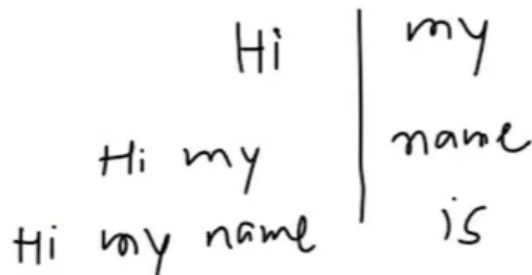


LSTM- Next Word Predictor

Dataset: FAQ

```
faqs = ""About the Program
What is the course fee for Data Science Mentorship Program (DSMP 2023)
The course follows a monthly subscription model where you have to make monthly payments of Rs 799/month.
What is the total duration of the course?
The total duration of the course is 7 months. So the total course fee becomes 799*7 = Rs 5600(approx.)
What is the syllabus of the mentorship program?
We will be covering the following modules:
Python Fundamentals
Python libraries for Data Science
Data Analysis
SQL for Data Science
Maths for Machine Learning
ML Algorithms
Practical ML
MLOPs
Case studies
You can check the detailed syllabus here - https://learnwith.campusx.in/courses/CampusX-Data-Science-Mentors
Will Deep Learning and NLP be a part of this program?
No, NLP and Deep Learning both are not a part of this program's curriculum.
What if I miss a live session? Will I get a recording of the session?
Yes all our sessions are recorded, so even if you miss a session you can go back and watch the recording.
Where can I find the class schedule?
Checkout this google sheet to see month by month time table of the course - https://docs.google.com/spreadsh
What is the time duration of all the live sessions?
```

Expected results:



Handwritten examples of next word prediction for the phrase "Hi my name is":

Hi	my
Hi my	name
Hi my name	is

```
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
```

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts([faqs])
```

- The input is in list because you can send multiple texts

```
len(tokenizer.word_index)
282
```

- There are 282 unique words.

Fetch sentences:

```
faqs.split('\n')
```

```
['About the Program',
 'What is the course fee for Data Science Mentorship Program (DSMP 2023)',
 'The course follows a monthly subscription model where you have to make monthly payments of Rs 799/month.',
 'What is the total duration of the course?',
 'The total duration of the course is 7 months. So the total course fee becomes 799*7 = Rs 5600(approx.)',
 'What is the syllabus of the mentorship program?',
 'We will be covering the following modules:',
 'Python Fundamentals',
 'Python libraries for Data Science',
 'Data Analysis',
 'SQL for Data Science',
 'Maths for Machine Learning',
```

- This will split the data on line break

Use `tokenizer.texts_to_sequences` to convert the sentences into numbers.

```
for sentence in faqs.split('\n'):
    print(tokenizer.texts_to_sequences([sentence]))
```

```

[[93, 1, 13]]
[[11, 7, 1, 12, 42, 15, 43, 53, 68, 13, 147, 148]]
[[1, 12, 94, 5, 44, 29, 95, 30, 2, 8, 4, 26, 44, 69, 6, 31, 70, 45]]
[[11, 7, 1, 71, 72, 6, 1, 12]]
[[1, 71, 72, 6, 1, 12, 7, 54, 73, 32, 1, 71, 12, 42, 149, 70, 54, 31, 55, 150]]
[[11, 7, 1, 96, 6, 1, 68, 13]]
[[27, 9, 19, 151, 1, 152, 153]]
[[97, 154]]
[[97, 155, 15, 43, 53]]
[[43, 156]]

```

to convert the `[[[]]]` → `[]`, we have to use `[0]`

```

for sentence in faqs.split('\n'):
    print(tokenizer.texts_to_sequences([sentence])[0])

```

```

[93, 1, 13]
[11, 7, 1, 12, 42, 15, 43, 53, 68, 13, 147, 148]
[1, 12, 94, 5, 44, 29, 95, 30, 2, 8, 4, 26, 44, 69, 6, 31, 70, 45]
[11, 7, 1, 71, 72, 6, 1, 12]
[1, 71, 72, 6, 1, 12, 7, 54, 73, 32, 1, 71, 12, 42, 149, 70, 54, 31, 55, 150]
[11, 7, 1, 96, 6, 1, 68, 13]
[27, 9, 19, 151, 1, 152, 153]
[97, 154]

```

We want:

- **Input:** 93, **Output:** 1
- **Input:** 93,1 **Output:** 13
- We will take `i+1`th item and print
 - We need to run a 2nd loop

