one_hot_encoding_of_words_or_characters

December 20, 2024

```
[3]: import keras
     keras.__version__
```

```
[3]: '3.5.0'
```

1 One-hot encoding of words or characters

This notebook contains the first code sample found in Chapter 6, Section 1 of Deep Learning with Python. Note that the original text features far more content, in particular further explanations and figures: in this notebook, you will only find source code and related comments.

One-hot encoding is the most common, most basic way to turn a token into a vector. You already saw it in action in our initial IMDB and Reuters examples from chapter 3 (done with words, in our case). **It consists of associating a unique integer index to every word, then turning this integer index i into a binary vector of size N, the size of the vocabulary, that would be all-zeros except for the i-th entry, which would be 1.**

Of course, one-hot encoding can be done at the character level as well. To unambiguously drive home what one-hot encoding is and how to implement it, here are two toy examples of one-hot encoding: one for words, the other for characters.

Word level one-hot encoding (toy example):

```
[4]: import numpy as np

     # This is our initial data; one entry per "sample"
     # (in this toy example, a "sample" is just a sentence, but
     # it could be an entire document).
     samples = ['The cat sat on the mat.', 'The dog ate my homework.']

     # First, build an index of all tokens in the data.
     token_index = {}
     for sample in samples:
         # We simply tokenize the samples via the `split` method.
         # in real life, we would also strip punctuation and special characters
         # from the samples.
         for word in sample.split():
```

```
        if word not in token_index:
            # Assign a unique index to each unique word
            token_index[word] = len(token_index)
            # Note that we don't attribute index 0 to anything.

# Next, we vectorize our samples.
# We will only consider the first `max_length` words in each sample.
max_length = 10

# This is where we store our results:
results = np.zeros((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)
        #print(index-1)
        results[i, j, index] = 1.
```

[5]:
```
print(token_index)
print(results)
```

```
{'The': 0, 'cat': 1, 'sat': 2, 'on': 3, 'the': 4, 'mat.': 5, 'dog': 6, 'ate': 7,
'my': 8, 'homework.': 9}
[[[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]

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

Character level one-hot encoding (toy example)

[7]:
```
import string

samples = ['', 'The dog ate my homework.']
#characters = string.printable  # All printable ASCII characters.
import numpy as np

samples = ['The cat sat on the mat.', 'The dog ate my homework.']

# Combine all characters from the samples into a single string
all_text = ''.join(samples)

# Get unique characters
characters = list(set(all_text))
characters.sort()
token_index = dict(zip(characters, range(0, len(characters))))

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

[8]:
```
print(token_index)
print(results)
```

```
{' ': 0, '.': 1, 'T': 2, 'a': 3, 'c': 4, 'd': 5, 'e': 6, 'g': 7, 'h': 8, 'k': 9,
'm': 10, 'n': 11, 'o': 12, 'r': 13, 's': 14, 't': 15, 'w': 16, 'y': 17}
[[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
  [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
  [0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
  [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
  [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
  [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 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. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]]]
```

[9]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[10]:
```
%ls
```

```
drive/  sample_data/
```

[11]:
```
%cd /content/drive/MyDrive/Deep_Learning_With_Tensorflow
```

```
/content/drive/MyDrive/Deep_Learning_With_Tensorflow
```

[13]:
```
# The map function applies the str function to each element of vector,␣
↪converting it into a string. The result is an iterable of strings that ','.
↪join() can handle

file_path = '/content/drive/MyDrive/Deep_Learning_With_Tensorflow/output.txt'

# Save the results in a human-readable format
with open(file_path, 'w') as f:
    for i, sample_vector in enumerate(results):
        f.write(f"Sample {i+1}:\n")
        for vector in sample_vector:
            f.write(' '.join(map(str, vector.astype(int))) + '\n')  # Convert␣
↪to integers and save as comma-separated
        f.write('\n')  # Add a blank line between samples for clarity
```

```python
# Read the file to verify the content
with open(file_path, 'r') as f:
    content = f.read()
    print(content)
```

```
Sample 1:
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0

Sample 2:
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 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 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
```

