DEEP LEARNING WITH PYTHON

- Book by François Chollet

CHAPTER 06

DEEP LEARNING FOR TEXT AND SEQUENCES

Yousra Mashkoor

OBJECTIVE:

Preprocessing text data into useful representations.

Working with recurrent neural network.

Using 1D convents for sequence processing

APPLICATION:

- Document classification (author , topic identification)
- Timeseries comparisons(level of similarities of documents)
- Sequence-to-sequence learning (English to French)
- Sentiment analysis (Positive/ negative)
- Timeseries forecasting (predicting weather)

WHAT IS TEXT DATA?

- Sequence of characters or sequence of words.
- NLP to text is what computer vision into pixels.
- Works only with numeric tensors.
- A technique called vectorizing text is used.
    - Can be done in multiple ways.

TECHNIQUES FOR VECTORIZING

01

Segment text into words and transform each word into a vector.

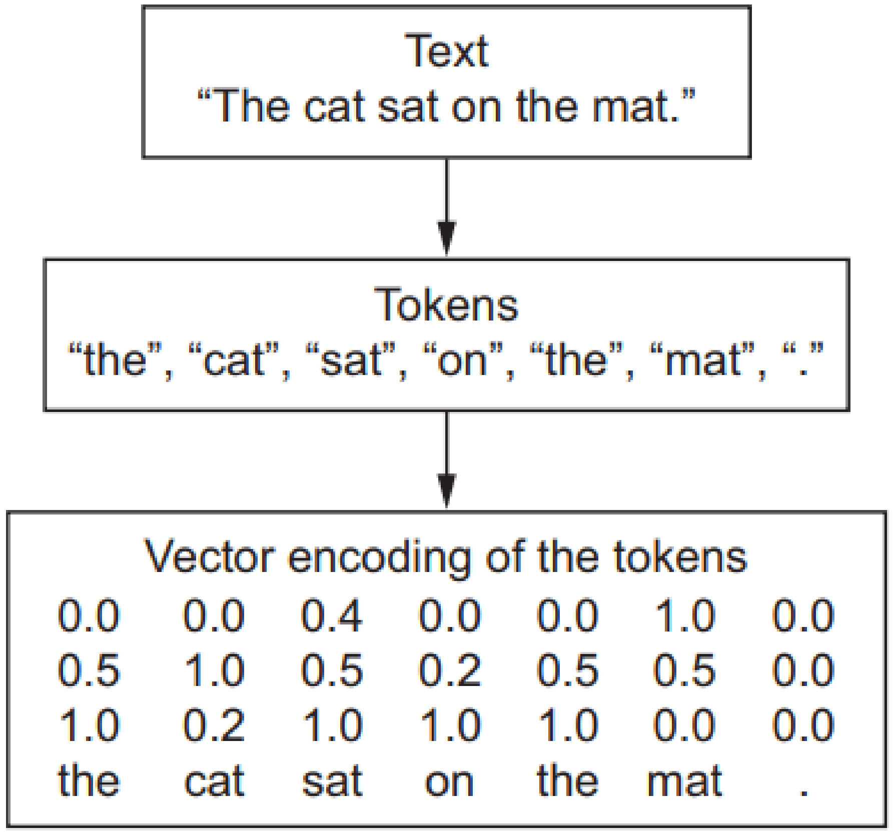
02

Segment text into characters and transform each character into a vector.

03

Extract n-grams of words or characters and transform each n-gram into a vector.

- N-grams are overlapping groups of multiple consecutive words or characters


Text
“The cat sat on the mat.”
Tokens
“the”, “cat”, “sat”, “on”, “the”, “mat”, “.”
Vector encoding of the tokens

0.0	0.0	0.4	0.0	0.0	1.0	0.0
0.5	1.0	0.5	0.2	0.5	0.5	0.0
1.0	0.2	1.0	1.0	1.0	0.0	0.0
the	cat	sat	on	the	mat	.

TOKENS AND TOKENIZATION

- Each level, words, characters, or n-grams are called tokens.
- Breaking text into tokens in called tokenization.
- Some Tokenization scheme is used to generate tokens.
- Text vectorization involves associating tokens with numeric tensors.
- Then fed into deep neural networks.

What is n-gram?