```

input_sequences = []
for sentence in faqs.split('\n'):
    tokenized_sentence = tokenizer.texts_to_sequences([sentence])[0]

```

```
for i in range(1,len(tokenized_sentence)):
    input_sequences.append(tokenized_sentence[:i+1])
```

input_sequences

```
[[93, 1],
 [93, 1, 13],
 [11, 7],
 [11, 7, 1],
 [11, 7, 1, 12],
 [11, 7, 1, 12, 42],
 [11, 7, 1, 12, 42, 15],
 [11, 7, 1, 12, 42, 15, 43],
 [11, 7, 1, 12, 42, 15, 43, 53],
 [11, 7, 1, 12, 42, 15, 43, 53, 68],
 [11, 7, 1, 12, 42, 15, 43, 53, 68, 13],
 [11, 7, 1, 12, 42, 15, 43, 53, 68, 13, 147],
 [11, 7, 1, 12, 42, 15, 43, 53, 68, 13, 147, 148],
 [1, 12],
 [1, 12, 94],
 [1, 12, 94, 5],
 [1, 12, 94, 5, 44],
 [1, 12, 94, 5, 44, 29],
 [1, 12, 94, 5, 44, 29, 95],
 [1, 12, 94, 5, 44, 29, 95, 30],
 [1, 12, 94, 5, 44, 29, 95, 30, 2],
 [1, 12, 94, 5, 44, 29, 95, 30, 2, 8],
 [1, 12, 94, 5, 44, 29, 95, 30, 2, 8, 4],
 [1, 12, 94, 5, 44, 29, 95, 30, 2, 8, 4, 26],
 [1, 12, 94, 5, 44, 29, 95, 30, 2, 8, 4, 26, 44],
```

- 🙌 It's a **2D list**

Apply Zero Padding:

Find out sentence with max length:

```
max_len = max([len(x) for x in input_sequences])
max_len
```

57

- Longest sentence has 57 words in it.

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
padded_input_sequences = pad_sequences(input_sequences, maxlen = max_l  
en, padding='pre')
```

```
padded_input_sequences
```

OR

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
padded_input_sequences = pad_sequences(input_sequences)
```

```
padded_input_sequences
```

```
array([[ 0,  0,  0, ...,  0, 93,  1],  
       [ 0,  0,  0, ..., 93,  1, 13],  
       [ 0,  0,  0, ...,  0, 11,  7],  
       ...,  
       [ 0,  0,  0, ..., 279, 18, 280],  
       [ 0,  0,  0, ..., 18, 280, 281],  
       [ 0,  0,  0, ..., 280, 281, 282]])
```

- `padding='pre'` because we want the last number in the list as an output. (**Default**)
- If you don't specify the `maxlen` parameter in `pad_sequences`, the function will automatically determine the maximum length among all the sequences in your `input_sequences` list and use that length as the target length for padding.

Separate Input & Output:

```
X = padded_input_sequences[:, :-1]
```

- `:` → All Rows
- `:-1` → All columns except the last one

```
y = padded_input_sequences[:, -1]
```

- `:` → All Rows
- `-1` → Only the last column

```
array([[ 0,  0,  0, ...,  0, 93,  1],
       [ 0,  0,  0, ..., 93,  1, 13],
       [ 0,  0,  0, ...,  0, 11,  7],
       ...,
       [ 0,  0,  0, ..., 279, 18, 280],
       [ 0,  0,  0, ..., 18, 280, 281],
       [ 0,  0,  0, ..., 280, 281, 282]])
```

