepFakeComment"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
```

```
 embedding (Embedding)       (None, 60, 128)           160896

 lstm (LSTM)                 (None, 128)               131584

 dense (Dense)               (None, 1257)              162153

=================================================================
Total params: 454,633
Trainable params: 454,633
Non-trainable params: 0
_________________________________________________________________

c:\users\shane\desktop\5.2d - deepfakecomments-generator\deepfakecomments-
generator\lib\site-packages\keras\optimizers\optimizer_v2\adam.py:110:
UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
```

Now, let's start our training.

[13]:
```
history = model.fit(input_to_model, label, epochs=200, batch_size=32, verbose=1)
```

```
Epoch 1/200
126/126 [==============================] - 8s 54ms/step - loss: 6.3738 -
accuracy: 0.0665
Epoch 2/200
126/126 [==============================] - 7s 56ms/step - loss: 5.8747 -
accuracy: 0.0767
Epoch 3/200
126/126 [==============================] - 7s 56ms/step - loss: 5.7285 -
accuracy: 0.0926
Epoch 4/200
126/126 [==============================] - 7s 57ms/step - loss: 5.5492 -
accuracy: 0.1199
Epoch 5/200
126/126 [==============================] - 7s 55ms/step - loss: 5.3410 -
accuracy: 0.1330
Epoch 6/200
126/126 [==============================] - 7s 55ms/step - loss: 5.1264 -
accuracy: 0.1509
Epoch 7/200
126/126 [==============================] - 7s 55ms/step - loss: 4.9385 -
accuracy: 0.1633
Epoch 8/200
126/126 [==============================] - 7s 55ms/step - loss: 4.7601 -
accuracy: 0.1739
Epoch 9/200
126/126 [==============================] - 7s 54ms/step - loss: 4.5863 -
accuracy: 0.1906
Epoch 10/200
```

```
126/126 [==============================] - 7s 54ms/step - loss: 4.4224 -
accuracy: 0.2047
Epoch 11/200
126/126 [==============================] - 7s 58ms/step - loss: 4.2517 -
accuracy: 0.2134
Epoch 12/200
126/126 [==============================] - 7s 55ms/step - loss: 4.0896 -
accuracy: 0.2323
Epoch 13/200
126/126 [==============================] - 7s 56ms/step - loss: 3.9231 -
accuracy: 0.2501
Epoch 14/200
126/126 [==============================] - 7s 55ms/step - loss: 3.7609 -
accuracy: 0.2682
Epoch 15/200
126/126 [==============================] - 7s 55ms/step - loss: 3.5985 -
accuracy: 0.2861
Epoch 16/200
126/126 [==============================] - 7s 55ms/step - loss: 3.4362 -
accuracy: 0.3065
Epoch 17/200
126/126 [==============================] - 7s 55ms/step - loss: 3.2717 -
accuracy: 0.3295
Epoch 18/200
126/126 [==============================] - 7s 55ms/step - loss: 3.1142 -
accuracy: 0.3591
Epoch 19/200
126/126 [==============================] - 7s 55ms/step - loss: 2.9581 -
accuracy: 0.3831
Epoch 20/200
126/126 [==============================] - 7s 56ms/step - loss: 2.8052 -
accuracy: 0.4087
Epoch 21/200
126/126 [==============================] - 7s 55ms/step - loss: 2.6498 -
accuracy: 0.4392
Epoch 22/200
126/126 [==============================] - 7s 56ms/step - loss: 2.4994 -
accuracy: 0.4700
Epoch 23/200
126/126 [==============================] - 7s 57ms/step - loss: 2.3553 -
accuracy: 0.5079
Epoch 24/200
126/126 [==============================] - 7s 55ms/step - loss: 2.2171 -
accuracy: 0.5412
Epoch 25/200
126/126 [==============================] - 7s 55ms/step - loss: 2.0803 -
accuracy: 0.5804
Epoch 26/200
```

```
126/126 [==============================] - 7s 55ms/step - loss: 1.9459 -
accuracy: 0.6161
Epoch 27/200
126/126 [==============================] - 7s 56ms/step - loss: 1.8179 -
accuracy: 0.6452
Epoch 28/200
126/126 [==============================] - 7s 55ms/step - loss: 1.6991 -
accuracy: 0.6806
Epoch 29/200
126/126 [==============================] - 7s 55ms/step - loss: 1.5836 -
accuracy: 0.7102
Epoch 30/200
126/126 [==============================] - 7s 56ms/step - loss: 1.4750 -
accuracy: 0.7400
Epoch 31/200
126/126 [==============================] - 7s 55ms/step - loss: 1.3742 -
accuracy: 0.7603
Epoch 32/200
126/126 [==============================] - 7s 55ms/step - loss: 1.2820 -
accuracy: 0.7774
Epoch 33/200
126/126 [==============================] - 7s 55ms/step - loss: 1.1930 -
accuracy: 0.8012
Epoch 34/200
126/126 [==============================] - 7s 55ms/step - loss: 1.1142 -
accuracy: 0.8072
Epoch 35/200
126/126 [==============================] - 7s 55ms/step - loss: 1.0399 -
accuracy: 0.8328
Epoch 36/200
126/126 [==============================] - 7s 55ms/step - loss: 0.9679 -
accuracy: 0.8447
Epoch 37/200
126/126 [==============================] - 7s 56ms/step - loss: 0.9042 -
accuracy: 0.8546
Epoch 38/200
126/126 [==============================] - 7s 55ms/step - loss: 0.8452 -
accuracy: 0.8680
Epoch 39/200
126/126 [==============================] - 7s 55ms/step - loss: 0.7926 -
accuracy: 0.8762
Epoch 40/200
126/126 [==============================] - 7s 57ms/step - loss: 0.7414 -
accuracy: 0.8814
Epoch 41/200
126/126 [==============================] - 7s 56ms/step - loss: 0.6949 -
accuracy: 0.8898
Epoch 42/200
```

```
126/126 [==============================] - 7s 56ms/step - loss: 0.6533 -
accuracy: 0.8958
Epoch 43/200
126/126 [==============================] - 7s 55ms/step - loss: 0.6143 -
accuracy: 0.9045
Epoch 44/200
126/126 [==============================] - 7s 56ms/step - loss: 0.5775 -
accuracy: 0.9082
Epoch 45/200
126/126 [==============================] - 7s 55ms/step - loss: 0.5450 -
accuracy: 0.9141
Epoch 46/200
126/126 [==============================] - 7s 57ms/step - loss: 0.5120 -
accuracy: 0.9174
Epoch 47/200
126/126 [==============================] - 7s 56ms/step - loss: 0.4837 -
accuracy: 0.9218
Epoch 48/200
126/126 [==============================] - 7s 56ms/step - loss: 0.4593 -
accuracy: 0.9226
Epoch 49/200
126/126 [==============================] - 7s 56ms/step - loss: 0.4343 -
accuracy: 0.9263
Epoch 50/200
126/126 [==============================] - 7s 55ms/step - loss: 0.4109 -
accuracy: 0.9308
Epoch 51/200
126/126 [==============================] - 7s 55ms/step - loss: 0.3908 -
accuracy: 0.9328
Epoch 52/200
126/126 [==============================] - 7s 55ms/step - loss: 0.3714 -
accuracy: 0.9335
Epoch 53/200
126/126 [==============================] - 7s 55ms/step - loss: 0.3542 -
accuracy: 0.9340
Epoch 54/200
126/126 [==============================] - 7s 56ms/step - loss: 0.3370 -
accuracy: 0.9387
Epoch 55/200
126/126 [==============================] - 7s 55ms/step - loss: 0.3220 -
accuracy: 0.9375
Epoch 56/200
126/126 [==============================] - 7s 55ms/step - loss: 0.3097 -
accuracy: 0.9392
Epoch 57/200
126/126 [==============================] - 7s 55ms/step - loss: 0.2956 -
accuracy: 0.9419
Epoch 58/200
```

```
126/126 [==============================] - 7s 55ms/step - loss: 0.2835 -
accuracy: 0.9427
Epoch 59/200
126/126 [==============================] - 7s 56ms/step - loss: 0.2739 -
accuracy: 0.9442
Epoch 60/200
126/126 [==============================] - 7s 55ms/step - loss: 0.2641 -
accuracy: 0.9454
Epoch 61/200
126/126 [==============================] - 7s 55ms/step - loss: 0.2530 -
accuracy: 0.9479
Epoch 62/200
126/126 [==============================] - 7s 56ms/step - loss: 0.2461 -
accuracy: 0.9447
Epoch 63/200
126/126 [==============================] - 8s 60ms/step - loss: 0.2381 -
accuracy: 0.9464
Epoch 64/200
126/126 [==============================] - 8s 60ms/step - loss: 0.2292 -
accuracy: 0.9484
Epoch 65/200
126/126 [==============================] - 7s 58ms/step - loss: 0.2237 -
accuracy: 0.9479
Epoch 66/200
126/126 [==============================] - 8s 61ms/step - loss: 0.2172 -
accuracy: 0.9481
Epoch 67/200
126/126 [==============================] - 8s 60ms/step - loss: 0.2111 -
accuracy: 0.9459
Epoch 68/200
126/126 [==============================] - 7s 59ms/step - loss: 0.2055 -
accuracy: 0.9467
Epoch 69/200
126/126 [==============================] - 7s 57ms/step - loss: 0.2007 -
accuracy: 0.9484
Epoch 70/200
126/126 [==============================] - 9s 70ms/step - loss: 0.1973 -
accuracy: 0.9471
Epoch 71/200
126/126 [==============================] - 8s 64ms/step - loss: 0.1915 -
accuracy: 0.9501
Epoch 72/200
126/126 [==============================] - 7s 58ms/step - loss: 0.1879 -
accuracy: 0.9484
Epoch 73/200
126/126 [==============================] - 7s 56ms/step - loss: 0.1838 -
accuracy: 0.9481
Epoch 74/200
```

```
126/126 [==============================] - 7s 56ms/step - loss: 0.1815 -
accuracy: 0.9494
Epoch 75/200
126/126 [==============================] - 8s 64ms/step - loss: 0.1780 -
accuracy: 0.9479
Epoch 76/200
126/126 [==============================] - 8s 60ms/step - loss: 0.1759 -
accuracy: 0.9474
Epoch 77/200
126/126 [==============================] - 7s 57ms/step - loss: 0.1727 -
accuracy: 0.9491
Epoch 78/200
126/126 [==============================] - 7s 56ms/step - loss: 0.1701 -
accuracy: 0.9499
Epoch 79/200
126/126 [==============================] - 7s 55ms/step - loss: 0.1669 -
accuracy: 0.9476
Epoch 80/200
126/126 [==============================] - 7s 55ms/step - loss: 0.1649 -
accuracy: 0.9494
Epoch 81/200
126/126 [==============================] - 7s 55ms/step - loss: 0.1634 -
accuracy: 0.9489
Epoch 82/200
126/126 [==============================] - 7s 55ms/step - loss: 0.1596 -
accuracy: 0.9484
Epoch 83/200
126/126 [==============================] - 7s 55ms/step - loss: 0.1595 -
accuracy: 0.9491
Epoch 84/200
126/126 [==============================] - 7s 57ms/step - loss: 0.1578 -
accuracy: 0.9486
Epoch 85/200
126/126 [==============================] - 8s 63ms/step - loss: 0.1545 -
accuracy: 0.9496
Epoch 86/200
126/126 [==============================] - 8s 62ms/step - loss: 0.1528 -
accuracy: 0.9494
Epoch 87/200
126/126 [==============================] - 9s 68ms/step - loss: 0.1533 -
accuracy: 0.9471
Epoch 88/200
126/126 [==============================] - 7s 59ms/step - loss: 0.1527 -
accuracy: 0.9504
Epoch 89/200
126/126 [==============================] - 7s 57ms/step - loss: 0.1505 -
accuracy: 0.9504
Epoch 90/200
```