- Word n-grams are groups of N (or fewer) consecutive words that you can extract from a sentence.
- set of 2-grams:

{"The", "The cat", "cat", "cat sat", "sat", "sat on", "on", "on the", "the", "the mat", "mat"}

- set of 3-grams:

{"The", "The cat", "cat", "cat sat", "The cat sat", "sat", "sat on", "on", "cat sat on", "on the", "the", "sat on the", "the mat", "mat", "on the mat"}

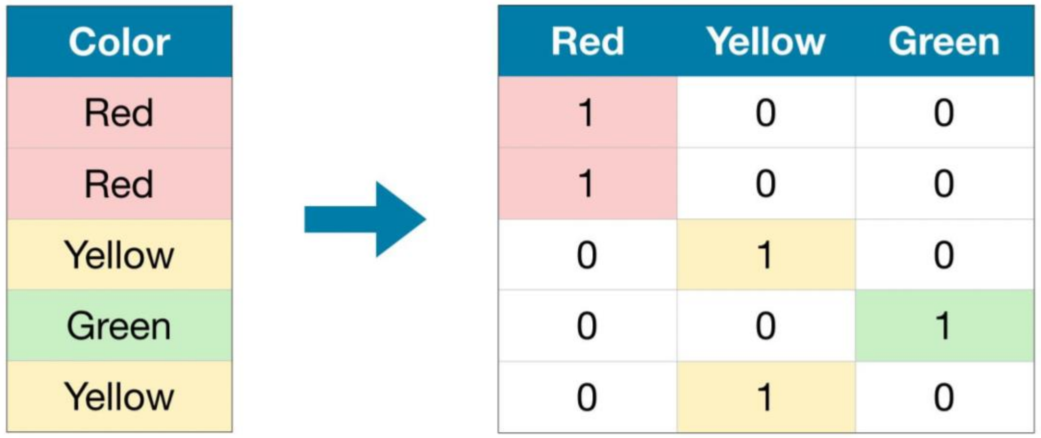
- Called a bag-of-2-grams or bag-of-3-grams, respectively
- Understood as set not sequence hence considered shallow language processing.
- Mostly use in feature engineering not in deep learning.
- Powerful when using lightweight, shallow text processing.

ONE HOT ENCODING

TOKEN EMBEDDING

2 Tokenization Schemes

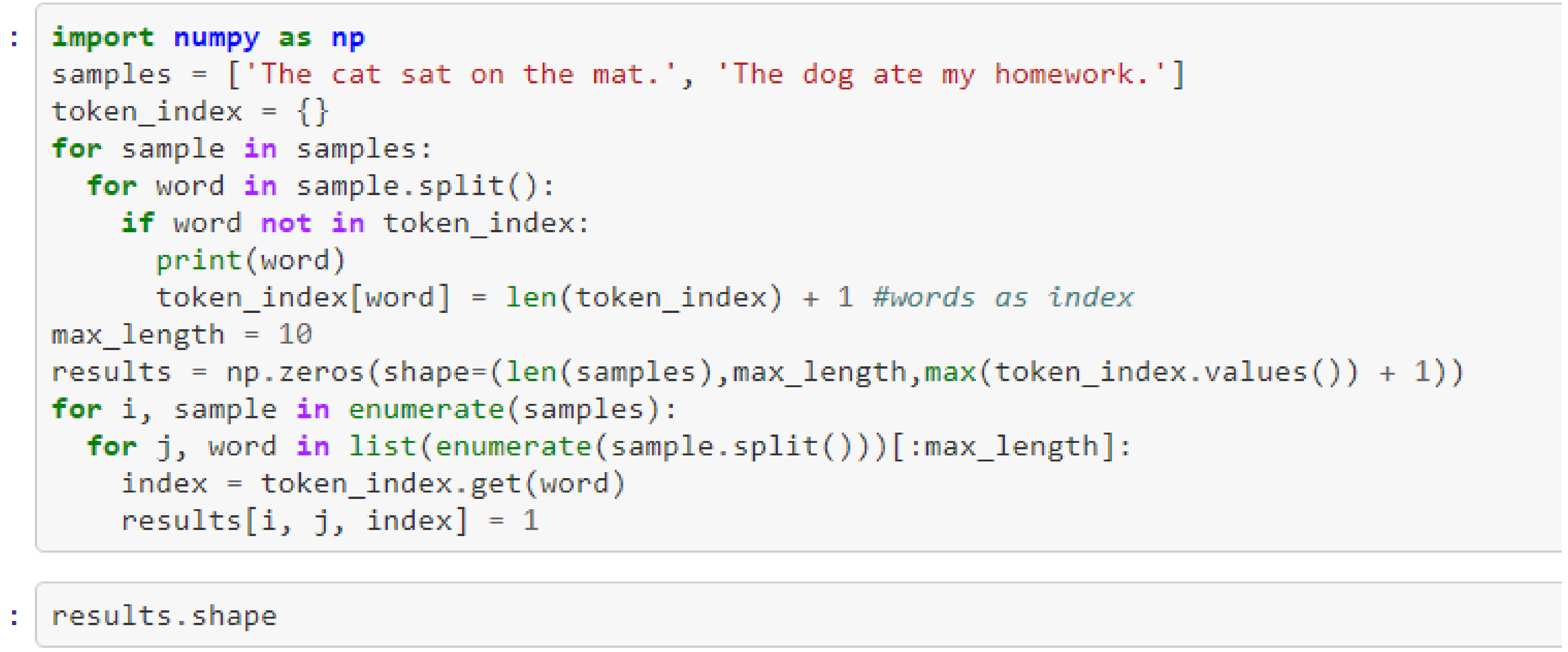
ONE HOT ENCODING:



Color
Red
Red
Yellow
Green
Yellow

Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1
0	1	0

- Assign a unique integer i to every word
- Create a binary vector of zeroes of size N (size of vocabulary)
- All vector is o except ith element

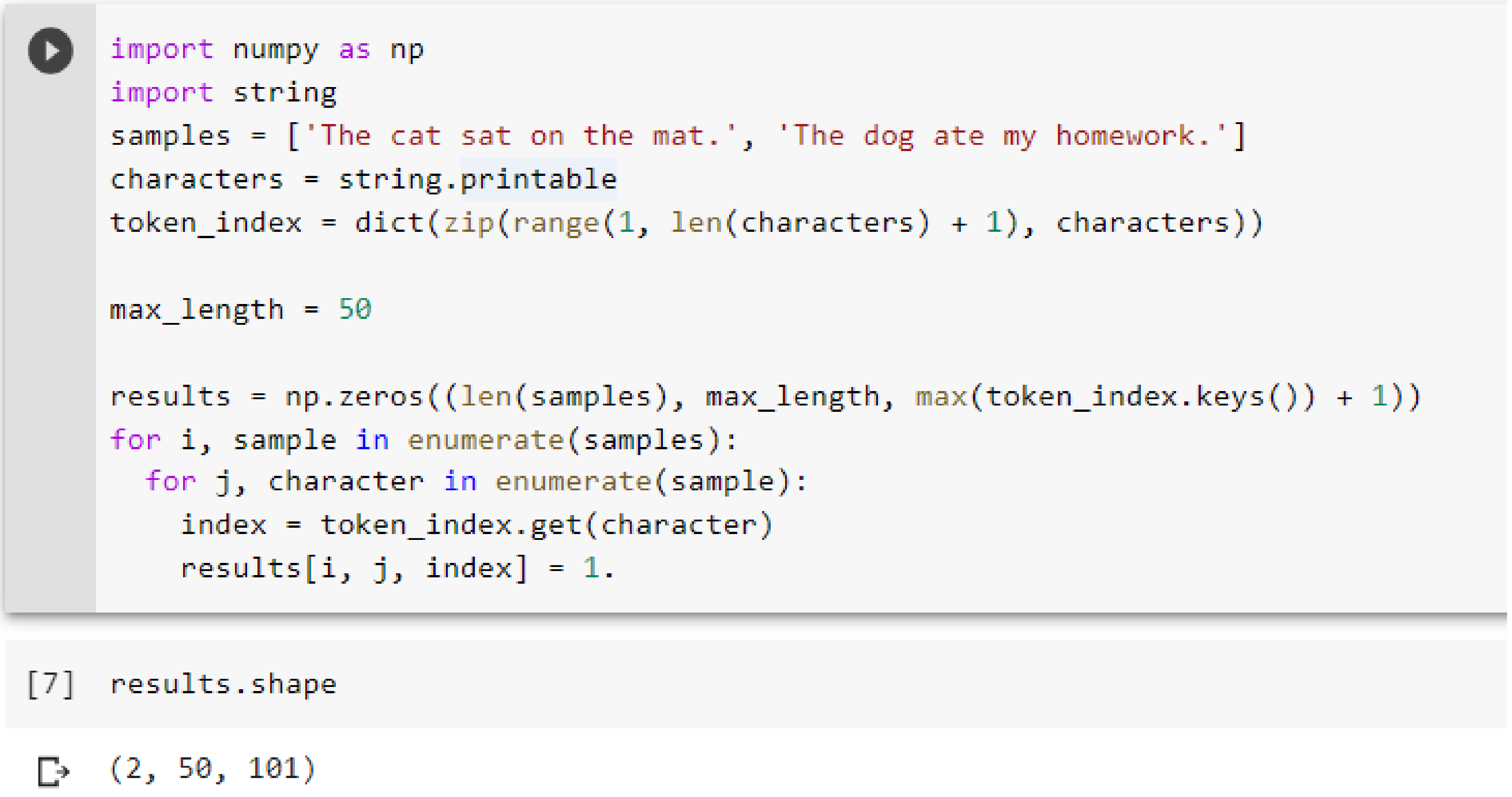


```
: import numpy as np
  samples = ['The cat sat on the mat.', 'The dog ate my homework.']
  token_index = {}
  for sample in samples:
    for word in sample.split():
      if word not in token_index:
        print(word)
        token_index[word] = len(token_index) + 1 #words as index
  max_length = 10
  results = np.zeros(shape=(len(samples),max_length,max(token_index.values()) + 1))
  for i, sample in enumerate(samples):
    for j, word in list(enumerate(sample.split()))[:max_length]:
      index = token_index.get(word)
      results[i, j, index] = 1
```

```
: results.shape
```

