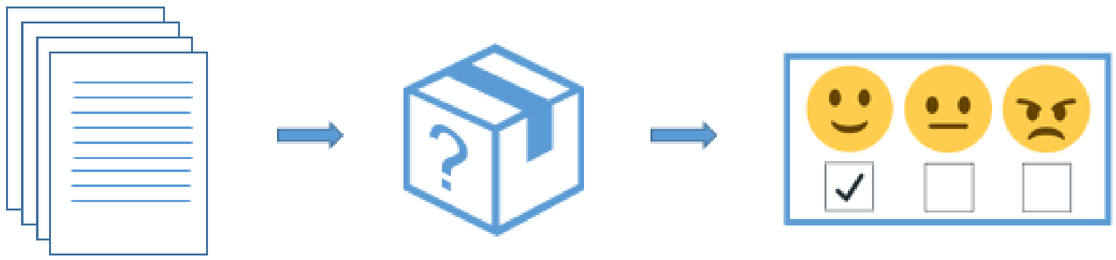
5.2D - DeepFakeComments Generator

August 25, 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, SGD
     from tensorflow.keras import regularizers
     import tensorflow.keras.utils as ku
     import keras.backend as K
     import matplotlib.pyplot as plt
     import numpy as np
```

1 Build the model

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

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

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

We also extract the vocabulary of the data.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

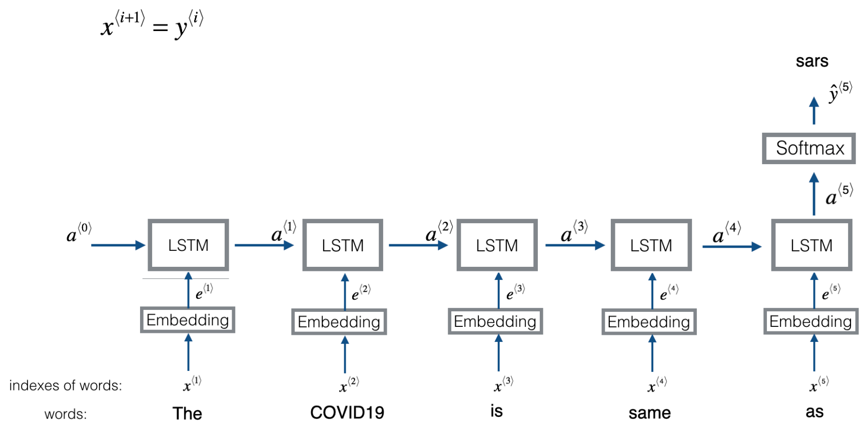
[10]:
```
print("The entry ",sample," in 'input_sequences' is: ")
print(input_sequences[sample])
print(", it corresponds to the following input to our model:")
print(input_to_model[sample])
print(" and the following output: ", label[sample])
```

```
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()
    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=SGD(learning_rate=0.001), metrics=['accuracy'])
    return model
    ### END CODE HERE ###

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

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 [==============================] - 9s 56ms/step - loss: 7.1358 -
accuracy: 0.0087
Epoch 2/200
126/126 [==============================] - 7s 57ms/step - loss: 7.1339 -
accuracy: 0.0593
Epoch 3/200
126/126 [==============================] - 7s 57ms/step - loss: 7.1320 -
accuracy: 0.0700
Epoch 4/200
126/126 [==============================] - 7s 57ms/step - loss: 7.1301 -
accuracy: 0.0732
Epoch 5/200
126/126 [==============================] - 7s 57ms/step - loss: 7.1282 -
accuracy: 0.0732
Epoch 6/200
126/126 [==============================] - 7s 57ms/step - loss: 7.1263 -
accuracy: 0.0734
Epoch 7/200
126/126 [==============================] - 7s 57ms/step - loss: 7.1244 -
accuracy: 0.0734
Epoch 8/200
126/126 [==============================] - 7s 57ms/step - loss: 7.1225 -
accuracy: 0.0734
Epoch 9/200
126/126 [==============================] - 7s 57ms/step - loss: 7.1206 -
accuracy: 0.0734
Epoch 10/200
126/126 [==============================] - 7s 57ms/step - loss: 7.1187 -
accuracy: 0.0734
Epoch 11/200
126/126 [==============================] - 7s 57ms/step - loss: 7.1168 -
accuracy: 0.0734
```