```
126/126 [==============================] - 7s 56ms/step - loss: 0.1497 -
accuracy: 0.9486
Epoch 91/200
126/126 [==============================] - 7s 56ms/step - loss: 0.1479 -
accuracy: 0.9491
Epoch 92/200
126/126 [==============================] - 7s 55ms/step - loss: 0.1471 -
accuracy: 0.9501
Epoch 93/200
126/126 [==============================] - 7s 57ms/step - loss: 0.1463 -
accuracy: 0.9501
Epoch 94/200
126/126 [==============================] - 7s 55ms/step - loss: 0.1446 -
accuracy: 0.9491
Epoch 95/200
126/126 [==============================] - 7s 56ms/step - loss: 0.1439 -
accuracy: 0.9496
Epoch 96/200
126/126 [==============================] - 7s 56ms/step - loss: 0.1431 -
accuracy: 0.9499
Epoch 97/200
126/126 [==============================] - 7s 56ms/step - loss: 0.1422 -
accuracy: 0.9494
Epoch 98/200
126/126 [==============================] - 7s 57ms/step - loss: 0.1416 -
accuracy: 0.9491
Epoch 99/200
126/126 [==============================] - 7s 57ms/step - loss: 0.1414 -
accuracy: 0.9489
Epoch 100/200
126/126 [==============================] - 7s 56ms/step - loss: 0.1401 -
accuracy: 0.9499
Epoch 101/200
126/126 [==============================] - 7s 56ms/step - loss: 0.1394 -
accuracy: 0.9491
Epoch 102/200
126/126 [==============================] - 7s 55ms/step - loss: 0.1390 -
accuracy: 0.9494
Epoch 103/200
126/126 [==============================] - 8s 65ms/step - loss: 0.1375 -
accuracy: 0.9496
Epoch 104/200
126/126 [==============================] - 7s 59ms/step - loss: 0.1382 -
accuracy: 0.9499
Epoch 105/200
126/126 [==============================] - 7s 57ms/step - loss: 0.1376 -
accuracy: 0.9476
Epoch 106/200
```

```
126/126 [==============================] - 7s 58ms/step - loss: 0.1368 -
accuracy: 0.9496
Epoch 107/200
126/126 [==============================] - 7s 58ms/step - loss: 0.1356 -
accuracy: 0.9489
Epoch 108/200
126/126 [==============================] - 8s 60ms/step - loss: 0.1355 -
accuracy: 0.9494
Epoch 109/200
126/126 [==============================] - 7s 56ms/step - loss: 0.1349 -
accuracy: 0.9496
Epoch 110/200
126/126 [==============================] - 7s 57ms/step - loss: 0.1341 -
accuracy: 0.9516
Epoch 111/200
126/126 [==============================] - 8s 64ms/step - loss: 0.1341 -
accuracy: 0.9504
Epoch 112/200
126/126 [==============================] - 8s 61ms/step - loss: 0.1340 -
accuracy: 0.9479
Epoch 113/200
126/126 [==============================] - 8s 61ms/step - loss: 0.1342 -
accuracy: 0.9504
Epoch 114/200
126/126 [==============================] - 7s 59ms/step - loss: 0.1335 -
accuracy: 0.9506
Epoch 115/200
126/126 [==============================] - 8s 60ms/step - loss: 0.1344 -
accuracy: 0.9494
Epoch 116/200
126/126 [==============================] - 8s 60ms/step - loss: 0.1616 -
accuracy: 0.9434
Epoch 117/200
126/126 [==============================] - 8s 63ms/step - loss: 0.1871 -
accuracy: 0.9382
Epoch 118/200
126/126 [==============================] - 7s 58ms/step - loss: 0.1494 -
accuracy: 0.9447
Epoch 119/200
126/126 [==============================] - 7s 58ms/step - loss: 0.1373 -
accuracy: 0.9489
Epoch 120/200
126/126 [==============================] - 8s 61ms/step - loss: 0.1334 -
accuracy: 0.9491
Epoch 121/200
126/126 [==============================] - 7s 56ms/step - loss: 0.1317 -
accuracy: 0.9514
Epoch 122/200
```

```
126/126 [==============================] - 7s 56ms/step - loss: 0.1315 -
accuracy: 0.9504
Epoch 123/200
126/126 [==============================] - 7s 56ms/step - loss: 0.1311 -
accuracy: 0.9489
Epoch 124/200
126/126 [==============================] - 7s 55ms/step - loss: 0.1316 -
accuracy: 0.9516
Epoch 125/200
126/126 [==============================] - 7s 56ms/step - loss: 0.1313 -
accuracy: 0.9501
Epoch 126/200
126/126 [==============================] - 7s 55ms/step - loss: 0.1309 -
accuracy: 0.9506
Epoch 127/200
126/126 [==============================] - 7s 55ms/step - loss: 0.1304 -
accuracy: 0.9504
Epoch 128/200
126/126 [==============================] - 7s 54ms/step - loss: 0.1300 -
accuracy: 0.9481
Epoch 129/200
126/126 [==============================] - 7s 55ms/step - loss: 0.1300 -
accuracy: 0.9476
Epoch 130/200
126/126 [==============================] - 7s 58ms/step - loss: 0.1301 -
accuracy: 0.9496
Epoch 131/200
126/126 [==============================] - 7s 56ms/step - loss: 0.1303 -
accuracy: 0.9501
Epoch 132/200
126/126 [==============================] - 8s 61ms/step - loss: 0.1300 -
accuracy: 0.9486
Epoch 133/200
126/126 [==============================] - 7s 55ms/step - loss: 0.1299 -
accuracy: 0.9489
Epoch 134/200
126/126 [==============================] - 7s 55ms/step - loss: 0.1293 -
accuracy: 0.9509
Epoch 135/200
126/126 [==============================] - 7s 53ms/step - loss: 0.1294 -
accuracy: 0.9476
Epoch 136/200
126/126 [==============================] - 7s 55ms/step - loss: 0.1290 -
accuracy: 0.9499
Epoch 137/200
126/126 [==============================] - 6s 51ms/step - loss: 0.1293 -
accuracy: 0.9514
Epoch 138/200
```

```
126/126 [==============================] - 6s 50ms/step - loss: 0.1284 -
accuracy: 0.9504
Epoch 139/200
126/126 [==============================] - 6s 49ms/step - loss: 0.1291 -
accuracy: 0.9491
Epoch 140/200
126/126 [==============================] - 7s 56ms/step - loss: 0.1288 -
accuracy: 0.9496
Epoch 141/200
126/126 [==============================] - 7s 53ms/step - loss: 0.1288 -
accuracy: 0.9481
Epoch 142/200
126/126 [==============================] - 6s 49ms/step - loss: 0.1289 -
accuracy: 0.9506
Epoch 143/200
126/126 [==============================] - 6s 49ms/step - loss: 0.1283 -
accuracy: 0.9501
Epoch 144/200
126/126 [==============================] - 6s 49ms/step - loss: 0.1279 -
accuracy: 0.9489
Epoch 145/200
126/126 [==============================] - 6s 50ms/step - loss: 0.1279 -
accuracy: 0.9519
Epoch 146/200
126/126 [==============================] - 6s 49ms/step - loss: 0.1274 -
accuracy: 0.9496
Epoch 147/200
126/126 [==============================] - 6s 49ms/step - loss: 0.1280 -
accuracy: 0.9489
Epoch 148/200
126/126 [==============================] - 6s 50ms/step - loss: 0.1274 -
accuracy: 0.9509
Epoch 149/200
126/126 [==============================] - 7s 55ms/step - loss: 0.1273 -
accuracy: 0.9479
Epoch 150/200
126/126 [==============================] - 7s 52ms/step - loss: 0.1268 -
accuracy: 0.9509
Epoch 151/200
126/126 [==============================] - 6s 49ms/step - loss: 0.1277 -
accuracy: 0.9496
Epoch 152/200
126/126 [==============================] - 6s 49ms/step - loss: 0.1270 -
accuracy: 0.9511
Epoch 153/200
126/126 [==============================] - 6s 49ms/step - loss: 0.1270 -
accuracy: 0.9504
Epoch 154/200
```

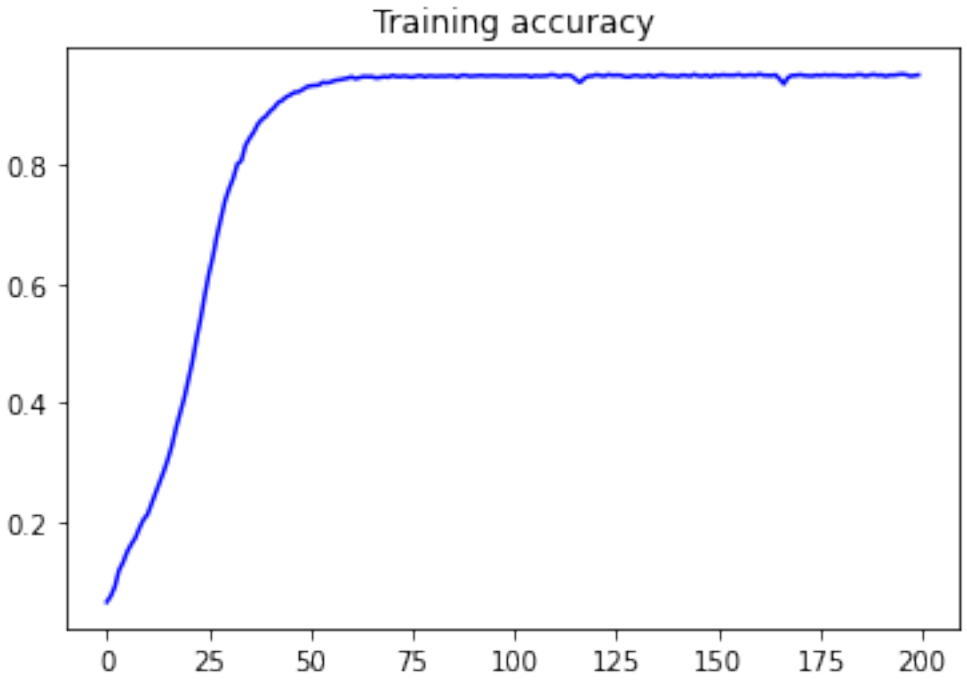
```
126/126 [==============================] - 6s 49ms/step - loss: 0.1270 -
accuracy: 0.9504
Epoch 155/200
126/126 [==============================] - 6s 49ms/step - loss: 0.1274 -
accuracy: 0.9504
Epoch 156/200
126/126 [==============================] - 6s 49ms/step - loss: 0.1269 -
accuracy: 0.9519
Epoch 157/200
126/126 [==============================] - 6s 49ms/step - loss: 0.1274 -
accuracy: 0.9494
Epoch 158/200
126/126 [==============================] - 6s 49ms/step - loss: 0.1269 -
accuracy: 0.9501
Epoch 159/200
126/126 [==============================] - 6s 48ms/step - loss: 0.1266 -
accuracy: 0.9511
Epoch 160/200
126/126 [==============================] - 6s 48ms/step - loss: 0.1263 -
accuracy: 0.9499
Epoch 161/200
126/126 [==============================] - 6s 48ms/step - loss: 0.1263 -
accuracy: 0.9524
Epoch 162/200
126/126 [==============================] - 6s 48ms/step - loss: 0.1264 -
accuracy: 0.9504
Epoch 163/200
126/126 [==============================] - 6s 48ms/step - loss: 0.1267 -
accuracy: 0.9501
Epoch 164/200
126/126 [==============================] - 6s 48ms/step - loss: 0.1263 -
accuracy: 0.9499
Epoch 165/200
126/126 [==============================] - 6s 49ms/step - loss: 0.1263 -
accuracy: 0.9511
Epoch 166/200
126/126 [==============================] - 6s 49ms/step - loss: 0.1459 -
accuracy: 0.9439
Epoch 167/200
126/126 [==============================] - 7s 53ms/step - loss: 0.1817 -
accuracy: 0.9357
Epoch 168/200
126/126 [==============================] - 6s 49ms/step - loss: 0.1415 -
accuracy: 0.9464
Epoch 169/200
126/126 [==============================] - 7s 54ms/step - loss: 0.1293 -
accuracy: 0.9501
Epoch 170/200
```

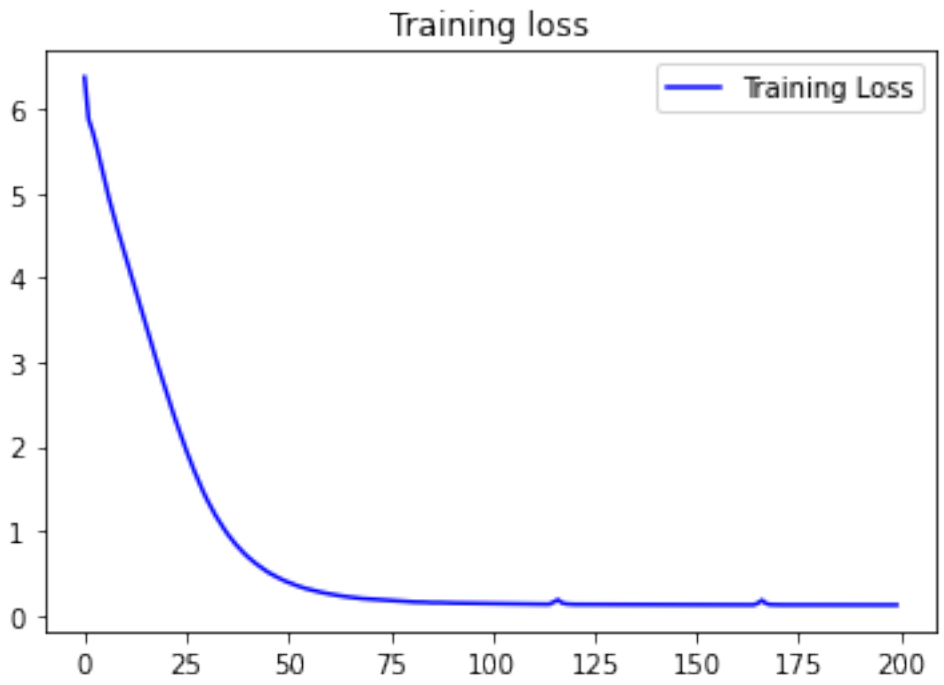
```
126/126 [==============================] - 7s 52ms/step - loss: 0.1275 -
accuracy: 0.9501
Epoch 171/200
126/126 [==============================] - 6s 48ms/step - loss: 0.1270 -
accuracy: 0.9514
Epoch 172/200
126/126 [==============================] - 6s 48ms/step - loss: 0.1263 -
accuracy: 0.9499
Epoch 173/200
126/126 [==============================] - 6s 51ms/step - loss: 0.1263 -
accuracy: 0.9496
Epoch 174/200
126/126 [==============================] - 7s 54ms/step - loss: 0.1266 -
accuracy: 0.9491
Epoch 175/200
126/126 [==============================] - 6s 49ms/step - loss: 0.1259 -
accuracy: 0.9506
Epoch 176/200
126/126 [==============================] - 6s 48ms/step - loss: 0.1261 -
accuracy: 0.9499
Epoch 177/200
126/126 [==============================] - 6s 48ms/step - loss: 0.1255 -
accuracy: 0.9511
Epoch 178/200
126/126 [==============================] - 6s 48ms/step - loss: 0.1263 -
accuracy: 0.9501
Epoch 179/200
126/126 [==============================] - 6s 50ms/step - loss: 0.1257 -
accuracy: 0.9509
Epoch 180/200
126/126 [==============================] - 6s 48ms/step - loss: 0.1253 -
accuracy: 0.9506
Epoch 181/200
126/126 [==============================] - 6s 50ms/step - loss: 0.1258 -
accuracy: 0.9501
Epoch 182/200
126/126 [==============================] - 7s 54ms/step - loss: 0.1262 -
accuracy: 0.9489
Epoch 183/200
126/126 [==============================] - 7s 53ms/step - loss: 0.1258 -
accuracy: 0.9504
Epoch 184/200
126/126 [==============================] - 8s 61ms/step - loss: 0.1251 -
accuracy: 0.9494
Epoch 185/200
126/126 [==============================] - 8s 59ms/step - loss: 0.1255 -
accuracy: 0.9514
Epoch 186/200
```