```
padded_input_sequences[:, -1]
```

✓ 0.0s

```
array([ 1, 13,  7,  1, 12, 42, 15, 43, 53, 68, 13, 147, 148,
        12, 94,  5, 44, 29, 95, 30,  2,  8,  4, 26, 44, 69,
         6, 31, 70, 45,  7,  1, 71, 72,  6,  1, 12, 71, 72,
```

X.shape

(863, 56)

Strategy:



Use multi-class classification.

- Replace the output with their **one hot encoded** vector.
 - `from tensorflow.keras.utils import to_categorical`

- Function → Softmax

$$\begin{bmatrix} 0.62 & 0.51 & 0.31 \end{bmatrix}$$

- You'll get a vector whose length is equal to the length of output vector

```
from tensorflow.keras.utils import to_categorical  
y = to_categorical(y, num_classes=283)
```

282 → No. of categories (num_classes= 282+1 because the output starts with 1)

- Keras auto detects it

```
len(tokenizer.word_index)
```

282



Using the default (num_classes=None) is convenient if you are sure that your input data y contains at least one example of every class label from 0 up to the maximum class label you expect.

However, if a class label is missing from your input

y, to_categorical will not create a column for it when num_classes is None .

Specifying num_classes explicitly is safer if you know the total number of classes beforehand, even if some classes might not be present in the specific y you are converting.

```
y.shape
```

```
(863, 283)
```

Now build the LSTM Architecture:

- **3 Layers:**

- **Layer 1:** Embedding Layer → Gives Dense vector for a word
- **Layer 2:** LSTM
- **Layer 3:** Dense layer for output → Softmax

```
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Embedding, LSTM, Dense
```

```
model = Sequential()  
model.add(Embedding(283, 100))  
model.add(LSTM(150))  
model.add(Dense(283, activation='softmax'))
```

`input_dim = 283` (vocabulary size)

`output_dim = 100` : This sets the size of the **embedding vectors**. Each integer token will be represented by a vector of size 100.

- Each word is represented by a 100D vector



Compile:

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Fit:

```
model.fit(X,y,epochs=100, validation_split= 0.2)
```

```
model.fit(X,y,epochs=100, validation_split= 0.2)
✓ 1m 7.1s
Epoch 1/100
22/22 ————— 1s 41ms/step - accuracy: 0.9560 - loss: 0.1230 - val_accuracy: 0.9480 - val_loss: 0.1518
Epoch 2/100
22/22 ————— 1s 29ms/step - accuracy: 0.9573 - loss: 0.1241 - val_accuracy: 0.9480 - val_loss: 0.1625
Epoch 3/100
22/22 ————— 1s 29ms/step - accuracy: 0.9436 - loss: 0.1335 - val_accuracy: 0.9364 - val_loss: 0.1788
Epoch 4/100
22/22 ————— 1s 28ms/step - accuracy: 0.9456 - loss: 0.1299 - val_accuracy: 0.9249 - val_loss: 0.1961
Epoch 5/100
22/22 ————— 1s 32ms/step - accuracy: 0.9502 - loss: 0.1212 - val_accuracy: 0.9364 - val_loss: 0.1973
Epoch 6/100
22/22 ————— 1s 30ms/step - accuracy: 0.9447 - loss: 0.1210 - val_accuracy: 0.9422 - val_loss: 0.2012
Epoch 7/100
22/22 ————— 1s 30ms/step - accuracy: 0.9671 - loss: 0.1042 - val_accuracy: 0.9306 - val_loss: 0.2120
Epoch 8/100
22/22 ————— 1s 29ms/step - accuracy: 0.9579 - loss: 0.1109 - val_accuracy: 0.9364 - val_loss: 0.2176
Epoch 9/100
22/22 ————— 1s 28ms/step - accuracy: 0.9523 - loss: 0.1180 - val_accuracy: 0.9306 - val_loss: 0.2189
Epoch 10/100
22/22 ————— 1s 28ms/step - accuracy: 0.9500 - loss: 0.1031 - val_accuracy: 0.9364 - val_loss: 0.2210
Epoch 11/100
22/22 ————— 1s 32ms/step - accuracy: 0.9471 - loss: 0.1248 - val_accuracy: 0.9249 - val_loss: 0.2342
Epoch 12/100
22/22 ————— 1s 28ms/step - accuracy: 0.9643 - loss: 0.1001 - val_accuracy: 0.9364 - val_loss: 0.2336
Epoch 13/100
...
Epoch 99/100
22/22 ————— 1s 28ms/step - accuracy: 0.9665 - loss: 0.0664 - val_accuracy: 0.9017 - val_loss: 0.4042
Epoch 100/100
22/22 ————— 1s 28ms/step - accuracy: 0.9411 - loss: 0.0998 - val_accuracy: 0.9191 - val_loss: 0.3978
```

Prediction:

For prediction, we have to:

1. Tokenize
2. Pad
3. Predict

