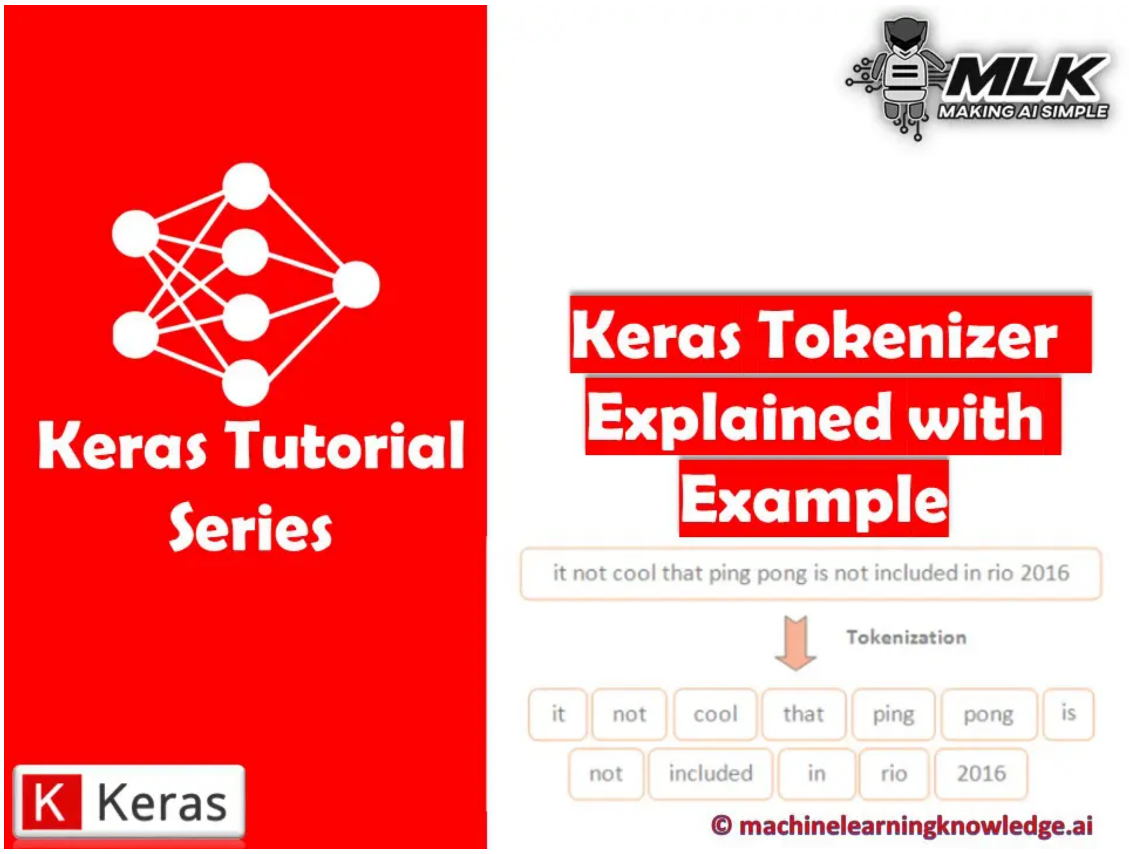
Keras Tokenizer Tutorial with Examples for Beginners

By **Palash Sharma** - January 1, 2021


Keras Tutorial Series
Keras
MLK MAKING AI SIMPLE
Keras Tokenizer Explained with Example
it not cool that ping pong is not included in rio 2016
Tokenization
it
not
cool
that
ping
pong
is
not
included
in
rio
2016
© machinelearningknowledge.ai

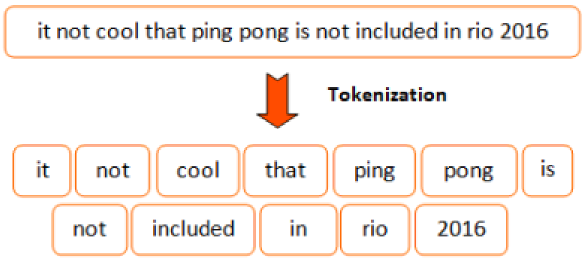
Contents [hide]

X

Introduction

In this article, we will go through the tutorial of Keras Tokenizer API for dealing with natural language processing (NLP). We will first understand the concept of tokenization in NLP and see different types of Keras tokenizer functions – fit_on_texts, texts_to_sequences, texts_to_matrix, sequences_to_matrix with examples.