```
126/126 [==============================] - 7s 54ms/step - loss: 0.1254 -
accuracy: 0.9514
Epoch 187/200
126/126 [==============================] - 7s 55ms/step - loss: 0.1256 -
accuracy: 0.9491
Epoch 188/200
126/126 [==============================] - 8s 60ms/step - loss: 0.1249 -
accuracy: 0.9506
Epoch 189/200
126/126 [==============================] - 7s 54ms/step - loss: 0.1257 -
accuracy: 0.9519
Epoch 190/200
126/126 [==============================] - 7s 54ms/step - loss: 0.1254 -
accuracy: 0.9501
Epoch 191/200
126/126 [==============================] - 6s 51ms/step - loss: 0.1253 -
accuracy: 0.9509
Epoch 192/200
126/126 [==============================] - 6s 50ms/step - loss: 0.1249 -
accuracy: 0.9486
Epoch 193/200
126/126 [==============================] - 6s 49ms/step - loss: 0.1254 -
accuracy: 0.9506
Epoch 194/200
126/126 [==============================] - 6s 50ms/step - loss: 0.1254 -
accuracy: 0.9504
Epoch 195/200
126/126 [==============================] - 6s 48ms/step - loss: 0.1248 -
accuracy: 0.9514
Epoch 196/200
126/126 [==============================] - 6s 47ms/step - loss: 0.1247 -
accuracy: 0.9524
Epoch 197/200
126/126 [==============================] - 6s 48ms/step - loss: 0.1253 -
accuracy: 0.9514
Epoch 198/200
126/126 [==============================] - 6s 49ms/step - loss: 0.1259 -
accuracy: 0.9489
Epoch 199/200
126/126 [==============================] - 6s 47ms/step - loss: 0.1256 -
accuracy: 0.9501
Epoch 200/200
126/126 [==============================] - 6s 48ms/step - loss: 0.1250 -
accuracy: 0.9509
```

Let's plot details of our training.

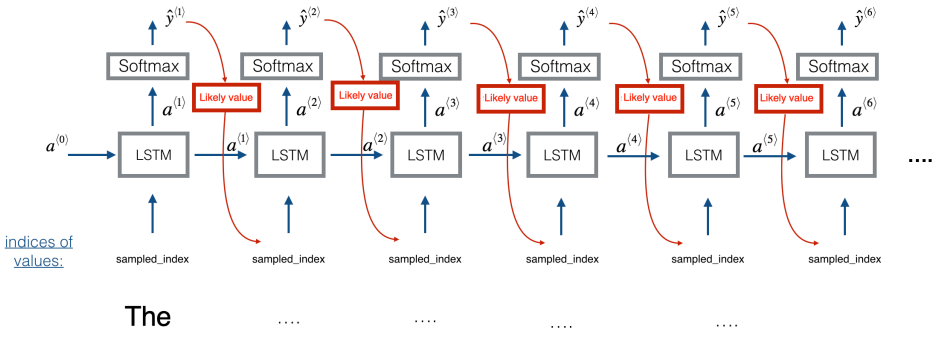
[14]:
```
acc = history.history['accuracy']
loss = history.history['loss']
epochs = range(len(acc))
plt.plot(epochs, acc, 'b', label='Training accuracy')
plt.title('Training accuracy')
plt.figure()
plt.plot(epochs, loss, 'b', label='Training Loss')
plt.title('Training loss')
plt.legend()
plt.show()
```





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()`.

[15]:
```
tokenizer.word_index.items()
```

[15]:
```
dict_items([('.', 1), ('the', 2), ('covid19', 3), ('in', 4), ('to', 5), ('a',
6), ('of', 7), (',', 8), ('coronavirus', 9), ('and', 10), ('says', 11), ('is',
12), ('for', 13), ('can', 14), ('from', 15), ('that', 16), ('are', 17),
('people', 18), ('with', 19), ('you', 20), ('virus', 21), ('on', 22), ('it',
23), ('or', 24), ('will', 25), ('was', 26), ('your', 27), ('has', 28), ('be',
29), ('kill', 30), ('!', 31), ('at', 32), ('not', 33), ('as', 34), ('have', 35),
('all', 36), ('this', 37), ('china', 38), ('trump', 39), ('by', 40), ('prevent',
41), ('new', 42), ('drinking', 43), ('they', 44), ('flu', 45), ('get', 46),
('out', 47), ('there', 48), ('vaccine', 49), ('days', 50), ('spread', 51),
('water', 52), ('lab', 53), ('shows', 54), ('pandemic', 55), ('so', 56),
('home', 57), ('against', 58), ('outbreak', 59), ('president', 60), ('donald',
61), ('if', 62), ('because', 63), ('only', 64), ('us', 65), ('been', 66), ('do',
67), ('being', 68), ('more', 69), ('protect', 70), ('one', 71), ('wuhan', 72),
('them', 73), ('chinese', 74), ('an', 75), ('had', 76), ('when', 77), ('up',
78), ('michigan', 79), ('now', 80), ('through', 81), ('hot', 82), ('may', 83),
('novel', 84), ('patients', 85), ('into', 86), ('two', 87), ('their', 88),
('bill', 89), ('government', 90), ('photo', 91), ('but', 92), ('about', 93),
('died', 94), ('obama', 95), ('death', 96), ('before', 97), ('corona', 98),
('door', 99), ('help', 100), ('cure', 101), ('than', 102), ('disease', 103),
('alcohol', 104), ('cold', 105), ('symptoms', 106), ('just', 107), ('kills',
108), ('lemon', 109), ('rate', 110), ('actually', 111), ('developed', 112),
('down', 113), ('2019', 114), ('cdc', 115), ('military', 116), ('risk', 117),
('gates', 118), ('years', 119), ('order', 120), ('deaths', 121), ('year', 122),
('cases', 123), ('000', 124), ('world', 125), ('her', 126), ('going', 127),
('after', 128), ('video', 129), ('swine', 130), ('states', 131), ('test', 132),
('treat', 133), ('hands', 134), ('source', 135), ('common', 136), ('bleach',
137), ('man', 138), ('transmitted', 139), ('5g', 140), ('sun', 141), ('means',
142), ('10', 143), ('without', 144), ('taking', 145), ('body', 146),
('infection', 147), ('scientists', 148), ('no', 149), ('100', 150), ('stomach',
151), ('immune', 152), ('vitamin', 153), ('weapon', 154), ('real', 155),
('created', 156), ('person', 157), ('could', 158), ('first', 159), ('everyone',
160), ('stay', 161), ('would', 162), ('country', 163), ('city', 164), ('why',
165), ('?', 166), ("isn't", 167), ('where', 168), ('viruses', 169), ('election',
170), ('church', 171), ('he', 172), ('york', 173), ('democrats', 174), ('money',
175), ('reports', 176), ('million', 177), ('1', 178), ('health', 179),
('testing', 180), ("it's", 181), ('like', 182), ('quarantine', 183), ('2020',
184), ('march', 185), ('around', 186), ('same', 187), ('sars', 188), ('eating',
189), ('sunlight', 190), ('made', 191), ("can't", 192), ('who', 193), ('die',
194), ('yourself', 195), ('life', 196), ('any', 197), ('other', 198), ('over',
199), ('getting', 200), ('breaking', 201), ('chloroquine', 202), ('hours', 203),
('patented', 204), ('then', 205), ('high', 206), ('c', 207), ('recommends',
208), ('party', 209), ('its', 210), ('biological', 211), ('programs', 212),
('bioweapon', 213), ('gargling', 214), ('humans', 215), ('buying', 216),
('make', 217), ('caused', 218), ('planned', 219), ('which', 220), ('lot', 221),
```

('warm', 222), ('should', 223), ('wear', 224), ('wearing', 225), ('rob', 226), ('street', 227), ('basis', 228), ('anthony', 229), ('âę', 230), ('hydroxychloroquine', 231), ('joe', 232), ('travel', 233), ('were', 234), ('news', 235), ('masks', 236), ('announced', 237), ('2015', 238), ('said', 239), ('governors', 240), ('we', 241), ('saying', 242), ('folks', 243), ('what', 244), ('center', 245), ('stimulus', 246), ('back', 247), ('per', 248), ('nancy', 249), ('go', 250), ('covid', 251), ('next', 252), ("you're", 253), ('hospital', 254), ("we're", 255), ('italy', 256), ('due', 257), ('predicted', 258), ('positive', 259), ('confirmed', 260), ('hospitals', 261), ("that's", 262), ('sanitizer', 263), ('turned', 264), ('caught', 265), ('very', 266), ('avoid', 267), ('17', 268), ('right', 269), ('citizens', 270), ('2', 271), ('week', 272), ('starting', 273), ('united', 274), ('â', 275), ('israel', 276), ('florida', 277), ('already', 278), ('surfaces', 279), ('7', 280), ('animal', 281), ('higher', 282), ('cannot', 283), ('able', 284), ('breath', 285), ('coughing', 286), ('vaccines', 287), ('helps', 288), ('young', 289), ('effective', 290), ('blood', 291), ('patent', 292), ('sudden', 293), ('solution', 294), ('shocked', 295), ('discover', 296), ('super', 297), ('under', 298), ('fish', 299), ('tank', 300), ('additive', 301), ('slices', 302), ('cup', 303), ('save', 304), ('proliferation', 305), ('huge', 306), ('results', 307), ('study', 308), ('show', 309), ('infected', 310), ('wuxi', 311), ('pharma', 312), ('located', 313), ('gets', 314), ('mouth', 315), ('liquids', 316), ('wash', 317), ('esophagus', 318), ('once', 319), ('tummy', 320), ('acid', 321), ('exposed', 322), ('invented', 323), ('usa', 324), ('planted', 325), ('color', 326), ('melanin', 327), ('slowed', 328), ('stopped', 329), ('immediate', 330), ('widespread', 331), ('use', 332), ('doses', 333), ('men', 334), ('shave', 335), ('beards', 336), ('bio', 337), ("china's", 338), ('army', 339), ('engineered', 340), ('communist', 341), ('finally', 342), ('admit', 343), ("wuhan'", 344), ('linked', 345), ('covert', 346), ('stole', 347), ('canada', 348), ('weaponized', 349), ('dogs', 350), ('form', 351), ('catch', 352), ('known', 353), ('originated', 354), ('soup', 355), ('products', 356), ('mosquitoes', 357), ('causes', 358), ('research', 359), ('ultraviolet', 360), ('mass', 361), ('killing', 362), ('stop', 363), ('vinegar', 364), ('thing', 365), ('infects', 366), ('such', 367), ('seeds', 368), ('salt', 369), ('eliminates', 370), ('petrol', 371), ("one's", 372), ('helicopters', 373), ('spray', 374), ('disinfectant', 375), ('homes', 376), ('off', 377), ('spreading', 378), ('treatments', 379), ('protest', 380), ("we've", 381), ('scientific', 382), ('dr', 383), ('fauci', 384), ('15', 385), ('current', 386), ('case', 387), ('uk', 388), ('i', 389), ('time', 390), ('things', 391), ('written', 392), ('calling', 393), ("trump's", 394), ('did', 395), ('2017', 396), ('3', 397), ('2014', 398), ('white', 399), ('house', 400), ('dollars', 401), ('used', 402), ('while', 403), ('times', 404), ('plague', 405), ('shut', 406), ('governor', 407), ('gretchen', 408), ('whitmer', 409), ('vacation', 410), ('foundation', 411), ('tested', 412), ('children', 413), ('vaccinated', 414), ('much', 415), ('here', 416), ('stores', 417), ('kennedy', 418), ('labeled', 419), ("wasn't", 420), ('until', 421), ('4', 422), ('come', 423), ('showing', 424), ('beach', 425), ('jacksonville', 426), ('fla', 427), ('claimed', 428), ('exact', 429), ('pelosi', 430), ('organization', 431), ('200', 432), ('checks', 433),

('advance', 434), ('tax', 435), ('bans', 436), ('sale', 437), ('send', 438), ('nurses', 439), ('few', 440), ('got', 441), ('bad', 442), ('countries', 443), ('weeks', 444), ('during', 445), ('lockdown', 446), ('dying', 447), ('decision', 448), ('open', 449), ('stock', 450), ('vaccination', 451), ('sick', 452), ('orders', 453), ('exactly', 454), ('every', 455), ('message', 456), ('enough', 457), ('drink', 458), ('fire', 459), ('employees', 460), ('instead', 461), ('full', 462), ('tests', 463), ('fake', 464), ('lady', 465), ('banning', 466), ('air', 467), ('state', 468), ('fever', 469), ('leave', 470), ('these', 471), ('mers', 472), ('closed', 473), ('census', 474), ('n', 475), ('y', 476), ('gov', 477), ('16', 478), ('ventilators', 479), ('buy', 480), ('preparing', 481), ('mandatory', 482), ('came', 483), ('way', 484), ('tom', 485), ('hanks', 486), ('hoax', 487), ('infections', 488), ('months', 489), ('olympics', 490), ('barack', 491), ('dead', 492), ('2008', 493), ('syndrome', 494), ('possible', 495), ('administration', 496), ('also', 497), ('medical', 498), ('umbrella', 499), ('corporation', 500), ('mask', 501), ('cleaning', 502), ('survives', 503), ('remedies', 504), ('terrorism', 505), ('heat', 506), ('hand', 507), ('dryers', 508), ('mail', 509), ('kids', 510), ('mobile', 511), ('networks', 512), ('exposing', 513), ('temperatures', 514), ('25c', 515), ('degrees', 516), ('recover', 517), ('catching', 518), ('hold', 519), ('seconds', 520), ('feeling', 521), ('discomfort', 522), ('free', 523), ('lung', 524), ('dangerous', 525), ('areas', 526), ('humid', 527), ('climates', 528), ('weather', 529), ('snow', 530), ('bath', 531), ('prevents', 532), ('spraying', 533), ('chlorine', 534), ('human', 535), ('pneumonia', 536), ('regularly', 537), ('rinsing', 538), ('nose', 539), ('saline', 540), ('protected', 541), ('garlic', 542), ('antibiotics', 543), ('preventing', 544), ('treating', 545), ('skin', 546), ('irritation', 547), ('expired', 548), ('despite', 549), ('media', 550), ('fear', 551), ('mongering', 552), ('available', 553), ('weed', 554), ('cocaine', 555), ('drug', 556), ('fight', 557), ("doesn't", 558), ('affect', 559), ('muslims', 560), ('hair', 561), ('weave', 562), ('lace', 563), ('fronts', 564), ('manufactured', 565), ('contain', 566), ('spawned', 567), ('code', 568), ('name', 569), ('zyphr', 570), ('damn', 571), ("'a", 572), ('older', 573), ('adults', 574), ('cats', 575), ('hiv', 576), ('prone', 577), ('mutated', 578), ('someone', 579), ('minutes', 580), ('thermal', 581), ('scanners', 582), ('diagnose', 583), ('parcels', 584), ('food', 585), ('deadliest', 586), ('laboratory', 587), ('began', 588), ('ate', 589), ('bat', 590), ('reduces', 591), ('deliberately', 592), ('released', 593), ('ordering', 594), ('shipped', 595), ('overseas', 596), ('pregnant', 597), ('women', 598), ('ticks', 599), ('eyes', 600), ('extra', 601), ('amounts', 602), ('accidental', 603), ('leak', 604), ('institute', 605), ('disinfection', 606), ('lamp', 607), ('responsible', 608), ('epidemic', 609), ('affects', 610), ('asians', 611), ('some', 612), ('kind', 613), ('ethnic', 614), ('companies', 615), ('hoping', 616), ('profit', 617), ('ago', 618), ('officials', 619), ('seeking', 620), ('approval', 621), ('start', 622), ('20000', 623), ('covering', 624), ('sesame', 625), ('oil', 626), ('mixture', 627), ('medicine', 628), ('banlangen', 629), ('prevented', 630), ('miracle', 631), ('mineral', 632), ('essentially', 633), ('black', 634), ('tea', 635), ('morning', 636), ('pepper', 637), ('lime', 638), ('flushes', 639), ('steam', 640), ('face', 641), ('inhale', 642), ('neem', 643), ('having',