```
import time
import numpy as np

text = "mail"

for i in range(10):
    # tokenize
    token_text = tokenizer.texts_to_sequences([text])[0]
    # padding
    padded_token_text = pad_sequences([token_text], maxlen=56)
    # predict
    pos = np.argmax(model.predict(padded_token_text))
```



!! If you don't specify `maxlen=56` , it will give output like `→ [[61]]`.
However, we want Output like 👉

```
array([[ 0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0, 61]])
```

- 61 is at last position as the input was only 1 word
- `model.predict(padded_token_text)` will give a 283D vector

```
# tokenize
token_text = tokenizer.texts_to_sequences([text])[0]
# padding
padded_token_text = pad_sequences([token_text], maxlen=56, padding='pre')
# predict
model.predict(padded_token_text)

1.61945754e-05, 1.44023536e-04, 2.54808296e-03, 1.36482544e-04,
4.25928738e-04, 1.76832782e-05, 8.18018627e-04, 2.44576880e-03,
1.91404342e-05, 6.56942371e-04, 5.41475893e-04, 1.26519357e-03,
4.34428863e-02, 9.50626563e-03, 1.52308028e-04, 1.93403994e-05,
8.48380150e-04, 1.49693398e-03, 6.41837250e-05, 5.83399960e-04,
4.38619056e-04, 6.35089062e-04, 3.16095888e-04, 3.79362100e-05,
3.27751652e-04, 6.31779840e-05, 9.14293632e-05, 2.36496217e-05,
6.23497064e-04, 1.46202865e-05, 9.22770869e-06, 4.35694074e-06,
1.16519135e-04, 5.29432716e-03, 5.95367892e-05, 5.33962293e-07,
```

- We want to find out the position of highest probability in this → `np.argmax()`

```
text = "mail"

# tokenize
token_text = tokenizer.texts_to_sequences([text])[0]
# padding
padded_token_text = pad_sequences([token_text], maxlen=56, padding='pre')
# predict
np.argmax(model.predict(padded_token_text))

1/1 [=====] - 0s 40ms/step
49
```

- Word at position 49 has the highest probability
- Now, we want to find out the word at 49th position

```
tokenizer.word_index
```

```
{'the': 1,  
'you': 2,  
'i': 3,  
'to': 4,  
'a': 5,  
'of': 6,  
'is': 7,
```

- This gives us the dictionary

```
tokenizer.word_index.items()
```

This is a **Python dictionary method** that gives you the `(key, value)` pairs — so you can loop through them.

```
tokenizer.word_index.items()
```

```
dict_items([('the', 1), ('you', 2), ('i', 3), ('to', 4), ('a', 5), ('of', 6)
```

```
for word,index in tokenizer.word_index.items():  
    if index == pos:  
  
        print(word)
```

```
1/1 ————— 0s 41ms/step  
us
```

Predict next 5 words:

```
import numpy as np  
  
text = "nlp and deep learning"
```

```

for i in range(5):
    # tokenize
    token_text = tokenizer.texts_to_sequences([text])[0]
    # padding
    padded_token_text = pad_sequences([token_text], maxlen=56, padding='pre')
    # predict
    pos = np.argmax(model.predict(padded_token_text))

    for word,index in tokenizer.word_index.items():
        if index == pos:
            text = text + " " + word
            print(text)

```

```

1/1 ————— 0s 40ms/step
nlp and deep learning both
1/1 ————— 0s 49ms/step
nlp and deep learning both are
1/1 ————— 0s 40ms/step
nlp and deep learning both are not
1/1 ————— 0s 39ms/step
nlp and deep learning both are not a
1/1 ————— 0s 43ms/step
nlp and deep learning both are not a part

```