[14]:
```python
import numpy as np

# This is our initial data; one entry per "sample"
samples = ['The cat sat on the mat.', 'The dog ate my homework.']

# First, build an index of all tokens in the data.
token_index = {}
for sample in samples:
    for word in sample.split():
        if word not in token_index:
            # Assign a unique index to each unique word
            token_index[word] = len(token_index)

# Traditional one-hot encoding
results = []  # To store the results for all sentences
vocabulary_size = len(token_index)  # Size of the vocabulary

for sample in samples:
    sentence_encoding = []  # To store the one-hot encoding for each word in␣
 ↪the sentence
    for word in sample.split():
        # Create a zero vector of length equal to the vocabulary size
        one_hot_vector = np.zeros(vocabulary_size)
        # Set the index corresponding to the word to 1
        one_hot_vector[token_index[word]] = 1
        # Add the one-hot vector to the sentence encoding
        sentence_encoding.append(one_hot_vector)
    # Add the sentence encoding to the results
    results.append(sentence_encoding)

# Display the results
for i, sample_encoding in enumerate(results):
    print(f"Sentence {i + 1} One-Hot Encoding:")
    for word_vector in sample_encoding:
        print(word_vector)
```

```
Sentence 1 One-Hot Encoding:
[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
Sentence 2 One-Hot Encoding:
[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
```

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

[15]:
```
import sys

# Save the current stdout
original_stdout = sys.stdout

# Open a file in write mode
with open('/content/drive/MyDrive/Deep_Learning_With_Tensorflow/output.txt',␣
 ↪'w') as f:
    # Redirect stdout to the file
    sys.stdout = f

    # Your print statements here
    print(results)
    #print("This is a sample output.")

# Reset stdout to the original value
sys.stdout = original_stdout
```

[19]:
```
!pip install keras.preprocessing
```

```
Collecting keras.preprocessing
  Downloading Keras_Preprocessing-1.1.2-py2.py3-none-any.whl.metadata (1.9 kB)
Requirement already satisfied: numpy>=1.9.1 in /usr/local/lib/python3.10/dist-
packages (from keras.preprocessing) (1.26.4)
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.10/dist-
packages (from keras.preprocessing) (1.17.0)
Downloading Keras_Preprocessing-1.1.2-py2.py3-none-any.whl (42 kB)
                         0.0/42.6 kB
? eta -:--:--
                      42.6/42.6 kB 3.1

MB/s eta 0:00:00
Installing collected packages: keras.preprocessing
Successfully installed keras.preprocessing-1.1.2
```

Note that Keras has built-in utilities for doing one-hot encoding text at the word level or character level, starting from raw text data. This is what you should actually be using, as it will take care of a number of important features, such as stripping special characters from strings, or only taking into the top N most common words in your dataset (a common restriction to avoid dealing with very large input vector spaces).

Using Keras for word-level one-hot encoding:

[23]:
```
from tensorflow.keras.preprocessing.text import Tokenizer

samples = ['The cat sat on the mat.', 'The dog ate my homework.']
```

```
# We create a tokenizer, configured to only take
# into account the top-1000 most common words
tokenizer = Tokenizer(num_words=1000)
# This builds the word index
tokenizer.fit_on_texts(samples)

# This turns strings into lists of integer indices.
sequences = tokenizer.texts_to_sequences(samples)

# You could also directly get the one-hot binary representations.
# Note that other vectorization modes than one-hot encoding are supported!
one_hot_results = tokenizer.texts_to_matrix(samples, mode='binary')

# This is how you can recover the word index that was computed
word_index = tokenizer.word_index
print(word_index)
print('Found %s unique tokens.' % len(word_index))
print(one_hot_results)
```

```
{'the': 1, 'cat': 2, 'sat': 3, 'on': 4, 'mat': 5, 'dog': 6, 'ate': 7, 'my': 8,
'homework': 9}
Found 9 unique tokens.
[[0. 1. 1. … 0. 0. 0.]
 [0. 1. 0. … 0. 0. 0.]]
```