```
Epoch 12/200
126/126 [==============================] - 7s 57ms/step - loss: 7.1148 -
accuracy: 0.0734
Epoch 13/200
126/126 [==============================] - 6s 48ms/step - loss: 7.1129 -
accuracy: 0.0734
Epoch 14/200
126/126 [==============================] - 5s 36ms/step - loss: 7.1110 -
accuracy: 0.0734
Epoch 15/200
126/126 [==============================] - 5s 37ms/step - loss: 7.1090 -
accuracy: 0.0734
Epoch 16/200
126/126 [==============================] - 5s 37ms/step - loss: 7.1070 -
accuracy: 0.0734
Epoch 17/200
126/126 [==============================] - 5s 37ms/step - loss: 7.1051 -
accuracy: 0.0734
Epoch 18/200
126/126 [==============================] - 5s 37ms/step - loss: 7.1031 -
accuracy: 0.0734
Epoch 19/200
126/126 [==============================] - 5s 37ms/step - loss: 7.1011 -
accuracy: 0.0734
Epoch 20/200
126/126 [==============================] - 5s 37ms/step - loss: 7.0991 -
accuracy: 0.0734
Epoch 21/200
126/126 [==============================] - 5s 37ms/step - loss: 7.0971 -
accuracy: 0.0734
Epoch 22/200
126/126 [==============================] - 5s 37ms/step - loss: 7.0950 -
accuracy: 0.0734
Epoch 23/200
126/126 [==============================] - 5s 36ms/step - loss: 7.0930 -
accuracy: 0.0734
Epoch 24/200
126/126 [==============================] - 5s 37ms/step - loss: 7.0909 -
accuracy: 0.0734
Epoch 25/200
126/126 [==============================] - 5s 36ms/step - loss: 7.0888 -
accuracy: 0.0734
Epoch 26/200
126/126 [==============================] - 5s 36ms/step - loss: 7.0867 -
accuracy: 0.0734
Epoch 27/200
126/126 [==============================] - 5s 36ms/step - loss: 7.0846 -
accuracy: 0.0734
```

```
Epoch 28/200
126/126 [==============================] - 5s 37ms/step - loss: 7.0824 -
accuracy: 0.0734
Epoch 29/200
126/126 [==============================] - 5s 36ms/step - loss: 7.0802 -
accuracy: 0.0734
Epoch 30/200
126/126 [==============================] - 5s 36ms/step - loss: 7.0780 -
accuracy: 0.0734
Epoch 31/200
126/126 [==============================] - 5s 36ms/step - loss: 7.0757 -
accuracy: 0.0734
Epoch 32/200
126/126 [==============================] - 5s 36ms/step - loss: 7.0734 -
accuracy: 0.0734
Epoch 33/200
126/126 [==============================] - 5s 36ms/step - loss: 7.0711 -
accuracy: 0.0734
Epoch 34/200
126/126 [==============================] - 5s 36ms/step - loss: 7.0687 -
accuracy: 0.0734
Epoch 35/200
126/126 [==============================] - 5s 36ms/step - loss: 7.0663 -
accuracy: 0.0734
Epoch 36/200
126/126 [==============================] - 5s 36ms/step - loss: 7.0638 -
accuracy: 0.0734
Epoch 37/200
126/126 [==============================] - 5s 36ms/step - loss: 7.0613 -
accuracy: 0.0734
Epoch 38/200
126/126 [==============================] - 5s 36ms/step - loss: 7.0587 -
accuracy: 0.0734
Epoch 39/200
126/126 [==============================] - 5s 36ms/step - loss: 7.0561 -
accuracy: 0.0734
Epoch 40/200
126/126 [==============================] - 5s 36ms/step - loss: 7.0533 -
accuracy: 0.0734
Epoch 41/200
126/126 [==============================] - 5s 36ms/step - loss: 7.0505 -
accuracy: 0.0734
Epoch 42/200
126/126 [==============================] - 5s 36ms/step - loss: 7.0476 -
accuracy: 0.0734
Epoch 43/200
126/126 [==============================] - 4s 36ms/step - loss: 7.0446 -
accuracy: 0.0734
```