WORD-LEVEL ONE-HOT ENCODING

https://github.com/YousraMashkoor/Deep-Learning-by-francois-chollet-SOLVED/blob/master/Chapter%2006/Listing_6_1_Word_level_one_hot_encoding_(toy_example).ipynb



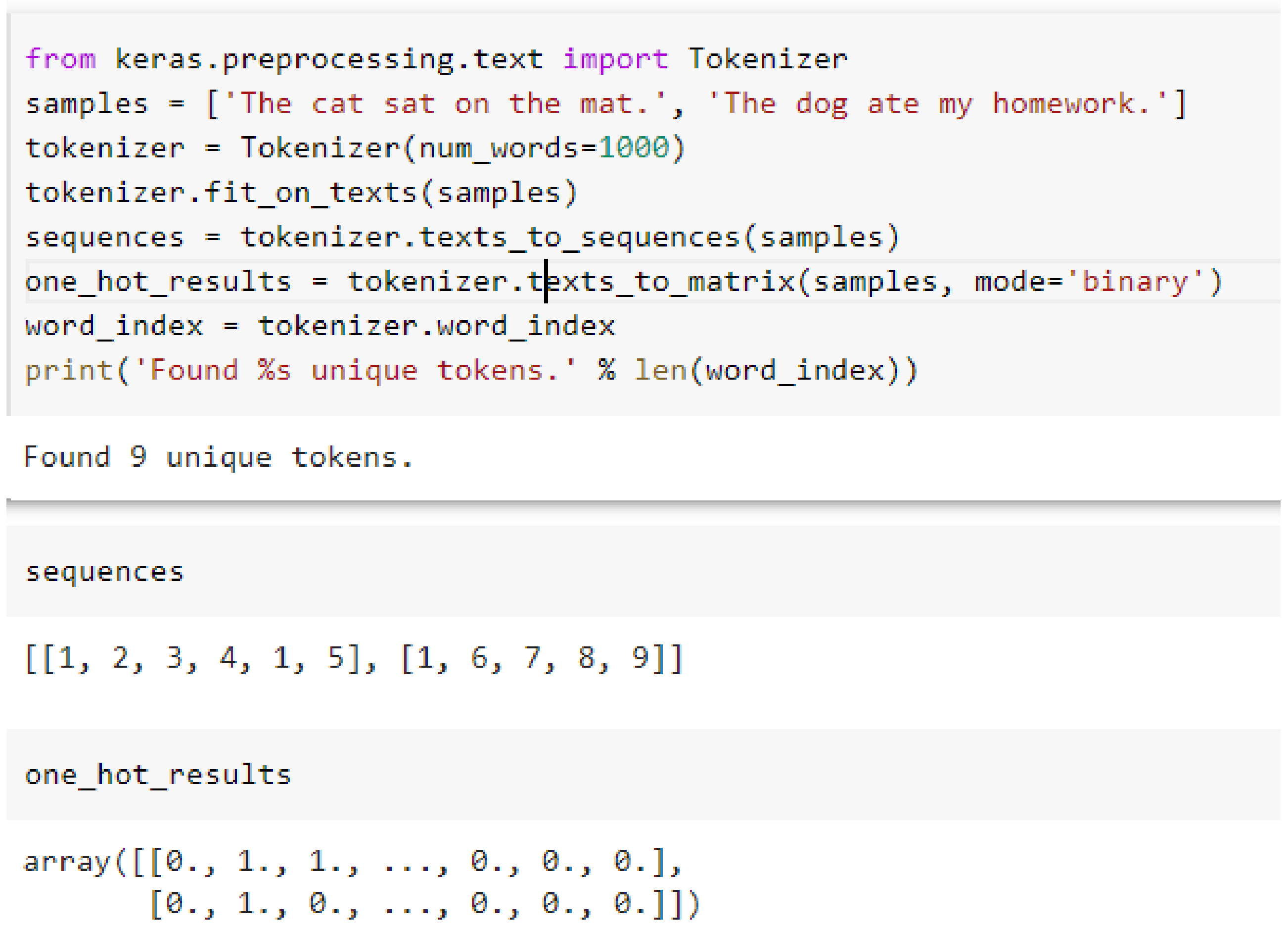
```
import numpy as np
import string
samples = ['The cat sat on the mat.', 'The dog ate my homework.']
characters = string.printable
token_index = dict(zip(range(1, len(characters) + 1), characters))

max_length = 50

results = np.zeros((len(samples), max_length, max(token_index.keys()) + 1))
for i, sample in enumerate(samples):
  for j, character in enumerate(sample):
    index = token_index.get(character)
    results[i, j, index] = 1.
```

```
[7] results.shape
```

```
(2, 50, 101)
```