644), ('malaria', 645), ('makes', 646), ('individuals', 647), ('living', 648),
('chronic', 649), ('conditions', 650), ('asthma', 651), ('diabetes', 652),
('consuming', 653), ('vaping', 654), ('containing', 655), ('cannabidiol', 656),
('cbd', 657), ('boost', 658), ('system', 659), ('acacia', 660), ('spreads',
661), ('pumps', 662), ('gloves', 663), ('filling', 664), ('cars', 665),
('shoes', 666), ('reason', 667), ('behind', 668), ('ibuprofen', 669),
('worsens', 670), ('particular', 671), ('d', 672), ('pills', 673), ('socks',
674), ('mustard', 675), ('patches', 676), ('well', 677), ('goose', 678), ('fat',
679), ('chest', 680), ('cantrall', 681), ('recent', 682), ('bailed', 683),
('wall', 684), ('main', 685), ('models', 686), ('projecting', 687), ('talking',
688), ('mitigation', 689), ('still', 690), ('believing', 691), ('social', 692),
('distancing', 693), ('necessary', 694), ('since', 695), ('never', 696),
('studied', 697), ('future', 698), ('volunteer', 699), ('trial', 700), ('see',
701), ('history', 702), ('perfectly', 703), ('healthy', 704), ('basically',
705), ('confined', 706), ('essential', 707), ('kansas', 708), ('st', 709),
('louis', 710), ('biden', 711), ('letter', 712), ('apology', 713),
('restrictions', 714), ('xenophobic', 715), ('nih', 716), ('give', 717), ('7m',
718), ('grants', 719), ('prohibited', 720), ('pres', 721), ('grant', 722),
('exception', 723), ('selling', 724), ('themed', 725), ('commemorative', 726),
('coins', 727), ('gift', 728), ('shop', 729), ('walmart', 730), ('amazon', 731),
('kroger', 732), ('target', 733), ('costco', 734), ('reported', 735),
('existence', 736), ('canine', 737), ('casts', 738), ('doubt', 739),
('statements', 740), ("âbiological'", 741), ("âfunded'", 742), ('barak',
743), ('sp', 744), ('hussein', 745), ('tune', 746), ('800', 747), ('american',
748), ('radiation', 749), ('administered', 750), ('bacteria', 751), ('transfer',
752), ('april', 753), ('22', 754), ('jump', 755), ('related', 756), ('1986',
757), ('founder', 758), ('square', 759), ('prophesied', 760), ('saw', 761),
('coming', 762), ('bars', 763), ('especially', 764), ('hard', 765), ('hit',
766), ("âhundreds'", 767), ('himâę', 768), ('50', 769), ('spent', 770),
('past', 771), ('weekend', 772), ('cottage', 773), ('birch', 774), ('lake',
775), ('violating', 776), ('own', 777), ('executive', 778), ('banner', 779),
('swastika', 780), ('pence', 781), ('polio', 782), ('vax', 783), ('india', 784),
('between', 785), ('2000', 786), ('paralysed', 787), ('496', 788), ('pushing',
789), ('implanted', 790), ('microchip', 791), ('trey', 792), ('gowdy', 793),
("i'm", 794), ('realâę', 795), ('pay', 796), ('attention', 797), ("there's",
798), ('meets', 799), ('eye', 800), ('nigerians', 801), ('burning', 802),
('pbs', 803), ('donated', 804), ("diffie's", 805), ('herd', 806), ('immunity',
807), ('probably', 808), ('california', 809), ('far', 810), ('fewer', 811),
('refuse', 812), ('sign', 813), ('small', 814), ('businesses', 815), ('taken',
816), ('journalist', 817), ('governmentreopened', 818), ("wisconsin's", 819),
('surge', 820), ('different', 821), ('beaches', 822), ('los', 823), ('angeles',
824), ('county', 825), ('360', 826), ('swimming', 827), ('pools', 828),
("don't", 829), ('govwhitmer', 830), ('banned', 831), ('purchasing', 832),
('baby', 833), ('car', 834), ('seat', 835), ('speaker', 836), ('deleted', 837),
('telling', 838), ('chinatown', 839), ('ask', 840), ("amazon's", 841), ('alexa',
842), ('origin', 843), ('stands', 844), ('lose', 845), ("âvaccine'", 846),
('invested', 847), ('think', 848), ('charged', 849), ('facts', 850), ('figures',

851), ('top', 852), ('relief', 853), ('return', 854), ('automatically', 855), ('owe', 856), ('season', 857), ('gardening', 858), ('vegetable', 859), ('fruit', 860), ('44', 861), ('senators', 862), ('voted', 863), ('italian', 864), ('guy', 865), ('twice', 866), ('line', 867), ('done', 868), ('capita', 869), ('bottom', 870), ('list', 871), ('gay', 872), ('asian', 873), ('hong', 874), ('kong', 875), ('bird', 876), ('flus', 877), ('each', 878), ('killed', 879), ('flags', 880), ('services', 881), ('resume', 882), ('ceos', 883), ('notice', 884), ('resigned', 885), ('dump', 886), ('bags', 887), ('dumped', 888), ('ditch', 889), ('situation', 890), ('senegal', 891), ('started', 892), ('yesterday', 893), ('8', 894), ('received', 895), ('spot', 896), ('suggests', 897), ('urged', 898), ('vote', 899), ('south', 900), ('carolina', 901), ('occurs', 902), ('rudy', 903), ('giuliani', 904), ('bought', 905), ('2m', 906), ('shares', 907), ('novartis', 908), ('primary', 909), ('supplier', 910), ('early', 911), ('february', 912), ('handing', 913), ('doused', 914), ('chemicals', 915), ('knocks', 916), ('thrown', 917), ('streets', 918), ('clear', 919), ('whole', 920), ('danger', 921), ('poem', 922), ('staying', 923), ('kathleen', 924), ("o'mara", 925), ('1869', 926), ('59', 927), ('pastor', 928), ('gives', 929), ('dettol', 930), ('destroying', 931), ('poles', 932), ('aware', 933), ('triggering', 934), ('goodwill', 935), ('laying', 936), ('rivers', 937), ('oklahoma', 938), ('catfish', 939), ('carrying', 940), ('nostradamus', 941), ('1551', 942), ('passage', 943), ('east', 944), ('exotic', 945), ('prison', 946), ('pictures', 947), ('empty', 948), ('prove', 949), ('crisis', 950), ('agendas', 951), ("pelosi's", 952), ('daughter', 953), ('board', 954), ('important', 955), ('chicago', 956), ('lowered', 957), ('applied', 958), ('forearms', 959), ('went', 960), ('kitchen', 961), ('cook', 962), ('moment', 963), ('she', 964), ('gas', 965), ('stove', 966), ('contained', 967), ('slow', 968), ('quick', 969), ('ban', 970), ('critical', 971), ('saving', 972), ('lives', 973), ('nevada', 974), ('issued', 975), ('prescription', 976), ('queen', 977), ('elizabeth', 978), ('remains', 979), ('eight', 980), ('required', 981), ('everywhere', 982), ("they're", 983), ('furloughing', 984), ('western', 985), ('certain', 986), ('imposters', 987), ('hazmat', 988), ('suits', 989), ('stockton', 990), ('calif', 991), ('checking', 992), ('residents', 993), ('enter', 994), ('physically', 995), ('attempt', 996), ('robbery', 997), ('objects', 998), ('contracting', 999), ('survive', 1000), ('blame', 1001), ('culture', 1002), ('eat', 1003), ('bats', 1004), ('snakes', 1005), ('receive', 1006), ('three', 1007), ('hotels', 1008), ('four', 1009), ('republican', 1010), ('how', 1011), ('hell', 1012), ('audio', 1013), ('lists', 1014), ('five', 1015), ('ways', 1016), ('humvee', 1017), ('team', 1018), ('interstate', 1019), ('696', 1020), ('las', 1021), ('vegas', 1022), ('knock', 1023), ('nv', 1024), ('power', 1025), ('cvd', 1026), ('19', 1027), ('testers', 1028), ('robbing', 1029), ('gunpoint', 1030), ('andrew', 1031), ('cuomo', 1032), ('rejected', 1033), ('recommended', 1034), ('established', 1035), ('panels', 1036), ('lotteries', 1037), ('chance', 1038), ('low', 1039), ('price', 1040), ('putin', 1041), ('stated', 1042), ('russian', 1043), ('options', 1044), ('jail', 1045), ('5', 1046), ('special', 1047), ('pesticide', 1048), ('skies', 1049), ('2020census', 1050), ('fill', 1051), ('check', 1052), ('federal', 1053), ('mobilize', 1054), ('national', 1055), ('guard', 1056), ('dispatch', 1057),

('across', 1058), ('announce', 1059), ('nationwide', 1060), ('colorado', 1061),
('springs', 1062), ('stating', 1063), ('george', 1064), ('soros', 1065),
('owns', 1066), ('conveniently', 1067), ('broke', 1068), ('entitled', 1069),
('700', 1070), ('usd', 1071), ("fauci's", 1072), ('statement', 1073),
('seasonal', 1074), ('claim', 1075), ('msnbc', 1076), ('âi', 1077), ('hope',
1078), ('harms', 1079), ('re', 1080), ('evoke', 1081), ('sic', 1082), ('called',
1083), ('stafford', 1084), ('act', 1085), ('nation', 1086), ('announcing',
1087), ('tomorrow', 1088), ('14', 1089), ('detain', 1090), ('anyone', 1091),
('indefinitely', 1092), ('detention', 1093), ("simpsons'", 1094), ('2007',
1095), ('trying', 1096), ('include', 1097), ('abortion', 1098), ('funding',
1099), ('combat', 1100), ('monitor', 1101), ('u', 1102), ('s', 1103), ('96',
1104), ('nearly', 1105), ('half', 1106), ('leaked', 1107), ('documents', 1108),
('reveal', 1109), ('safest', 1110), ('fly', 1111), ('donate', 1112), ("akira'",
1113), ('1988', 1114), ('movie', 1115), ('apocalyptic', 1116), ('event', 1117),
('place', 1118), ('tokyo', 1119), ('advising', 1120), ('japan', 1121),
('postpone', 1122), ('set', 1123), ('anti', 1124), ('47', 1125), ('vulnerable',
1126), ('something', 1127), ('know', 1128), ('37', 1129), ('volleyball', 1130),
('keep', 1131), ('him', 1132), ('company', 1133), ("he's", 1134),
('quarantined', 1135), ('insurance', 1136), ('industry', 1137), ('agreed',
1138), ('waive', 1139), ('co', 1140), ('payments', 1141), ('nyc', 1142),
('drops', 1143), ('middle', 1144), ('suspected', 1145), ('2004', 1146),
('avian', 1147), ('2010', 1148), ('2012', 1149), ('ebola', 1150), ('2018',
1151), ('zika', 1152), ('2016', 1153), ('simply', 1154), ('fabricated', 1155),
('cover', 1156), ('global', 1157), ('tweeted', 1158), ('market', 1159),
('economy', 1160), ('even', 1161), ('stronger', 1162), ('besides', 1163),
('best', 1164), ('inventions', 1165), ('judaism', 1166), ('olives', 1167),
('wonderful', 1168), ('bombed', 1169), ('peace', 1170), ("didn't", 1171),
('anything', 1172), ('detrimental', 1173), ('doing', 1174), ('outside', 1175),
("won't", 1176), ('allowed', 1177), ('measures', 1178), ('workplaces', 1179),
('paid', 1180), ('6', 1181), ('schools', 1182), ('close', 1183), ('6th', 1184),
('vatican', 1185), ('pope', 1186), ('francis', 1187), ('negative', 1188),
('second', 1189), ('visited', 1190), ('albany', 1191), ('waited', 1192), ('six',
1193), ('call', 1194), ('emergency', 1195), ('thousands', 1196), ('valley',
1197), ('regional', 1198), ('brownsville', 1199), ('texas', 1200), ('looked',
1201), ('shands', 1202), ('gainesville', 1203), ('mississippi', 1204),
('nobody', 1205), ('airborne', 1206), ('wind', 1207), ('blows', 1208),
('direction', 1209), ("you'll", 1210), ('issue', 1211), ('saint', 1212),
('laurent', 1213), ('hazardous', 1214), ('materials', 1215), ('suit', 1216),
('âresident', 1217), ("evil'", 1218), ('releases', 1219), ('raccoon', 1220),
('logo', 1221), ('found', 1222), ('nothing', 1223), ('america', 1224),
('vaccinating', 1225), ('cattle', 1226), ('yet', 1227), ('tells', 1228),
('gunna', 1229), ("âinfestation'", 1230), ('multiple', 1231), ('poll', 1232),
('finds', 1233), ('38', 1234), ('americans', 1235), ('say', 1236), ('beer',
1237), ('mortality', 1238), ('cremated', 1239), ('alive', 1240), ('book', 1241),
('end', 1242), ('prepares', 1243), ('hundreds', 1244), ('updates', 1245),
('using', 1246), ('supposed', 1247), ('useâę', 1248), ('side', 1249),
('filter', 1250), ('part', 1251), ('lysol', 1252), ('listed', 1253), ('clorox',