What does Tokenization mean?



(Source)

Tokenization is a method to segregate a particular text into small chunks or tokens. Here the tokens or chunks can be anything from words to characters, even subwords.

Tokenization is divided into 3 major types-

1. Word Tokenization
2. Character Tokenization
3. Subword tokenization

Let us understand this concept of **word tokenization** with the help of an example sentence – *"We will win"*.

Usually, word tokenization is performed by using space acts as a delimiter. So in our example, we obtain three word tokens from the above sentence, i.e. **We-will-win**.

Just like the above example, if we have a word say **Relaxing**. Then the character tokens and subword tokens are shown below:

Character Tokens : R-e-l-a-x-i-n-g

Subword Tokens: Relax-ing

I hope, this section explains the basic concept of tokenization, let us now go into details about Keras Tokenizer Class.

Keras Tokenizer Class

The Tokenizer class of Keras is used for vectorizing a text corpus. For this either, each text input is converted into integer sequence or a vector that has a coefficient for each token in the form of binary values.

Keras Tokenizer Syntax

The below syntax shows the Keras "Tokenizer" function, along with all the parameters that are used in the function for various purposes.

```
 tf.keras.preprocessing.text.Tokenizer(
num_words=None,
filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n',
lower=True,
document_count=0,
**kwargs)
```

Methods of Keras Tokenizer Class

We will discuss the following methods of Keras Tokenizer class in the article –

1. fit_on_texts

2. texts_to_sequences
3. texts_to_matrix
4. sequences_to_matrix

1. fit_on_texts

The **fit_on_texts** method is a part of Keras tokenizer class which is used to update the internal vocabulary for the texts list. We need to call be before using other methods of texts_to_sequences or texts_to_matrix.

The object returned by **fit_on_texts** can be used to derive more information by using the following attributes-

- **word_counts** : It is a dictionary of words along with the counts.
- **word_docs** : Again a dictionary of words, this tells us how many documents contain this word
- **word_index** : In this dictionary, we have unique integers assigned to each word.
- **document_count** : This integer count will tell us the total number of documents used for fitting the tokenizer.

Example 1 : fit_on_texts on Document List

In [1]:

X

```
from keras.preprocessing.text import Tokenizer

t  = Tokenizer()
# Defining 4 document lists
fit_text = ['Machine Learning Knowledge',
            'Machine Learning',
            'Deep Learning',
            'Artificial Intelligence']

t.fit_on_texts(fit_text)
```

The **document_count** prints the **number of documents** present in our corpus. In our above example, there are 4 documents present.

In [2]:

```
print("The document count",t.document_count)
```

```
The document count 4
```

The **word_count** shows the number of times words occur in the text corpus passed to the Keras tokenizer class model. In our example, the word ‘machine’ has occurred 2 times, ‘learning’ 3 times, and so on.

In [3]:

```
print("The count of words",t.word_counts)
```

```
The count of words OrderedDict([('machine', 2), ('learning', 3), ('knowledge', 1), ('deep',
```

The **word_index** assigns a unique index to each word present in the text. This unique integer helps the model during training purposes.

In [4]:

```
print("The word index",t.word_index)
```

```
learning': 1, 'machine': 2, 'knowledge': 3, 'deep': 4, 'artificial': 5, 'intelligence': 6}
```

The **word_doc** tells in how many documents each of the words appear. In our example, ‘machine’

```
print("The word docs",t.word_docs)


The word docs defaultdict(<class 'int'>, {'knowledge': 1, 'learning': 3, 'machine': 2, 'de
```

Example 2 : fit_on_texts on String

fit_on_texts, when applied on a string text, its attributes produce different types of results.

- The **word_count** shows the number of times a character has occurred.
- The **document_count** prints the number of characters present in our input text.
- The **word_index** assigns a unique index to each character present in the text.
- The **word_docs** produces results similar to word_counts and gives the frequency of characters.

[6]:

```
t  = Tokenizer()

fit_text = 'Machine Learning'

t.fit_on_texts(fit_text)

print("Count of characters:",t.word_counts)
print("Length of text:",t.document_count)
print("Character index",t.word_index)
print("Frequency of characters:",t.word_docs)


Count of characters: OrderedDict([('m', 1), ('a', 2), ('c', 1), ('h', 1), ('i', 2), ('n',
Length of text: 16
Character index {'n': 1, 'a': 2, 'i': 3, 'e': 4, 'm': 5, 'c': 6, 'h': 7, 'l': 8, 'r': 9, '
Frequency of characters: defaultdict(<class 'int'>, {'m': 1, 'a': 2, 'c': 1, 'h': 1, 'i':
```

2. texts_to_sequences

texts_to_sequences method helps in converting tokens of text corpus into a sequence of integers.