```
Epoch 44/200
126/126 [==============================] - 4s 36ms/step - loss: 7.0415 -
accuracy: 0.0734
Epoch 45/200
126/126 [==============================] - 4s 36ms/step - loss: 7.0383 -
accuracy: 0.0734
Epoch 46/200
126/126 [==============================] - 4s 36ms/step - loss: 7.0349 -
accuracy: 0.0734
Epoch 47/200
126/126 [==============================] - 5s 36ms/step - loss: 7.0314 -
accuracy: 0.0734
Epoch 48/200
126/126 [==============================] - 5s 36ms/step - loss: 7.0277 -
accuracy: 0.0734
Epoch 49/200
126/126 [==============================] - 5s 36ms/step - loss: 7.0237 -
accuracy: 0.0734
Epoch 50/200
126/126 [==============================] - 5s 36ms/step - loss: 7.0196 -
accuracy: 0.0734
Epoch 51/200
126/126 [==============================] - 5s 36ms/step - loss: 7.0151 -
accuracy: 0.0734
Epoch 52/200
126/126 [==============================] - 5s 36ms/step - loss: 7.0104 -
accuracy: 0.0734
Epoch 53/200
126/126 [==============================] - 5s 37ms/step - loss: 7.0053 -
accuracy: 0.0734
Epoch 54/200
126/126 [==============================] - 5s 36ms/step - loss: 6.9997 -
accuracy: 0.0734
Epoch 55/200
126/126 [==============================] - 5s 36ms/step - loss: 6.9936 -
accuracy: 0.0734
Epoch 56/200
126/126 [==============================] - 5s 36ms/step - loss: 6.9869 -
accuracy: 0.0734
Epoch 57/200
126/126 [==============================] - 5s 36ms/step - loss: 6.9795 -
accuracy: 0.0734
Epoch 58/200
126/126 [==============================] - 5s 38ms/step - loss: 6.9713 -
accuracy: 0.0734
Epoch 59/200
126/126 [==============================] - 5s 36ms/step - loss: 6.9620 -
accuracy: 0.0734
```

```
Epoch 60/200
126/126 [==============================] - 5s 36ms/step - loss: 6.9515 -
accuracy: 0.0734
Epoch 61/200
126/126 [==============================] - 5s 36ms/step - loss: 6.9396 -
accuracy: 0.0734
Epoch 62/200
126/126 [==============================] - 5s 36ms/step - loss: 6.9262 -
accuracy: 0.0734
Epoch 63/200
126/126 [==============================] - 5s 36ms/step - loss: 6.9109 -
accuracy: 0.0734
Epoch 64/200
126/126 [==============================] - 5s 36ms/step - loss: 6.8939 -
accuracy: 0.0734
Epoch 65/200
126/126 [==============================] - 5s 36ms/step - loss: 6.8749 -
accuracy: 0.0734
Epoch 66/200
126/126 [==============================] - 5s 36ms/step - loss: 6.8541 -
accuracy: 0.0734
Epoch 67/200
126/126 [==============================] - 5s 36ms/step - loss: 6.8315 -
accuracy: 0.0734
Epoch 68/200
126/126 [==============================] - 4s 36ms/step - loss: 6.8074 -
accuracy: 0.0734
Epoch 69/200
126/126 [==============================] - 5s 36ms/step - loss: 6.7818 -
accuracy: 0.0734
Epoch 70/200
126/126 [==============================] - 5s 36ms/step - loss: 6.7550 -
accuracy: 0.0734
Epoch 71/200
126/126 [==============================] - 5s 36ms/step - loss: 6.7272 -
accuracy: 0.0734
Epoch 72/200
126/126 [==============================] - 5s 36ms/step - loss: 6.6985 -
accuracy: 0.0734
Epoch 73/200
126/126 [==============================] - 4s 36ms/step - loss: 6.6692 -
accuracy: 0.0734
Epoch 74/200
126/126 [==============================] - 4s 36ms/step - loss: 6.6395 -
accuracy: 0.0734
Epoch 75/200
126/126 [==============================] - 4s 36ms/step - loss: 6.6099 -
accuracy: 0.0734
```

