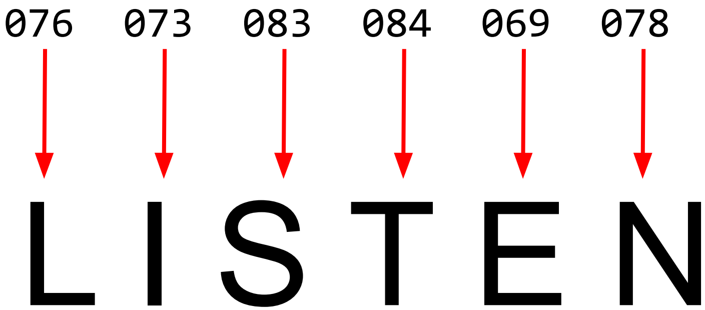
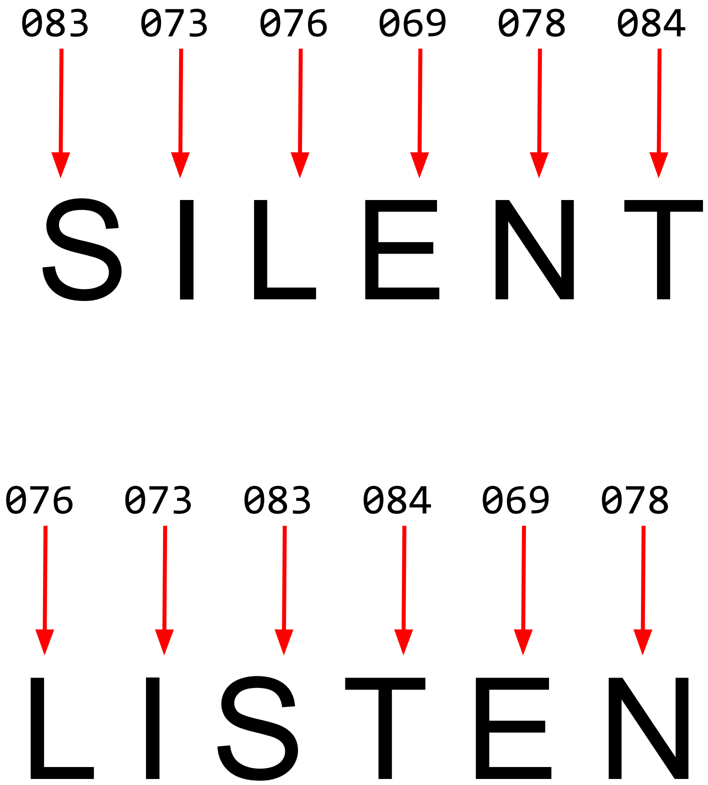
Copyright Notice