Example 1: texts_to_sequences on Document List

We can see here in the example that given a corpus of documents, texts_to_sequences assign integers to words. For example, 'machine' is assigned value 2.

In [7]:

```
t = Tokenizer()

test_text = ['Machine Learning Knowledge',
              'Machine Learning',
             'Deep Learning',
             'Artificial Intelligence']

t.fit_on_texts(test_text)

sequences = t.texts_to_sequences(test_text)

print("The sequences generated from text are : ",sequences)
```

```
The sequences generated from text are :  [[2, 1, 3], [2, 1], [4, 1], [5, 6]]
```

Example 2: texts_to_sequences on String

In this example, texts_to_sequences assign integers to characters. For example, 'e' is assigned a

X

```
t = Tokenizer()

test_text = "Machine Learning"

t.fit_on_texts(test_text)

sequences = t.texts_to_sequences(test_text)

print("The sequences generated from text are : ",sequences)
```

```
The sequences generated from text are :  [[5], [2], [6], [7], [3], [1], [4], [], [8], [4],
```

3. texts_to_matrix

Another useful method of tokenizer class is **texts_to_matrix()** function for converting the document into a numpy matrix form.

This function works in 4 different modes –

- binary : The default value that tells us about the presence of each word in a document.
- count : As the name suggests, the count for each word in the document is known.
- tfidf : The TF-IDF score for each word in the document.
- freq : The frequency tells us about ratio of words in each document.

Example 1: texts_to_matrix with mode = binary

The binary mode in **texts_to_matrix()** function determines the presence of text by using '1' in the matrix where the word is present and '0' where the word is not present.

NOTE: This mode doesn't count the total number of times a particular word or text, but it just tells about the presence of word in each of the documents.

In [9]:

```
                'Dazzling Deep Learning',
                'Champion Computer Vision',
                'Notorious Natural Language Processing Notorious Natural Language Processi
# create the tokenizer
t = Tokenizer()

t.fit_on_texts(docs)

encoded_docs = t.texts_to_matrix(docs, mode='binary')
print(encoded_docs)
```

```
[[0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1.]
 [0. 0. 0. 0. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

Example 2: texts_to_matrix with mode = count

The count mode in **texts_to_matrix()** function determines the number of times the words appear in each of the documents.

In [10]:

```
# define 5 documents
docs = ['Marvellous Machine Learning Marvellous Machine Learning',
'Amazing Artificial Intelligence',
'Dazzling Deep Learning',
'Champion Computer Vision',
'Notorious Natural Language Processing Notorious Natural Language Processing']

# create the tokenizer
t = Tokenizer()

t.fit_on_texts(docs)
```

```
print(encoded_docs)
```

```
[[0. 2. 2. 2. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1.]
 [0. 0. 0. 0. 2. 2. 2. 2. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

Example 3: texts_to_matrix with mode = tfidf

TF-IDF or Term Frequency – Inverse Document Frequency, works by checking the relevance of a word in a given text corpus.

In this mode, a **proportional score** is given to words on the basis of the number of times they occur in the text corpus. In this way, this model can determine which words are worthy and which aren’t.

In [12]:

```
t.fit_on_texts(docs)
encoded_docs = t.texts_to_matrix(docs, mode='tfidf')
print(encoded_docs)
```

```
[[0.         2.19987732 2.97631218 2.97631218 0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         1.75785792 1.75785792 1.75785792 0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         1.75785792 1.75785792 1.75785792]
 [0.         0.         0.         0.         1.29928298 1.29928298
  1.29928298 1.29928298 0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         1.29928298 1.29928298
  1.29928298 1.29928298 0.         0.         0.         0.
  0.         0.         0.         0.        ]]
```

Example 4: texts_to_matrix with mode = freq

This last mode used in texts_to_matrix() is the **frequency** that actually determines a score and assigning to each on the basis of the ratio of the word with all the words in the document or text corpus.