```
Epoch 76/200
126/126 [==============================] - 5s 36ms/step - loss: 6.5806 -
accuracy: 0.0734
Epoch 77/200
126/126 [==============================] - 5s 36ms/step - loss: 6.5519 -
accuracy: 0.0734
Epoch 78/200
126/126 [==============================] - 5s 36ms/step - loss: 6.5243 -
accuracy: 0.0734
Epoch 79/200
126/126 [==============================] - 5s 36ms/step - loss: 6.4980 -
accuracy: 0.0734
Epoch 80/200
126/126 [==============================] - 5s 36ms/step - loss: 6.4733 -
accuracy: 0.0734
Epoch 81/200
126/126 [==============================] - 5s 36ms/step - loss: 6.4502 -
accuracy: 0.0734
Epoch 82/200
126/126 [==============================] - 5s 36ms/step - loss: 6.4289 -
accuracy: 0.0734
Epoch 83/200
126/126 [==============================] - 5s 36ms/step - loss: 6.4092 -
accuracy: 0.0734
Epoch 84/200
126/126 [==============================] - 5s 36ms/step - loss: 6.3909 -
accuracy: 0.0734
Epoch 85/200
126/126 [==============================] - 5s 37ms/step - loss: 6.3740 -
accuracy: 0.0734
Epoch 86/200
126/126 [==============================] - 5s 36ms/step - loss: 6.3580 -
accuracy: 0.0734
Epoch 87/200
126/126 [==============================] - 5s 37ms/step - loss: 6.3430 -
accuracy: 0.0734
Epoch 88/200
126/126 [==============================] - 5s 36ms/step - loss: 6.3287 -
accuracy: 0.0734
Epoch 89/200
126/126 [==============================] - 5s 36ms/step - loss: 6.3150 -
accuracy: 0.0734
Epoch 90/200
126/126 [==============================] - 5s 36ms/step - loss: 6.3019 -
accuracy: 0.0734
Epoch 91/200
126/126 [==============================] - 5s 36ms/step - loss: 6.2891 -
accuracy: 0.0734
```

```
Epoch 92/200
126/126 [==============================] - 5s 36ms/step - loss: 6.2768 -
accuracy: 0.0734
Epoch 93/200
126/126 [==============================] - 5s 36ms/step - loss: 6.2649 -
accuracy: 0.0734
Epoch 94/200
126/126 [==============================] - 5s 36ms/step - loss: 6.2533 -
accuracy: 0.0734
Epoch 95/200
126/126 [==============================] - 5s 36ms/step - loss: 6.2420 -
accuracy: 0.0734
Epoch 96/200
126/126 [==============================] - 5s 36ms/step - loss: 6.2311 -
accuracy: 0.0734
Epoch 97/200
126/126 [==============================] - 5s 36ms/step - loss: 6.2205 -
accuracy: 0.0734
Epoch 98/200
126/126 [==============================] - 5s 36ms/step - loss: 6.2102 -
accuracy: 0.0734
Epoch 99/200
126/126 [==============================] - 5s 36ms/step - loss: 6.2003 -
accuracy: 0.0734
Epoch 100/200
126/126 [==============================] - 5s 36ms/step - loss: 6.1907 -
accuracy: 0.0734
Epoch 101/200
126/126 [==============================] - 5s 36ms/step - loss: 6.1815 -
accuracy: 0.0734
Epoch 102/200
126/126 [==============================] - 5s 36ms/step - loss: 6.1727 -
accuracy: 0.0734
Epoch 103/200
126/126 [==============================] - 5s 36ms/step - loss: 6.1641 -
accuracy: 0.0734
Epoch 104/200
126/126 [==============================] - 5s 36ms/step - loss: 6.1560 -
accuracy: 0.0734
Epoch 105/200
126/126 [==============================] - 5s 36ms/step - loss: 6.1481 -
accuracy: 0.0734
Epoch 106/200
126/126 [==============================] - 5s 37ms/step - loss: 6.1406 -
accuracy: 0.0734
Epoch 107/200
126/126 [==============================] - 5s 37ms/step - loss: 6.1334 -
accuracy: 0.0734
```