LISTEN



076 073 083 084 069 078

LISTEN



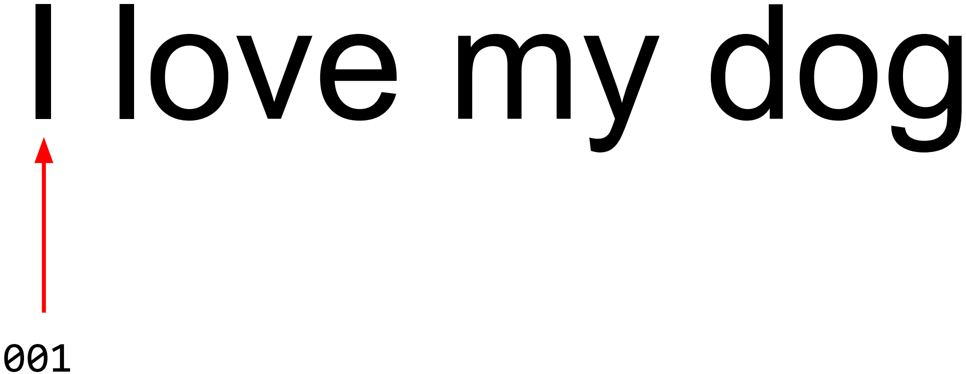
083 073 076 069 078 084

SILENT

076 073 083 084 069 078

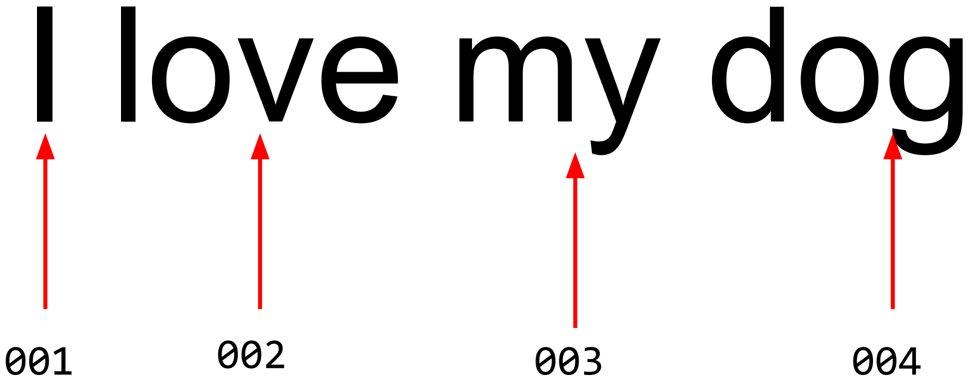
LISTEN

I love my dog



I love my dog

001



I love my dog

001

002

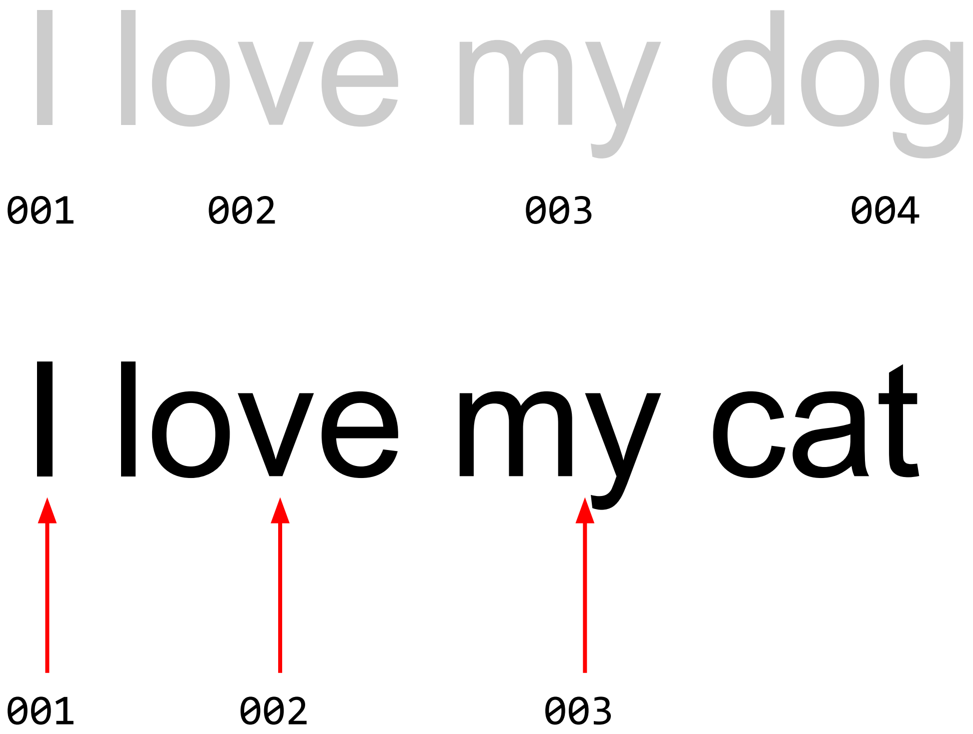
003

004

I love my dog

001 002 003 004

I love my cat

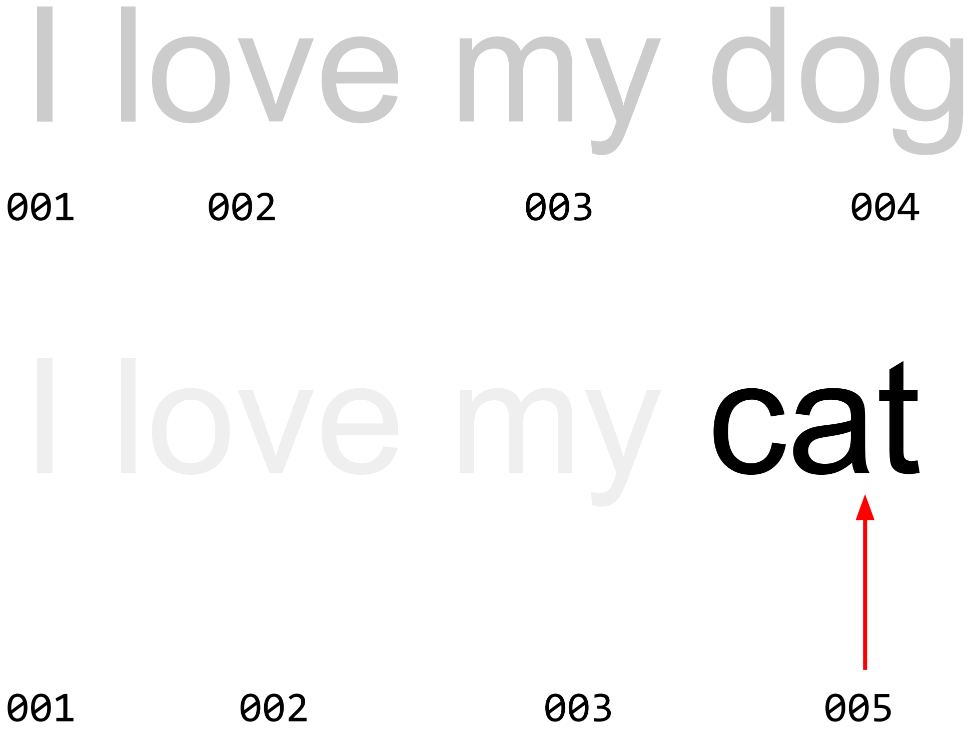


I love my dog

001 002 003 004

I love my cat

001 002 003



I love my dog

001 002 003 004

I love my cat

001 002 003 005

001	002	003	004
001	002	003	005

```python
import tensorflow as tf

sentences = [
    'I love my dog',
    'I love my cat'
]

vectorize_layer = tf.keras.layers.TextVectorization()

vectorize_layer.adapt(sentences)

vocabulary = vectorize_layer.get_vocabulary(include_special_tokens=False)

print(vocabulary)
```

```
import tensorflow as tf

sentences = [
    'I love my dog',
    'I love my cat'
]

vectorize_layer = tf.keras.layers.TextVectorization()

vectorize_layer.adapt(sentences)

vocabulary = vectorize_layer.get_vocabulary(include_special_tokens=False)

print(vocabulary)
```

```
import tensorflow as tf

sentences = [
    'I love my dog',
    'I love my cat'
]

vectorize_layer = tf.keras.layers.TextVectorization()

vectorize_layer.adapt(sentences)

vocabulary = vectorize_layer.get_vocabulary(include_special_tokens=False)

print(vocabulary)
```