In [13]:

```
t.fit_on_texts(docs)

encoded_docs = t.texts_to_matrix(docs, mode='freq')

print(encoded_docs)
```

```
[[0.         0.33333333 0.33333333 0.33333333 0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.33333333 0.33333333 0.33333333 0.
  0.         0.         0.         0.        ]
 [0.         0.33333333 0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.33333333
  0.33333333 0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.33333333 0.33333333 0.33333333]
 [0.         0.         0.         0.         0.25       0.25
```

```
  0.25       0.25       0.         0.         0.         0.
  0.         0.         0.         0.        ]]
```

4. sequences_to_matrix

This function **sequences_to_matrix()** of Keras tokenizer class is used to convert the sequences into a numpy matrix form.

sequences_to_matrix also has 4 different modes to work with –

- binary : The default value that tells us about the presence of each word in a document.
- count : As the name suggests, the count for each word in the document is known.
- tfidf : The TF-IDF score for each word in the document.
- freq : The frequency tells us about ratio of words in each document.

Example 1: sequences_to_matrix with mode = binary

In [14]:

```
# define 4 documents
docs =['Machine Learning Knowledge',
       'Machine Learning and Deep Learning',
       'Deep Learning',
       'Artificial Intelligence']

# create the tokenizer
t = Tokenizer()

t.fit_on_texts(docs)

sequences = t.texts_to_sequences(docs)

encoded_docs = t.sequences_to_matrix(sequences, mode='binary')
print(encoded_docs)
```

```
 [0. 1. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 1.]]
```

Example 2: sequences_to_matrix with mode = count

In [15]:

```
# define 4 documents
docs =['Machine Learning Knowledge',
       'Machine Learning and Deep Learning',
      'Deep Learning',
      'Artificial Intelligence']

# create the tokenizer
t = Tokenizer()

t.fit_on_texts(docs)

sequences = t.texts_to_sequences(docs)

encoded_docs = t.sequences_to_matrix(sequences, mode='count')
print(encoded_docs)
```

```
[[0. 1. 1. 0. 1. 0. 0. 0.]
 [0. 2. 1. 1. 0. 1. 0. 0.]
 [0. 1. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 1.]]
```

Example 3: sequences_to_matrix with mode = tfidf

In [16]:

```
# define 4 documents
docs =['Machine Learning Knowledge',
      'Machine Learning and Deep Learning',
      'Deep Learning',
      'Artificial Intelligence']

# create the tokenizer
t = Tokenizer()

t.fit_on_texts(docs)

sequences = t.texts_to_sequences(docs)

encoded_docs = t.sequences_to_matrix(sequences, mode='tfidf')
```

```
[[0.         0.69314718 0.84729786 0.         1.09861229 0.
  0.         0.        ]
 [0.         1.17360019 0.84729786 0.84729786 0.         1.09861229
  0.         0.        ]
 [0.         0.69314718 0.         0.84729786 0.         0.
  0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  1.09861229 1.09861229]]
```

Example 4: sequences_to_matrix with mode = freq

In [17]:

```
# define 4 documents
docs =['Machine Learning Knowledge',
       'Machine Learning and Deep Learning',
       'Deep Learning',
       'Artificial Intelligence']

# create the tokenizer
t = Tokenizer()

t.fit_on_texts(docs)

sequences = t.texts_to_sequences(docs)

encoded_docs = t.sequences_to_matrix(sequences, mode='freq')
print(encoded_docs)
```

```
[[0.         0.33333333 0.33333333 0.         0.33333333 0.
```

```
 [0.         0.5        0.         0.5        0.         0.
  0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.5        0.5       ]]
```

Training Model using Keras Tokenizer sequences_to_matrix

We will now see how we can use the Keras tokenizer sequences_to_matrix while training a neural network model. We will use the binary mode of sequences_to_matrix in our example.

In [18]:

```
import keras
from keras.datasets import reuters
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from keras.preprocessing.text import Tokenizer
import tensorflow as tf
(X_train,y_train),(X_test,y_test) = reuters.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/reuters.
2113536/2110848 [==============================] - 0s 0us/step
```

Now that we have loaded the data, we need to explore it. For this we will be looking at the shape of training and testing data.

In [19]:

```
print(X_train.shape)

print(X_test.shape)
```

Output:

```
(8982,)
(2246,)
```

In addition to all of this, training and testing labels will be consisting of 46 different classes.
We will be using the default method i.e. **Binary Mode** for converting sequence to matrix**

Here we are using the **sequences_to_matrix()** function.