```
Epoch 108/200
126/126 [==============================] - 5s 36ms/step - loss: 6.1266 -
accuracy: 0.0734
Epoch 109/200
126/126 [==============================] - 5s 37ms/step - loss: 6.1200 -
accuracy: 0.0734
Epoch 110/200
126/126 [==============================] - 5s 37ms/step - loss: 6.1138 -
accuracy: 0.0734
Epoch 111/200
126/126 [==============================] - 5s 36ms/step - loss: 6.1078 -
accuracy: 0.0734
Epoch 112/200
126/126 [==============================] - 5s 36ms/step - loss: 6.1020 -
accuracy: 0.0734
Epoch 113/200
126/126 [==============================] - 5s 37ms/step - loss: 6.0966 -
accuracy: 0.0734
Epoch 114/200
126/126 [==============================] - 5s 36ms/step - loss: 6.0913 -
accuracy: 0.0734
Epoch 115/200
126/126 [==============================] - 5s 37ms/step - loss: 6.0863 -
accuracy: 0.0734
Epoch 116/200
126/126 [==============================] - 5s 37ms/step - loss: 6.0815 -
accuracy: 0.0734
Epoch 117/200
126/126 [==============================] - 5s 37ms/step - loss: 6.0768 -
accuracy: 0.0734
Epoch 118/200
126/126 [==============================] - 5s 39ms/step - loss: 6.0723 -
accuracy: 0.0734
Epoch 119/200
126/126 [==============================] - 5s 36ms/step - loss: 6.0680 -
accuracy: 0.0734
Epoch 120/200
126/126 [==============================] - 4s 35ms/step - loss: 6.0639 -
accuracy: 0.0734
Epoch 121/200
126/126 [==============================] - 4s 35ms/step - loss: 6.0599 -
accuracy: 0.0734
Epoch 122/200
126/126 [==============================] - 4s 33ms/step - loss: 6.0560 -
accuracy: 0.0734
Epoch 123/200
126/126 [==============================] - 4s 33ms/step - loss: 6.0522 -
accuracy: 0.0734
```

```
Epoch 124/200
126/126 [==============================] - 4s 33ms/step - loss: 6.0486 -
accuracy: 0.0734
Epoch 125/200
126/126 [==============================] - 4s 33ms/step - loss: 6.0451 -
accuracy: 0.0734
Epoch 126/200
126/126 [==============================] - 4s 33ms/step - loss: 6.0416 -
accuracy: 0.0734
Epoch 127/200
126/126 [==============================] - 4s 33ms/step - loss: 6.0383 -
accuracy: 0.0734
Epoch 128/200
126/126 [==============================] - 4s 33ms/step - loss: 6.0351 -
accuracy: 0.0734
Epoch 129/200
126/126 [==============================] - 4s 33ms/step - loss: 6.0319 -
accuracy: 0.0734
Epoch 130/200
126/126 [==============================] - 4s 33ms/step - loss: 6.0288 -
accuracy: 0.0734
Epoch 131/200
126/126 [==============================] - 4s 33ms/step - loss: 6.0258 -
accuracy: 0.0734
Epoch 132/200
126/126 [==============================] - 4s 34ms/step - loss: 6.0229 -
accuracy: 0.0734
Epoch 133/200
126/126 [==============================] - 4s 33ms/step - loss: 6.0200 -
accuracy: 0.0734
Epoch 134/200
126/126 [==============================] - 4s 34ms/step - loss: 6.0172 -
accuracy: 0.0734
Epoch 135/200
126/126 [==============================] - 4s 33ms/step - loss: 6.0146 -
accuracy: 0.0734
Epoch 136/200
126/126 [==============================] - 4s 34ms/step - loss: 6.0119 -
accuracy: 0.0734
Epoch 137/200
126/126 [==============================] - 4s 33ms/step - loss: 6.0093 -
accuracy: 0.0734
Epoch 138/200
126/126 [==============================] - 4s 33ms/step - loss: 6.0068 -
accuracy: 0.0734
Epoch 139/200
126/126 [==============================] - 4s 33ms/step - loss: 6.0043 -
accuracy: 0.0734
```