```
1254), ('bottle', 1255), ('âa', 1256)])
```

[24]:
```python
#TASK 2
# Implement the generate() function
def generate(seed_text):
    ### START CODE HERE ###
    count = []
    text = seed_text
    while True:
        token_list = tokenizer.texts_to_sequences([text])
        print(token_list)
        token_list = pad_sequences(token_list, maxlen=60, padding='pre')
        print(token_list)
        predicted = np.argmax(model.predict(token_list),axis=-1)
        output_word = ""
        for word, index in tokenizer.word_index.items():
            if index == predicted:
                output_word = word
                count.append(word)
                break
        text += " " + output_word
        if len(count) > 1:
            if count[-1] == word:
                text += "."
                break
        if output_word == '.' or output_word == '?' or output_word == '!':
            break
        if len(count) == 250:
            text += "."
            break
    return text
    ### END CODE HERE ###
```

Let's test it:

[25]:
```python
#print(generate("COVID19 virus"))
print(generate("COVID19 is the"))
#print(generate("The usa is"))
#print(generate("The new virus"))
#print(generate("China has"))
```

```
[[3, 12, 2]]
[[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   0  0  0  0  0  0  0  0  0  3 12  2]]
1/1 [==============================] - 0s 351ms/step
[[3, 12, 2, 586]]
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

```
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   3  12   2 586]]
1/1 [==============================] - 0s 24ms/step
COVID19 is the deadliest virus.
```

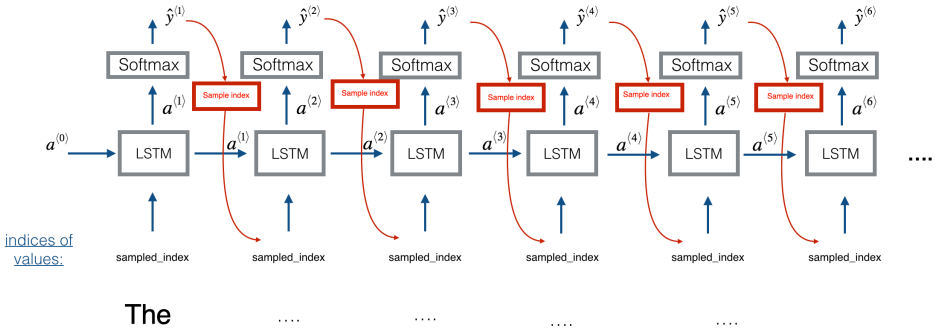
Let's test it in an interactive mode:

[26]:
```
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: Covid is
[[251, 12]]
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0 251  12]]
1/1 [==============================] - 0s 21ms/step
[[251, 12, 2]]
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0 251  12   2]]
1/1 [==============================] - 0s 23ms/step
```

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.

[27]:
```
#TASK 3
# Implement the generate_sample() function
def generate_sample(seed_text):
    ### START CODE HERE ###
    count = []
    text = seed_text
    while True:
        token_list = tokenizer.texts_to_sequences([text])
        token_list = pad_sequences(token_list, maxlen=60, padding='pre')
        predicted = random.choice(model.predict(token_list))
        ra = len(predicted)
        predic = random.randint(1,ra)
        print(predic)
        output_word = ""
        for word, index in tokenizer.word_index.items():
            if index == predic:
                output_word = word
                count.append(word)
                break
        text += " " + output_word
        if output_word == '.' or output_word == '?' or output_word == '!':
            break
        if len(count) == 50:
            text += "."
            break
    return text
```

Let's test it in an interactive mode:

[28]:
```
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: Covid is
1/1 [==============================] - 0s 25ms/step
872
1/1 [==============================] - 0s 20ms/step
139
1/1 [==============================] - 0s 22ms/step
988
1/1 [==============================] - 0s 21ms/step
1201
1/1 [==============================] - 0s 23ms/step
41
1/1 [==============================] - 0s 23ms/step
191
```

```
1/1 [==============================] - 0s 23ms/step
948
1/1 [==============================] - 0s 22ms/step
153
1/1 [==============================] - 0s 23ms/step
1248
1/1 [==============================] - 0s 23ms/step
468
1/1 [==============================] - 0s 30ms/step
151
1/1 [==============================] - 0s 21ms/step
1021
1/1 [==============================] - 0s 23ms/step
362
1/1 [==============================] - 0s 23ms/step
686
1/1 [==============================] - 0s 20ms/step
654
1/1 [==============================] - 0s 20ms/step
906
1/1 [==============================] - 0s 23ms/step
430
1/1 [==============================] - 0s 21ms/step
875
1/1 [==============================] - 0s 22ms/step
867
1/1 [==============================] - 0s 23ms/step
341
1/1 [==============================] - 0s 26ms/step
1218
1/1 [==============================] - 0s 21ms/step
566
1/1 [==============================] - 0s 21ms/step
310
1/1 [==============================] - 0s 23ms/step
1232
1/1 [==============================] - 0s 32ms/step
1019
1/1 [==============================] - 0s 21ms/step
575
1/1 [==============================] - 0s 23ms/step
827
1/1 [==============================] - 0s 27ms/step
643
1/1 [==============================] - 0s 25ms/step
91
1/1 [==============================] - 0s 22ms/step
317
```

```
1/1 [==============================] - 0s 22ms/step
573
1/1 [==============================] - 0s 26ms/step
298
1/1 [==============================] - 0s 20ms/step
108
1/1 [==============================] - 0s 25ms/step
349
1/1 [==============================] - 0s 27ms/step
330
1/1 [==============================] - 0s 24ms/step
213
1/1 [==============================] - 0s 24ms/step
112
1/1 [==============================] - 0s 25ms/step
323
1/1 [==============================] - 0s 23ms/step
200
1/1 [==============================] - 0s 22ms/step
1183
1/1 [==============================] - 0s 22ms/step
624
1/1 [==============================] - 0s 22ms/step
142
1/1 [==============================] - 0s 22ms/step
733
1/1 [==============================] - 0s 28ms/step
629
1/1 [==============================] - 0s 24ms/step
734
1/1 [==============================] - 0s 26ms/step
14
1/1 [==============================] - 0s 25ms/step
297
1/1 [==============================] - 0s 25ms/step
147
1/1 [==============================] - 0s 25ms/step
1067
1/1 [==============================] - 0s 24ms/step
381
Covid is gay transmitted hazmat looked prevent made empty vitamin useâę state
stomach las killing models vaping 2m pelosi kong line communist evil' contain
infected poll interstate cats swimming neem photo wash older under kills
weaponized immediate bioweapon developed invented getting close covering means
```