CHARACTER-LEVEL ONE-HOT ENCODING

https://github.com/YousraMashkoor/Deep-Learning-by-francois-chollet-SOLVED/blob/master/Chapter%2006/Listing_6_2_Character_level_one_hot_encoding_(toy_example).ipynb



```python
from keras.preprocessing.text import Tokenizer
samples = ['The cat sat on the mat.', 'The dog ate my homework.']
tokenizer = Tokenizer(num_words=1000)
tokenizer.fit_on_texts(samples)
sequences = tokenizer.texts_to_sequences(samples)
one_hot_results = tokenizer.texts_to_matrix(samples, mode='binary')
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
```

```
Found 9 unique tokens.
```

```python
sequences
```

```
[[1, 2, 3, 4, 1, 5], [1, 6, 7, 8, 9]]
```

```python
one_hot_results
```

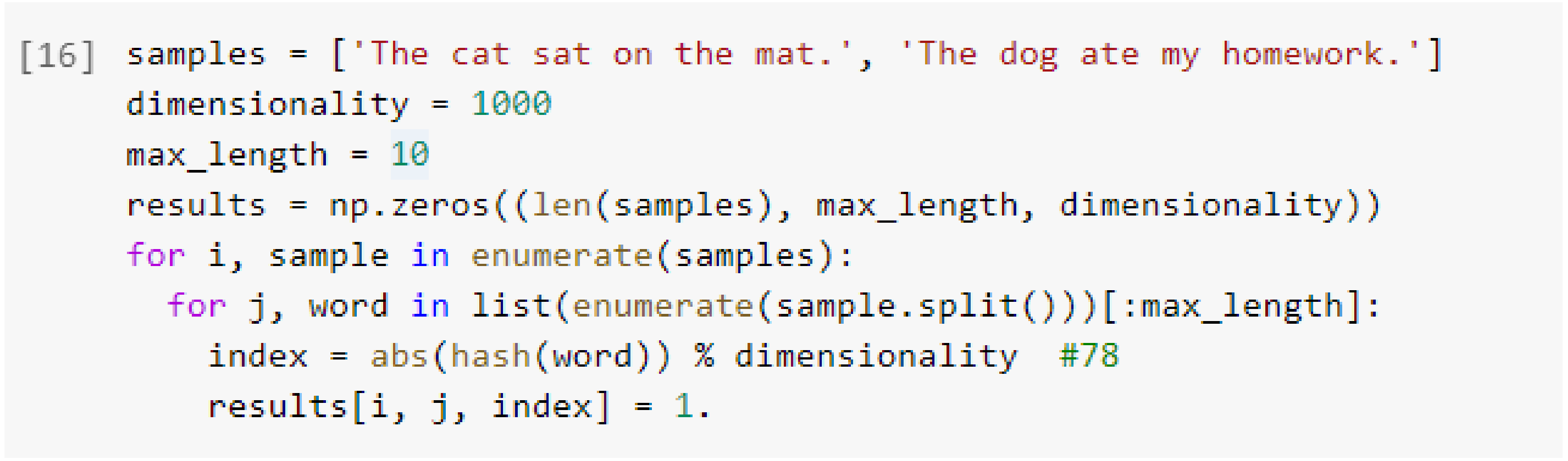
```
array([[0., 1., 1., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.]])
```