```
Epoch 140/200
126/126 [==============================] - 4s 33ms/step - loss: 6.0019 -
accuracy: 0.0734
Epoch 141/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9996 -
accuracy: 0.0734
Epoch 142/200
126/126 [==============================] - 4s 34ms/step - loss: 5.9973 -
accuracy: 0.0734
Epoch 143/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9950 -
accuracy: 0.0734
Epoch 144/200
126/126 [==============================] - 4s 34ms/step - loss: 5.9929 -
accuracy: 0.0734
Epoch 145/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9907 -
accuracy: 0.0734
Epoch 146/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9887 -
accuracy: 0.0734
Epoch 147/200
126/126 [==============================] - 4s 34ms/step - loss: 5.9866 -
accuracy: 0.0734
Epoch 148/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9846 -
accuracy: 0.0734
Epoch 149/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9827 -
accuracy: 0.0734
Epoch 150/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9808 -
accuracy: 0.0734
Epoch 151/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9789 -
accuracy: 0.0734
Epoch 152/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9771 -
accuracy: 0.0734
Epoch 153/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9753 -
accuracy: 0.0734
Epoch 154/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9735 -
accuracy: 0.0734
Epoch 155/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9718 -
accuracy: 0.0734
```

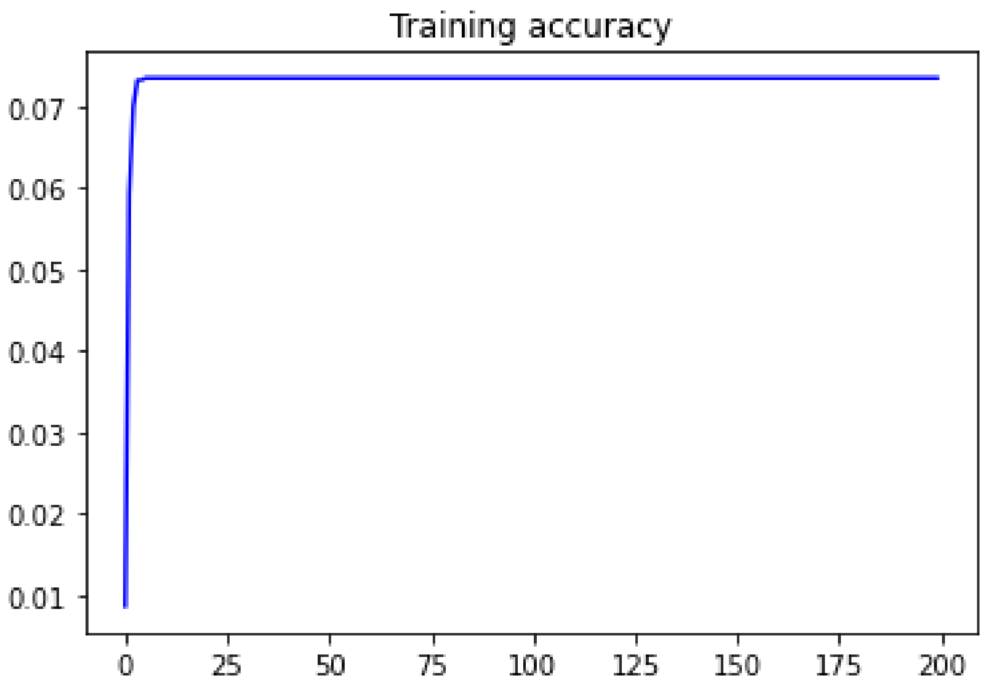
```
Epoch 156/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9701 -
accuracy: 0.0734
Epoch 157/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9685 -
accuracy: 0.0734
Epoch 158/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9669 -
accuracy: 0.0734
Epoch 159/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9653 -
accuracy: 0.0734
Epoch 160/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9638 -
accuracy: 0.0734
Epoch 161/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9623 -
accuracy: 0.0734
Epoch 162/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9608 -
accuracy: 0.0734
Epoch 163/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9593 -
accuracy: 0.0734
Epoch 164/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9579 -
accuracy: 0.0734
Epoch 165/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9565 -
accuracy: 0.0734
Epoch 166/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9551 -
accuracy: 0.0734
Epoch 167/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9538 -
accuracy: 0.0734
Epoch 168/200
126/126 [==============================] - 4s 34ms/step - loss: 5.9525 -
accuracy: 0.0734
Epoch 169/200
126/126 [==============================] - 4s 34ms/step - loss: 5.9512 -
accuracy: 0.0734
Epoch 170/200
126/126 [==============================] - 4s 34ms/step - loss: 5.9499 -
accuracy: 0.0734
Epoch 171/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9487 -
accuracy: 0.0734
```

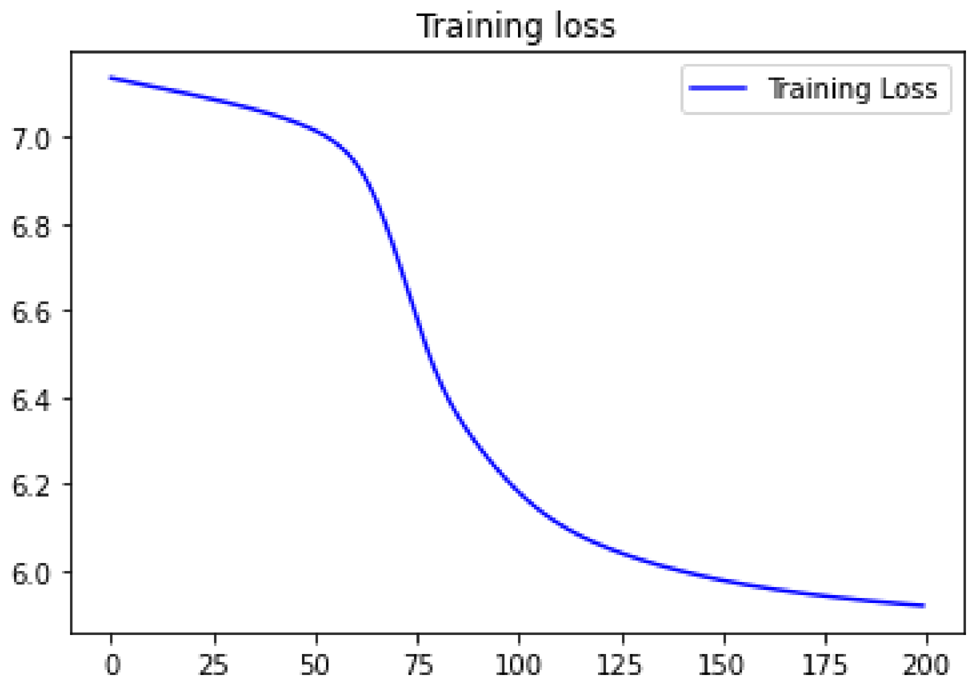
```
Epoch 172/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9474 -
accuracy: 0.0734
Epoch 173/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9462 -
accuracy: 0.0734
Epoch 174/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9451 -
accuracy: 0.0734
Epoch 175/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9439 -
accuracy: 0.0734
Epoch 176/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9428 -
accuracy: 0.0734
Epoch 177/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9417 -
accuracy: 0.0734
Epoch 178/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9406 -
accuracy: 0.0734
Epoch 179/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9395 -
accuracy: 0.0734
Epoch 180/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9385 -
accuracy: 0.0734
Epoch 181/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9374 -
accuracy: 0.0734
Epoch 182/200
126/126 [==============================] - 4s 34ms/step - loss: 5.9364 -
accuracy: 0.0734
Epoch 183/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9354 -
accuracy: 0.0734
Epoch 184/200
126/126 [==============================] - 4s 34ms/step - loss: 5.9344 -
accuracy: 0.0734
Epoch 185/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9335 -
accuracy: 0.0734
Epoch 186/200
126/126 [==============================] - 4s 34ms/step - loss: 5.9325 -
accuracy: 0.0734
Epoch 187/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9316 -
accuracy: 0.0734
```