4 Generate your own text

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

[31]:
```
def test_model():
    ### START CODE HERE ###
    model = Sequential(name="Test")
    model.add(Embedding(total_words, 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(lr=0.001),␣
 ↪metrics=['accuracy'])
    ### END CODE HERE ###
    return model

#Print details of the model.
model1 = test_model()
model1.summary()
```

```
Model: "Test"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_2 (Embedding)      (None, 60, 128)           182528

 lstm_2 (LSTM)                (None, 128)               131584

 dense_2 (Dense)              (None, 1426)              183954

=================================================================
Total params: 498,066
Trainable params: 498,066
Non-trainable params: 0
_________________________________________________________________
```

[32]:
```
data = open('test.txt').read().replace(".", " . ").replace(",", " , ").
 ↪replace("?", " ? ").replace("!", " ! ")
corpus = data.lower().split("\n")
tokenizer.fit_on_texts(corpus)
total_words = len(tokenizer.word_index) + 1
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)
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'))
input_to_model, label = input_sequences[:,:-1],input_sequences[:,-1]
label = ku.to_categorical(label, num_classes=total_words)

history = model1.fit(input_to_model, label, epochs=200, batch_size=32,␣
 ↪verbose=1)
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()
```

```
Epoch 1/200
16/16 [==============================] - 2s 48ms/step - loss: 7.2036 - accuracy:
0.0551
Epoch 2/200
16/16 [==============================] - 1s 47ms/step - loss: 6.0489 - accuracy:
0.0714
Epoch 3/200
16/16 [==============================] - 1s 46ms/step - loss: 5.2459 - accuracy:
0.0714
Epoch 4/200
16/16 [==============================] - 1s 46ms/step - loss: 5.1062 - accuracy:
0.0531
Epoch 5/200
16/16 [==============================] - 1s 46ms/step - loss: 5.0834 - accuracy:
0.0714
Epoch 6/200
16/16 [==============================] - 1s 46ms/step - loss: 5.0650 - accuracy:
0.0714
Epoch 7/200
16/16 [==============================] - 1s 48ms/step - loss: 5.0549 - accuracy:
0.0714
Epoch 8/200
16/16 [==============================] - 1s 48ms/step - loss: 5.0531 - accuracy:
0.0714
Epoch 9/200
16/16 [==============================] - 1s 46ms/step - loss: 5.0507 - accuracy:
0.0714
Epoch 10/200
16/16 [==============================] - 1s 46ms/step - loss: 5.0293 - accuracy:
```

```
0.0714
Epoch 11/200
16/16 [==============================] - 1s 46ms/step - loss: 5.0277 - accuracy:
0.0714
Epoch 12/200
16/16 [==============================] - 1s 46ms/step - loss: 5.0060 - accuracy:
0.0714
Epoch 13/200
16/16 [==============================] - 1s 46ms/step - loss: 4.9774 - accuracy:
0.0714
Epoch 14/200
16/16 [==============================] - 1s 46ms/step - loss: 4.9283 - accuracy:
0.0714
Epoch 15/200
16/16 [==============================] - 1s 45ms/step - loss: 4.8729 - accuracy:
0.0714
Epoch 16/200
16/16 [==============================] - 1s 46ms/step - loss: 4.8011 - accuracy:
0.0714
Epoch 17/200
16/16 [==============================] - 1s 46ms/step - loss: 4.7219 - accuracy:
0.0735
Epoch 18/200
16/16 [==============================] - 1s 46ms/step - loss: 4.6464 - accuracy:
0.0755
Epoch 19/200
16/16 [==============================] - 1s 47ms/step - loss: 4.5630 - accuracy:
0.0857
Epoch 20/200
16/16 [==============================] - 1s 47ms/step - loss: 4.4912 - accuracy:
0.1000
Epoch 21/200
16/16 [==============================] - 1s 46ms/step - loss: 4.4204 - accuracy:
0.1082
Epoch 22/200
16/16 [==============================] - 1s 47ms/step - loss: 4.3372 - accuracy:
0.1184
Epoch 23/200
16/16 [==============================] - 1s 51ms/step - loss: 4.2642 - accuracy:
0.1306
Epoch 24/200
16/16 [==============================] - 1s 53ms/step - loss: 4.1830 - accuracy:
0.1510
Epoch 25/200
16/16 [==============================] - 1s 52ms/step - loss: 4.1082 - accuracy:
0.1816
Epoch 26/200
16/16 [==============================] - 1s 48ms/step - loss: 4.0372 - accuracy:
```

```
0.1816
Epoch 27/200
16/16 [==============================] - 1s 51ms/step - loss: 3.9603 - accuracy:
0.1959
Epoch 28/200
16/16 [==============================] - 1s 51ms/step - loss: 3.8853 - accuracy:
0.2122
Epoch 29/200
16/16 [==============================] - 1s 45ms/step - loss: 3.8122 - accuracy:
0.2245
Epoch 30/200
16/16 [==============================] - 1s 48ms/step - loss: 3.7366 - accuracy:
0.2510
Epoch 31/200
16/16 [==============================] - 1s 45ms/step - loss: 3.6638 - accuracy:
0.2531
Epoch 32/200
16/16 [==============================] - 1s 45ms/step - loss: 3.5892 - accuracy:
0.2673
Epoch 33/200
16/16 [==============================] - 1s 45ms/step - loss: 3.5214 - accuracy:
0.2755
Epoch 34/200
16/16 [==============================] - 1s 45ms/step - loss: 3.4445 - accuracy:
0.2939
Epoch 35/200
16/16 [==============================] - 1s 45ms/step - loss: 3.3756 - accuracy:
0.2939
Epoch 36/200
16/16 [==============================] - 1s 45ms/step - loss: 3.3025 - accuracy:
0.3143
Epoch 37/200
16/16 [==============================] - 1s 45ms/step - loss: 3.2342 - accuracy:
0.3367
Epoch 38/200
16/16 [==============================] - 1s 45ms/step - loss: 3.1663 - accuracy:
0.3490
Epoch 39/200
16/16 [==============================] - 1s 45ms/step - loss: 3.0982 - accuracy:
0.3551
Epoch 40/200
16/16 [==============================] - 1s 45ms/step - loss: 3.0333 - accuracy:
0.3735
Epoch 41/200
16/16 [==============================] - 1s 45ms/step - loss: 2.9663 - accuracy:
0.3714
Epoch 42/200
16/16 [==============================] - 1s 46ms/step - loss: 2.9027 - accuracy:
```

```
0.3816
Epoch 43/200
16/16 [==============================] - 1s 46ms/step - loss: 2.8372 - accuracy:
0.4061
Epoch 44/200
16/16 [==============================] - 1s 47ms/step - loss: 2.7705 - accuracy:
0.4204
Epoch 45/200
16/16 [==============================] - 1s 45ms/step - loss: 2.7095 - accuracy:
0.4245
Epoch 46/200
16/16 [==============================] - 1s 45ms/step - loss: 2.6474 - accuracy:
0.4408
Epoch 47/200
16/16 [==============================] - 1s 46ms/step - loss: 2.5835 - accuracy:
0.4531
Epoch 48/200
16/16 [==============================] - 1s 46ms/step - loss: 2.5212 - accuracy:
0.4612
Epoch 49/200
16/16 [==============================] - 1s 46ms/step - loss: 2.4596 - accuracy:
0.4755
Epoch 50/200
16/16 [==============================] - 1s 45ms/step - loss: 2.3982 - accuracy:
0.4898
Epoch 51/200
16/16 [==============================] - 1s 45ms/step - loss: 2.3386 - accuracy:
0.5020
Epoch 52/200
16/16 [==============================] - 1s 45ms/step - loss: 2.2797 - accuracy:
0.5122
Epoch 53/200
16/16 [==============================] - 1s 45ms/step - loss: 2.2248 - accuracy:
0.5429
Epoch 54/200
16/16 [==============================] - 1s 46ms/step - loss: 2.1679 - accuracy:
0.5612
Epoch 55/200
16/16 [==============================] - 1s 46ms/step - loss: 2.1168 - accuracy:
0.5776
Epoch 56/200
16/16 [==============================] - 1s 46ms/step - loss: 2.0605 - accuracy:
0.5898
Epoch 57/200
16/16 [==============================] - 1s 45ms/step - loss: 2.0127 - accuracy:
0.6102
Epoch 58/200
16/16 [==============================] - 1s 45ms/step - loss: 1.9588 - accuracy:
```

```
0.6408
Epoch 59/200
16/16 [==============================] - 1s 45ms/step - loss: 1.9082 - accuracy:
0.6490
Epoch 60/200
16/16 [==============================] - 1s 46ms/step - loss: 1.8568 - accuracy:
0.6633
Epoch 61/200
16/16 [==============================] - 1s 45ms/step - loss: 1.8071 - accuracy:
0.6735
Epoch 62/200
16/16 [==============================] - 1s 46ms/step - loss: 1.7623 - accuracy:
0.6776
Epoch 63/200
16/16 [==============================] - 1s 46ms/step - loss: 1.7162 - accuracy:
0.6878
Epoch 64/200
16/16 [==============================] - 1s 45ms/step - loss: 1.6713 - accuracy:
0.7020
Epoch 65/200
16/16 [==============================] - 1s 45ms/step - loss: 1.6262 - accuracy:
0.7061
Epoch 66/200
16/16 [==============================] - 1s 45ms/step - loss: 1.5830 - accuracy:
0.7061
Epoch 67/200
16/16 [==============================] - 1s 45ms/step - loss: 1.5431 - accuracy:
0.7143
Epoch 68/200
16/16 [==============================] - 1s 46ms/step - loss: 1.5007 - accuracy:
0.7245
Epoch 69/200
16/16 [==============================] - 1s 45ms/step - loss: 1.4647 - accuracy:
0.7347
Epoch 70/200
16/16 [==============================] - 1s 45ms/step - loss: 1.4269 - accuracy:
0.7367
Epoch 71/200
16/16 [==============================] - 1s 46ms/step - loss: 1.3899 - accuracy:
0.7429
Epoch 72/200
16/16 [==============================] - 1s 45ms/step - loss: 1.3538 - accuracy:
0.7490
Epoch 73/200
16/16 [==============================] - 1s 46ms/step - loss: 1.3154 - accuracy:
0.7531
Epoch 74/200
16/16 [==============================] - 1s 46ms/step - loss: 1.2808 - accuracy:
```

```
0.7612
Epoch 75/200
16/16 [==============================] - 1s 45ms/step - loss: 1.2493 - accuracy:
0.7755
Epoch 76/200
16/16 [==============================] - 1s 45ms/step - loss: 1.2187 - accuracy:
0.7776
Epoch 77/200
16/16 [==============================] - 1s 46ms/step - loss: 1.1880 - accuracy:
0.7898
Epoch 78/200
16/16 [==============================] - 1s 45ms/step - loss: 1.1575 - accuracy:
0.7837
Epoch 79/200
16/16 [==============================] - 1s 46ms/step - loss: 1.1302 - accuracy:
0.7939
Epoch 80/200
16/16 [==============================] - 1s 45ms/step - loss: 1.1012 - accuracy:
0.7898
Epoch 81/200
16/16 [==============================] - 1s 44ms/step - loss: 1.0751 - accuracy:
0.8041
Epoch 82/200
16/16 [==============================] - 1s 45ms/step - loss: 1.0485 - accuracy:
0.8122
Epoch 83/200
16/16 [==============================] - 1s 44ms/step - loss: 1.0244 - accuracy:
0.8224
Epoch 84/200
16/16 [==============================] - 1s 45ms/step - loss: 1.0015 - accuracy:
0.8163
Epoch 85/200
16/16 [==============================] - 1s 46ms/step - loss: 0.9764 - accuracy:
0.8367
Epoch 86/200
16/16 [==============================] - 1s 45ms/step - loss: 0.9517 - accuracy:
0.8327
Epoch 87/200
16/16 [==============================] - 1s 46ms/step - loss: 0.9316 - accuracy:
0.8347
Epoch 88/200
16/16 [==============================] - 1s 45ms/step - loss: 0.9094 - accuracy:
0.8449
Epoch 89/200
16/16 [==============================] - 1s 47ms/step - loss: 0.8916 - accuracy:
0.8388
Epoch 90/200
16/16 [==============================] - 1s 53ms/step - loss: 0.8723 - accuracy:
```

```
0.8469
Epoch 91/200
16/16 [==============================] - 1s 51ms/step - loss: 0.8517 - accuracy:
0.8531
Epoch 92/200
16/16 [==============================] - 1s 51ms/step - loss: 0.8358 - accuracy:
0.8449
Epoch 93/200
16/16 [==============================] - 1s 49ms/step - loss: 0.8174 - accuracy:
0.8633
Epoch 94/200
16/16 [==============================] - 1s 46ms/step - loss: 0.7967 - accuracy:
0.8673
Epoch 95/200
16/16 [==============================] - 1s 48ms/step - loss: 0.7813 - accuracy:
0.8694
Epoch 96/200
16/16 [==============================] - 1s 52ms/step - loss: 0.7665 - accuracy:
0.8673
Epoch 97/200
16/16 [==============================] - 1s 48ms/step - loss: 0.7511 - accuracy:
0.8714
Epoch 98/200
16/16 [==============================] - 1s 47ms/step - loss: 0.7356 - accuracy:
0.8816
Epoch 99/200
16/16 [==============================] - 1s 47ms/step - loss: 0.7215 - accuracy:
0.8857
Epoch 100/200
16/16 [==============================] - 1s 46ms/step - loss: 0.7063 - accuracy:
0.8898
Epoch 101/200
16/16 [==============================] - 1s 47ms/step - loss: 0.6992 - accuracy:
0.8898
Epoch 102/200
16/16 [==============================] - 1s 45ms/step - loss: 0.6800 - accuracy:
0.8939
Epoch 103/200
16/16 [==============================] - 1s 49ms/step - loss: 0.6661 - accuracy:
0.8939
Epoch 104/200
16/16 [==============================] - 1s 48ms/step - loss: 0.6798 - accuracy:
0.9061
Epoch 105/200
16/16 [==============================] - 1s 49ms/step - loss: 0.6417 - accuracy:
0.9020
Epoch 106/200
16/16 [==============================] - 1s 49ms/step - loss: 0.6319 - accuracy:
```

```
0.9163
Epoch 107/200
16/16 [==============================] - 1s 49ms/step - loss: 0.6187 - accuracy:
0.9143
Epoch 108/200
16/16 [==============================] - 1s 48ms/step - loss: 0.6097 - accuracy:
0.9163
Epoch 109/200
16/16 [==============================] - 1s 46ms/step - loss: 0.5977 - accuracy:
0.9204
Epoch 110/200
16/16 [==============================] - 1s 49ms/step - loss: 0.5839 - accuracy:
0.9224
Epoch 111/200
16/16 [==============================] - 1s 48ms/step - loss: 0.5720 - accuracy:
0.9265
Epoch 112/200
16/16 [==============================] - 1s 47ms/step - loss: 0.5618 - accuracy:
0.9265
Epoch 113/200
16/16 [==============================] - 1s 50ms/step - loss: 0.5510 - accuracy:
0.9286
Epoch 114/200
16/16 [==============================] - 1s 50ms/step - loss: 0.5443 - accuracy:
0.9306
Epoch 115/200
16/16 [==============================] - 1s 49ms/step - loss: 0.5335 - accuracy:
0.9327
Epoch 116/200
16/16 [==============================] - 1s 48ms/step - loss: 0.5210 - accuracy:
0.9286
Epoch 117/200
16/16 [==============================] - 1s 55ms/step - loss: 0.5123 - accuracy:
0.9347
Epoch 118/200
16/16 [==============================] - 1s 47ms/step - loss: 0.5054 - accuracy:
0.9367
Epoch 119/200
16/16 [==============================] - 1s 49ms/step - loss: 0.4949 - accuracy:
0.9408
Epoch 120/200
16/16 [==============================] - 1s 51ms/step - loss: 0.4859 - accuracy:
0.9388
Epoch 121/200
16/16 [==============================] - 1s 47ms/step - loss: 0.4791 - accuracy:
0.9429
Epoch 122/200
16/16 [==============================] - 1s 51ms/step - loss: 0.4687 - accuracy:
```