In [20]:

```
tokenizer = Tokenizer(num_words=50000)
X_train = tokenizer.sequences_to_matrix(X_train, mode='binary')
X_test = tokenizer.sequences_to_matrix(X_test, mode='binary')
y_train = keras.utils.to_categorical(y_train,num_classes=46)
y_test = keras.utils.to_categorical(y_test,num_classes=46)
```

Now we have completed the processing steps, it's time to build a neural network for the classification task. We'll be using the below code for defining the model.

In [21]:

```
model = Sequential()
model.add(tf.keras.layers.Dense(128,input_shape=(X_train[0].shape)))
model.add(tf.keras.layers.Dense(512, activation='relu'))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(512, activation='relu'))
model.add(Dropout(0.25))
model.add(tf.keras.layers.Dense(46, activation='softmax'))
```

The below code snippet will help us summarize the model.

In [21]:

```
print(model.summary())
```

Output:

```
Model: "sequential"
_________________________________________________________________
```

```
_________________________________________________________________
dense_1 (Dense)              (None, 512)               66048
_________________________________________________________________
batch_normalization (BatchNo (None, 512)               2048
_________________________________________________________________
dense_2 (Dense)              (None, 128)               65664
_________________________________________________________________
dense_3 (Dense)              (None, 512)               66048
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense_4 (Dense)              (None, 46)                23598
=================================================================
Total params: 6,623,534
Trainable params: 6,622,510
Non-trainable params: 1,024
_________________________________________________________________
None
```

This step will be to compile the model using stochastic gradient descent optimizer, cross-entropy loss, and accuracy as metrics for measuring performance.

The compilation will be followed by model training and also validate the data. The batch size used is 64 and 10 epochs will be used.

In [22]:

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

model.fit(X_train,y_train,validation_data=(X_test,y_test),batch_size=64,epochs=10)
```

Output:

```
Epoch 1/10
141/141 [==============================] - 8s 58ms/step - loss: 1.8339 - accuracy: 0.5949
Epoch 2/10
141/141 [==============================] - 8s 54ms/step - loss: 1.1011 - accuracy: 0.7440
Epoch 3/10
141/141 [==============================] - 8s 54ms/step - loss: 0.8321 - accuracy: 0.8087
Epoch 4/10
141/141 [==============================] - 8s 54ms/step - loss: 0.6463 - accuracy: 0.8516
Epoch 5/10
141/141 [==============================] - 8s 56ms/step - loss: 0.5211 - accuracy: 0.8810
Epoch 6/10
141/141 [==============================] - 8s 54ms/step - loss: 0.4254 - accuracy: 0.9016
Epoch 7/10
141/141 [==============================] - 8s 54ms/step - loss: 0.3660 - accuracy: 0.9137
Epoch 8/10
Epoch 10/10
141/141 [==============================] - 8s 55ms/step - loss: 0.2438 - accuracy: 0.9421
```

As we perform the evaluation, we can see that the accuracy achieved is 79%.

In [23]:

```
model.evaluate(X_test,y_test)
```

Output:

```
71/71 [==============================] - 1s 13ms/step - loss: 1.0121 - accuracy: 0.7876
```

```
[1.0120643377304077, 0.7876224517822266]
```

In the same manner, as shown above, we can use other methods for creating and training the model. You would only have to change the method from **binary to either count, tfidf, or freq**.

- **Also Read** – Different Types of Keras Layers Explained for Beginners
- **Also Read** – Keras Dropout Layer Explained for Beginners
- **Also Read** – Keras Dense Layer Explained for Beginners
- **Also Read** – Keras Convolution Layer – A Beginner's Guide
- **Also Read –** Beginners's Guide to Keras Models API
- **Also Read** – Types of Keras Loss Functions Explained for Beginners

Conclusion

This tutorial has come to an end, we looked at Keras tokenizer. We understood the concept of tokenization in NLP and see different types of Keras tokenizer functions – fit_on_texts, texts_to_sequences, texts_to_matrix, sequences_to_matrix with examples. At last, the tutorial ended with the training of a model using the binary mode of sequences to matrix function.

Reference Keras Documentation

- Term: Class
- Term: Artificial Neural Network
- Term: Classification

Palash Sharma

I am Palash Sharma, an undergraduate student who loves to explore and garner in-depth knowledge in the fields like Artificial Intelligence and Machine Learning. I am captivated by the wonders these fields have produced with their novel implementations. With this, I have a desire to share my knowledge with others in all my capacity.