```
Epoch 188/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9307 -
accuracy: 0.0734
Epoch 189/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9298 -
accuracy: 0.0734
Epoch 190/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9289 -
accuracy: 0.0734
Epoch 191/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9280 -
accuracy: 0.0734
Epoch 192/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9272 -
accuracy: 0.0734
Epoch 193/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9263 -
accuracy: 0.0734
Epoch 194/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9255 -
accuracy: 0.0734
Epoch 195/200
126/126 [==============================] - 4s 32ms/step - loss: 5.9247 -
accuracy: 0.0734
Epoch 196/200
126/126 [==============================] - 4s 32ms/step - loss: 5.9239 -
accuracy: 0.0734
Epoch 197/200
126/126 [==============================] - 4s 32ms/step - loss: 5.9231 -
accuracy: 0.0734
Epoch 198/200
126/126 [==============================] - 4s 32ms/step - loss: 5.9223 -
accuracy: 0.0734
Epoch 199/200
126/126 [==============================] - 4s 33ms/step - loss: 5.9216 -
accuracy: 0.0734
Epoch 200/200
126/126 [==============================] - 4s 34ms/step - loss: 5.9208 -
accuracy: 0.0734
```

Let's plot details of our training.

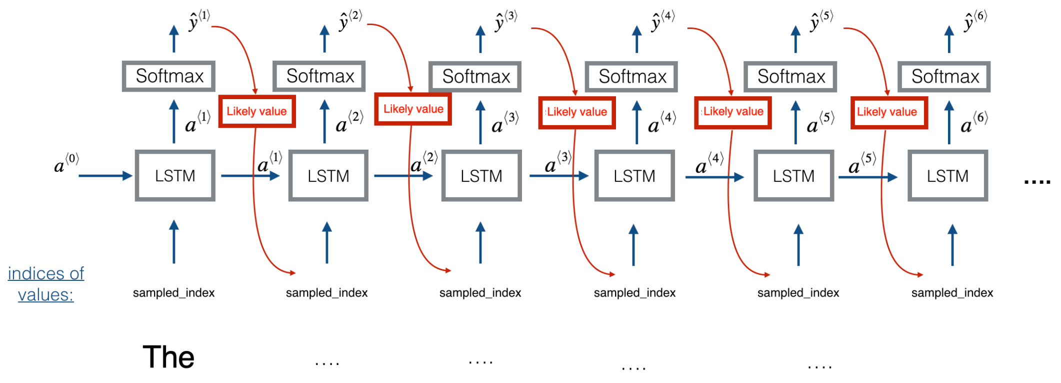
[14]:
```python
acc = history.history['accuracy']
loss = history.history['loss']
epochs = range(len(acc))
plt.plot(epochs, acc, 'b', label='Training accuracy')
plt.title('Training accuracy')
plt.figure()
plt.plot(epochs, loss, 'b', label='Training Loss')
plt.title('Training loss')
plt.legend()
plt.show()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