KERAS FOR WORD-LEVEL ONE-HOT ENCODING

https://github.com/YousraMashkoor/Deep-Learning-by-francois-chollet-SOLVED/blob/master/Chapter%2006/Listing_6_3_Using_Keras_for_word_level_one_hot_encoding.ipynb

One hot hashing trick

- A variant of one-hot encoding is the so-called one-hot hashing trick.
- when the number of unique tokens is too large
- hash words into vectors of fixed size.
- does away with maintaining an explicit word index
- saves memory
- allows online encoding of the data
- The likelihood of hash collisions decreases when the dimensionality of the hashing space is larger than the total number of unique tokens .

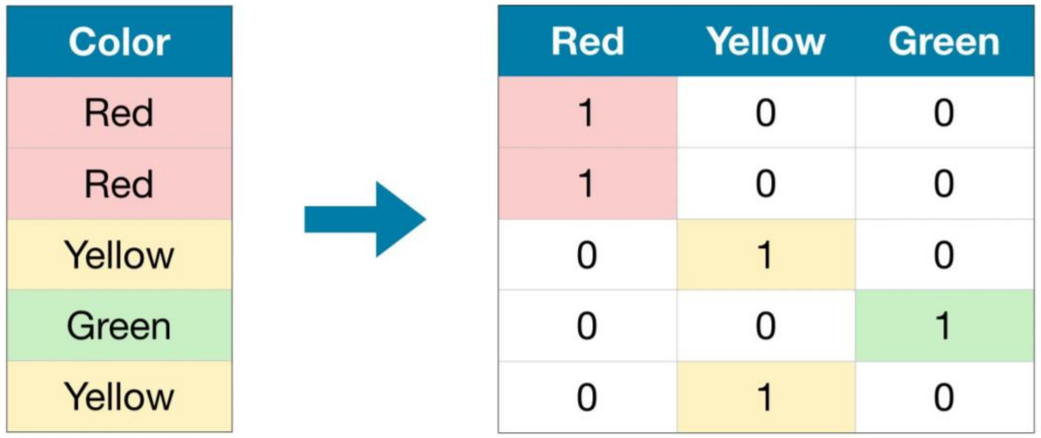


```
[16] samples = ['The cat sat on the mat.', 'The dog ate my homework.']
     dimensionality = 1000
     max_length = 10
     results = np.zeros((len(samples), max_length, dimensionality))
     for i, sample in enumerate(samples):
       for j, word in list(enumerate(sample.split()))[:max_length]:
         index = abs(hash(word)) % dimensionality  #78
         results[i, j, index] = 1.
```

WORD-LEVEL ONE-HOT ENCODING WITH HASHING TRICK

https://github.com/YousraMashkoor/Deep-Learning-by-francois-chollet-SOLVED/blob/master/Chapter%2006/Listing_6_4_Word_level_one_hot_encoding_with_hashing_trick_(toy_example).ipynb

WORD EMBEDDING



Color
Red
Red
Yellow
Green
Yellow

Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1
0	1	0

- Use of dense word vectors, also called word embeddings.
- word embeddings are lowdimensional floating-point vectors
- Word embeddings pack more information into far fewer dimensions.

TWO WAYS

Learning word vectors

- start with random word vectors and then learn word vectors in the same way you learn the weights of a neural network.

pretrained word embeddings.

To be continued!

Title Lorem Ipsum Dolor

LOREM IPSUM DOLOR SIT AMET, CONSECTETUER ADIPISCING ELIT.

NUNC VIVERRA IMPERDIET ENIM. FUSCE EST. VIVAMUS A TELLUS.

LOREM IPSUM DOLOR SIT AMET, CONSECTETUER ADIPISCING ELIT.