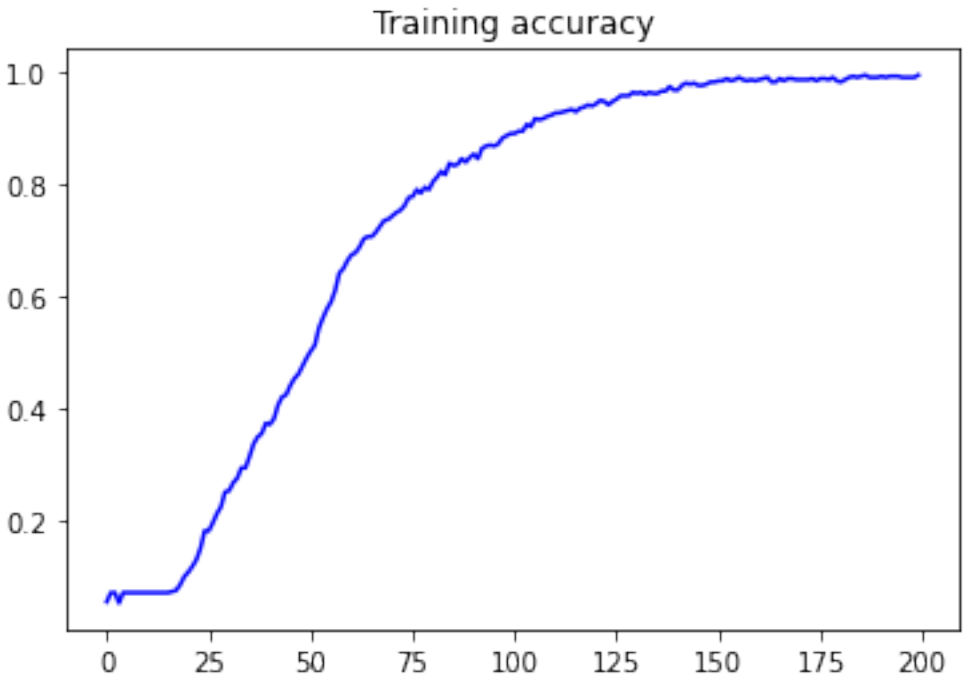
```
0.9490
Epoch 123/200
16/16 [==============================] - 1s 48ms/step - loss: 0.4601 - accuracy:
0.9469
Epoch 124/200
16/16 [==============================] - 1s 45ms/step - loss: 0.4528 - accuracy:
0.9408
Epoch 125/200
16/16 [==============================] - 1s 46ms/step - loss: 0.4457 - accuracy:
0.9469
Epoch 126/200
16/16 [==============================] - 1s 45ms/step - loss: 0.4379 - accuracy:
0.9510
Epoch 127/200
16/16 [==============================] - 1s 46ms/step - loss: 0.4303 - accuracy:
0.9571
Epoch 128/200
16/16 [==============================] - 1s 46ms/step - loss: 0.4221 - accuracy:
0.9571
Epoch 129/200
16/16 [==============================] - 1s 48ms/step - loss: 0.4147 - accuracy:
0.9571
Epoch 130/200
16/16 [==============================] - 1s 50ms/step - loss: 0.4069 - accuracy:
0.9633
Epoch 131/200
16/16 [==============================] - 1s 46ms/step - loss: 0.4021 - accuracy:
0.9612
Epoch 132/200
16/16 [==============================] - 1s 46ms/step - loss: 0.3935 - accuracy:
0.9633
Epoch 133/200
16/16 [==============================] - 1s 46ms/step - loss: 0.3870 - accuracy:
0.9592
Epoch 134/200
16/16 [==============================] - 1s 46ms/step - loss: 0.3812 - accuracy:
0.9633
Epoch 135/200
16/16 [==============================] - 1s 46ms/step - loss: 0.3739 - accuracy:
0.9612
Epoch 136/200
16/16 [==============================] - 1s 46ms/step - loss: 0.3671 - accuracy:
0.9612
Epoch 137/200
16/16 [==============================] - 1s 44ms/step - loss: 0.3594 - accuracy:
0.9653
Epoch 138/200
16/16 [==============================] - 1s 46ms/step - loss: 0.3542 - accuracy:
```