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()`.

---

[25]:
```
#TASK 2
#predict_classes is no longer available in tensorflow, so using predict
# Implement the generate() function

def generate(seed_text):
    ### START CODE HERE ###
    while True:
        tokens = tokenizer.texts_to_sequences([seed_text])
        if not tokens[0]:
            return seed_text
        predicted = model.predict(tokens)
        predicted_index = np.argmax(predicted[0]) + 1

        print("Pred index: ", predicted_index)
        next_word = ""
        for key, val in tokenizer.word_index.items():
            if val == predicted_index:
                next_word = key
                return seed_text
        seed_text = seed_text + " " + next_word
        print("Pred word: ", text)
        if next_word == '.':
            return seed_text
    return seed_text
    ### END CODE HERE ###
```

[26]:
```
predicted = model.predict(tokenizer.texts_to_sequences(["The"]))
print(predicted)
```

```
1/1 [==============================] - 0s 41ms/step
[[0.00077254 0.00181123 0.0015084  … 0.0007949  0.00078358 0.00078022]]
```

Let's test it:

[27]:
```
print(generate("COVID19 virus"))
print(generate("COVID19 is the"))
print(generate("The usa is"))
print(generate("The new virus"))
print(generate("China has"))
```

```
1/1 [==============================] - 1s 503ms/step
Pred index:  2
COVID19 virus
1/1 [==============================] - 0s 39ms/step
Pred index:  2
COVID19 is the
1/1 [==============================] - 0s 39ms/step
Pred index:  2
The usa is
1/1 [==============================] - 0s 38ms/step
Pred index:  2
The new virus
1/1 [==============================] - 0s 40ms/step
Pred index:  2
China has
```

Let's test it in an interactive mode:

[31]:
```
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: Is this
```

```
1/1 [==============================] - 0s 40ms/step
Pred index:  2
Is this
```

[32]:
```
#TASK 3
# Implement the generate_sample() function
def generate_sample(seed_text):
    ### START CODE HERE ###
    pass
     ### END CODE HERE ###
```

Let's test it in an interactive mode:

[33]:
```
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:
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-33-793b29d450e9> in <module>
      1 usr_input = input("Write the beginning of your tweet, the algorithm␣
 ↪machine will complete it. Your input is: ")
----> 2 for w in generate_sample(usr_input).split():
      3     print(w, end =" ")
      4     time.sleep(0.4)

AttributeError: 'NoneType' object has no attribute 'split'
```

3 Generate your own text

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

[ ]:
```

```

Export your notebook to a pdf document

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

4 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!