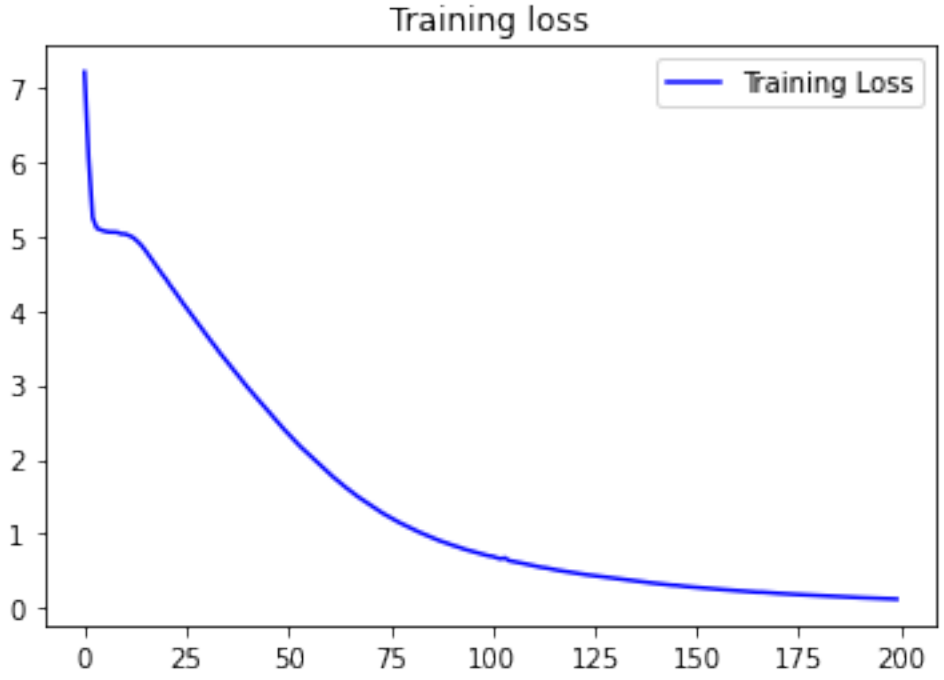
```
0.9653
Epoch 139/200
16/16 [==============================] - 1s 46ms/step - loss: 0.3482 - accuracy:
0.9735
Epoch 140/200
16/16 [==============================] - 1s 46ms/step - loss: 0.3420 - accuracy:
0.9673
Epoch 141/200
16/16 [==============================] - 1s 46ms/step - loss: 0.3375 - accuracy:
0.9673
Epoch 142/200
16/16 [==============================] - 1s 47ms/step - loss: 0.3305 - accuracy:
0.9755
Epoch 143/200
16/16 [==============================] - 1s 50ms/step - loss: 0.3240 - accuracy:
0.9796
Epoch 144/200
16/16 [==============================] - 1s 48ms/step - loss: 0.3193 - accuracy:
0.9776
Epoch 145/200
16/16 [==============================] - 1s 48ms/step - loss: 0.3120 - accuracy:
0.9796
Epoch 146/200
16/16 [==============================] - 1s 53ms/step - loss: 0.3102 - accuracy:
0.9755
Epoch 147/200
16/16 [==============================] - 1s 54ms/step - loss: 0.3030 - accuracy:
0.9755
Epoch 148/200
16/16 [==============================] - 1s 48ms/step - loss: 0.2965 - accuracy:
0.9776
Epoch 149/200
16/16 [==============================] - 1s 46ms/step - loss: 0.2931 - accuracy:
0.9816
Epoch 150/200
16/16 [==============================] - 1s 47ms/step - loss: 0.2870 - accuracy:
0.9816
Epoch 151/200
16/16 [==============================] - 1s 51ms/step - loss: 0.2818 - accuracy:
0.9837
Epoch 152/200
16/16 [==============================] - 1s 47ms/step - loss: 0.2778 - accuracy:
0.9837
Epoch 153/200
16/16 [==============================] - 1s 47ms/step - loss: 0.2732 - accuracy:
0.9878
Epoch 154/200
16/16 [==============================] - 1s 46ms/step - loss: 0.2683 - accuracy:
```

```
0.9837
Epoch 155/200
16/16 [==============================] - 1s 47ms/step - loss: 0.2648 - accuracy:
0.9857
Epoch 156/200
16/16 [==============================] - 1s 49ms/step - loss: 0.2589 - accuracy:
0.9898
Epoch 157/200
16/16 [==============================] - 1s 48ms/step - loss: 0.2543 - accuracy:
0.9857
Epoch 158/200
16/16 [==============================] - 1s 46ms/step - loss: 0.2499 - accuracy:
0.9837
Epoch 159/200
16/16 [==============================] - 1s 46ms/step - loss: 0.2458 - accuracy:
0.9857
Epoch 160/200
16/16 [==============================] - 1s 47ms/step - loss: 0.2417 - accuracy:
0.9837
Epoch 161/200
16/16 [==============================] - 1s 46ms/step - loss: 0.2382 - accuracy:
0.9857
Epoch 162/200
16/16 [==============================] - 1s 47ms/step - loss: 0.2338 - accuracy:
0.9878
Epoch 163/200
16/16 [==============================] - 1s 46ms/step - loss: 0.2298 - accuracy:
0.9898
Epoch 164/200
16/16 [==============================] - 1s 47ms/step - loss: 0.2261 - accuracy:
0.9816
Epoch 165/200
16/16 [==============================] - 1s 46ms/step - loss: 0.2219 - accuracy:
0.9816
Epoch 166/200
16/16 [==============================] - 1s 46ms/step - loss: 0.2178 - accuracy:
0.9878
Epoch 167/200
16/16 [==============================] - 1s 46ms/step - loss: 0.2181 - accuracy:
0.9837
Epoch 168/200
16/16 [==============================] - 1s 45ms/step - loss: 0.2132 - accuracy:
0.9878
Epoch 169/200
16/16 [==============================] - 1s 46ms/step - loss: 0.2088 - accuracy:
0.9878
Epoch 170/200
16/16 [==============================] - 1s 45ms/step - loss: 0.2054 - accuracy:
```

```
0.9857
Epoch 171/200
16/16 [==============================] - 1s 51ms/step - loss: 0.2021 - accuracy:
0.9857
Epoch 172/200
16/16 [==============================] - 1s 50ms/step - loss: 0.1998 - accuracy:
0.9857
Epoch 173/200
16/16 [==============================] - 1s 51ms/step - loss: 0.1952 - accuracy:
0.9857
Epoch 174/200
16/16 [==============================] - 1s 48ms/step - loss: 0.1913 - accuracy:
0.9878
Epoch 175/200
16/16 [==============================] - 1s 48ms/step - loss: 0.1893 - accuracy:
0.9837
Epoch 176/200
16/16 [==============================] - 1s 48ms/step - loss: 0.1864 - accuracy:
0.9878
Epoch 177/200
16/16 [==============================] - 1s 47ms/step - loss: 0.1822 - accuracy:
0.9878
Epoch 178/200
16/16 [==============================] - 1s 46ms/step - loss: 0.1793 - accuracy:
0.9857
Epoch 179/200
16/16 [==============================] - 1s 46ms/step - loss: 0.1766 - accuracy:
0.9898
Epoch 180/200
16/16 [==============================] - 1s 46ms/step - loss: 0.1745 - accuracy:
0.9837
Epoch 181/200
16/16 [==============================] - 1s 46ms/step - loss: 0.1717 - accuracy:
0.9816
Epoch 182/200
16/16 [==============================] - 1s 46ms/step - loss: 0.1696 - accuracy:
0.9857
Epoch 183/200
16/16 [==============================] - 1s 45ms/step - loss: 0.1654 - accuracy:
0.9898
Epoch 184/200
16/16 [==============================] - 1s 46ms/step - loss: 0.1627 - accuracy:
0.9918
Epoch 185/200
16/16 [==============================] - 1s 45ms/step - loss: 0.1601 - accuracy:
0.9898
Epoch 186/200
16/16 [==============================] - 1s 47ms/step - loss: 0.1575 - accuracy:
```

```
0.9918
Epoch 187/200
16/16 [==============================] - 1s 46ms/step - loss: 0.1556 - accuracy:
0.9939
Epoch 188/200
16/16 [==============================] - 1s 47ms/step - loss: 0.1525 - accuracy:
0.9898
Epoch 189/200
16/16 [==============================] - 1s 46ms/step - loss: 0.1505 - accuracy:
0.9898
Epoch 190/200
16/16 [==============================] - 1s 46ms/step - loss: 0.1479 - accuracy:
0.9898
Epoch 191/200
16/16 [==============================] - 1s 47ms/step - loss: 0.1452 - accuracy:
0.9918
Epoch 192/200
16/16 [==============================] - 1s 45ms/step - loss: 0.1435 - accuracy:
0.9898
Epoch 193/200
16/16 [==============================] - 1s 46ms/step - loss: 0.1413 - accuracy:
0.9918
Epoch 194/200
16/16 [==============================] - 1s 45ms/step - loss: 0.1382 - accuracy:
0.9918
Epoch 195/200
16/16 [==============================] - 1s 45ms/step - loss: 0.1372 - accuracy:
0.9918
Epoch 196/200
16/16 [==============================] - 1s 46ms/step - loss: 0.1347 - accuracy:
0.9898
Epoch 197/200
16/16 [==============================] - 1s 45ms/step - loss: 0.1325 - accuracy:
0.9898
Epoch 198/200
16/16 [==============================] - 1s 46ms/step - loss: 0.1301 - accuracy:
0.9898
Epoch 199/200
16/16 [==============================] - 1s 46ms/step - loss: 0.1281 - accuracy:
0.9898
Epoch 200/200
16/16 [==============================] - 1s 46ms/step - loss: 0.1254 - accuracy:
0.9939
```





[33]:
```
#TASK 2
# Implement the generate() function
```

```
def generate(seed_text):
    ### START CODE HERE ###
    count = []
    text = seed_text
    while True:
        token_list = tokenizer.texts_to_sequences([text])
        print(token_list)
        token_list = pad_sequences(token_list, maxlen=60, padding='pre')
        print(token_list)
        predicted = np.argmax(model1.predict(token_list),axis=-1)
        output_word = ""
        for word, index in tokenizer.word_index.items():
            if index == predicted:
                output_word = word
                count.append(word)
                break
        text += " " + output_word
        if len(count) > 1:
            if count[-1] == word:
                text += "."
                break
        if output_word == '.' or output_word == '?' or output_word == '!':
            break
        if len(count) == 250:
            text += "."
            break
    return text
    ### END CODE HERE ###
```

[36]:
```
print(generate("moogles"))
```

```
[[565]]
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0 565]]
1/1 [==============================] - 0s 21ms/step
[[565, 3]]
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0 565   3]]
1/1 [==============================] - 0s 20ms/step
moogles , however.
```

Export your notebook to a pdf document

[39]:
```
!jupyter nbconvert --to pdf 'YOUR_LINK_TO_THE_IPYNOTE_NOTEBOOK'
```

```
This application is used to convert notebook files (*.ipynb)
        to various other formats.

        WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.

Options
=======
The options below are convenience aliases to configurable class-options,
as listed in the "Equivalent to" description-line of the aliases.
To see all configurable class-options for some <cmd>, use:
    <cmd> --help-all

--debug
    set log level to logging.DEBUG (maximize logging output)
    Equivalent to: [--Application.log_level=10]
--show-config
    Show the application's configuration (human-readable format)
    Equivalent to: [--Application.show_config=True]
--show-config-json
    Show the application's configuration (json format)
    Equivalent to: [--Application.show_config_json=True]
--generate-config
    generate default config file
    Equivalent to: [--JupyterApp.generate_config=True]
-y
    Answer yes to any questions instead of prompting.
    Equivalent to: [--JupyterApp.answer_yes=True]
--execute
    Execute the notebook prior to export.
    Equivalent to: [--ExecutePreprocessor.enabled=True]
--allow-errors
    Continue notebook execution even if one of the cells throws an error and
include the error message in the cell output (the default behaviour is to abort
conversion). This flag is only relevant if '--execute' was specified, too.
    Equivalent to: [--ExecutePreprocessor.allow_errors=True]
--stdin
    read a single notebook file from stdin. Write the resulting notebook with
default basename 'notebook.*'
    Equivalent to: [--NbConvertApp.from_stdin=True]
--stdout
    Write notebook output to stdout instead of files.
    Equivalent to: [--NbConvertApp.writer_class=StdoutWriter]
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
            relevant when converting to notebook format)
    Equivalent to: [--NbConvertApp.use_output_suffix=False
--NbConvertApp.export_format=notebook --FilesWriter.build_directory=]
--clear-output
```

```
    Clear output of current file and save in place,
            overwriting the existing notebook.
    Equivalent to: [--NbConvertApp.use_output_suffix=False
--NbConvertApp.export_format=notebook --FilesWriter.build_directory=
--ClearOutputPreprocessor.enabled=True]
--no-prompt
    Exclude input and output prompts from converted document.
    Equivalent to: [--TemplateExporter.exclude_input_prompt=True
--TemplateExporter.exclude_output_prompt=True]
--no-input
    Exclude input cells and output prompts from converted document.
            This mode is ideal for generating code-free reports.
    Equivalent to: [--TemplateExporter.exclude_output_prompt=True
--TemplateExporter.exclude_input=True]
--log-level=<Enum>
    Set the log level by value or name.
    Choices: any of [0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR',
'CRITICAL']
    Default: 30
    Equivalent to: [--Application.log_level]
--config=<Unicode>
    Full path of a config file.
    Default: ''
    Equivalent to: [--JupyterApp.config_file]
--to=<Unicode>
    The export format to be used, either one of the built-in formats
            ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook',
'pdf', 'python', 'rst', 'script', 'slides']
            or a dotted object name that represents the import path for an
            `Exporter` class
    Default: 'html'
    Equivalent to: [--NbConvertApp.export_format]
--template=<Unicode>
    Name of the template file to use
    Default: ''
    Equivalent to: [--TemplateExporter.template_file]
--writer=<DottedObjectName>
    Writer class used to write the
                                        results of the conversion
    Default: 'FilesWriter'
    Equivalent to: [--NbConvertApp.writer_class]
--post=<DottedOrNone>
    PostProcessor class used to write the
                                        results of the conversion
    Default: ''
    Equivalent to: [--NbConvertApp.postprocessor_class]
--output=<Unicode>
    overwrite base name use for output files.
```

```
                can only be used when converting one notebook at a time.
    Default: ''
    Equivalent to: [--NbConvertApp.output_base]
--output-dir=<Unicode>
    Directory to write output(s) to. Defaults
                                    to output to the directory of each notebook.
To recover
                                    previous default behaviour (outputting to the
current
                                    working directory) use . as the flag value.
    Default: ''
    Equivalent to: [--FilesWriter.build_directory]
--reveal-prefix=<Unicode>
    The URL prefix for reveal.js (version 3.x).
            This defaults to the reveal CDN, but can be any url pointing to a
copy
            of reveal.js.
            For speaker notes to work, this must be a relative path to a local
            copy of reveal.js: e.g., "reveal.js".
            If a relative path is given, it must be a subdirectory of the
            current directory (from which the server is run).
            See the usage documentation
            (https://nbconvert.readthedocs.io/en/latest/usage.html#reveal-js-
html-slideshow)
            for more details.
    Default: ''
    Equivalent to: [--SlidesExporter.reveal_url_prefix]
--nbformat=<Enum>
    The nbformat version to write.
            Use this to downgrade notebooks.
    Choices: any of [1, 2, 3, 4]
    Default: 4
    Equivalent to: [--NotebookExporter.nbformat_version]

Examples
--------

    The simplest way to use nbconvert is

            > jupyter nbconvert mynotebook.ipynb

            which will convert mynotebook.ipynb to the default format (probably
HTML).

            You can specify the export format with `--to`.
            Options include ['asciidoc', 'custom', 'html', 'latex', 'markdown',
'notebook', 'pdf', 'python', 'rst', 'script', 'slides'].
```

```
            > jupyter nbconvert --to latex mynotebook.ipynb

            Both HTML and LaTeX support multiple output templates. LaTeX
includes
            'base', 'article' and 'report'.  HTML includes 'basic' and 'full'.
You
            can specify the flavor of the format used.

            > jupyter nbconvert --to html --template basic mynotebook.ipynb

            You can also pipe the output to stdout, rather than a file

            > jupyter nbconvert mynotebook.ipynb --stdout

            PDF is generated via latex

            > jupyter nbconvert mynotebook.ipynb --to pdf

            You can get (and serve) a Reveal.js-powered slideshow

            > jupyter nbconvert myslides.ipynb --to slides --post serve

            Multiple notebooks can be given at the command line in a couple of
            different ways:

            > jupyter nbconvert notebook*.ipynb
            > jupyter nbconvert notebook1.ipynb notebook2.ipynb

            or you can specify the notebooks list in a config file, containing::

                c.NbConvertApp.notebooks = ["my_notebook.ipynb"]

            > jupyter nbconvert --config mycfg.py

To see all available configurables, use `--help-all`.


[NbConvertApp] WARNING | pattern "'YOUR_LINK_TO_THE_IPYNOTE_NOTEBOOK'" matched
no files
```

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!