```python
import tensorflow as tf

sentences = [
    'I love my dog',
    'I love my cat'
]

vectorize_layer = tf.keras.layers.TextVectorization()

vectorize_layer.adapt(sentences)

vocabulary = vectorize_layer.get_vocabulary(include_special_tokens=False)

print(vocabulary)
```

```
import tensorflow as tf

sentences = [
    'I love my dog',
    'I love my cat'
]

vectorize_layer = tf.keras.layers.TextVectorization()

vectorize_layer.adapt(sentences)

vocabulary = vectorize_layer.get_vocabulary(include_special_tokens=False)

print(vocabulary)
```

```
import tensorflow as tf

sentences = [
    'I love my dog',
    'I love my cat'
]

vectorize_layer = tf.keras.layers.TextVectorization()

vectorize_layer.adapt(sentences)

vocabulary = vectorize_layer.get_vocabulary(include_special_tokens=False)

print(vocabulary)
```

```
['my', 'love', 'i', 'dog', 'cat']
```

```python
sentences = [
    'I love my dog',
    'I love my cat',
    'You love my dog!'
]
```

```
sentences = [
    'I love my dog',
    'I love my cat',
    'You love my dog!'
]
```

```
['my', 'love', 'i', 'dog', 'you', 'cat']
```

```
['my', 'love', 'i', 'dog', 'you', 'cat']
```

```
['my', 'love', 'i', 'dog', 'you', 'cat']
```

```python
import tensorflow as tf

sentences = [
    'I love my dog',
    'I love my cat'
]

vectorize_layer = tf.keras.layers.TextVectorization()

vectorize_layer.adapt(sentences)

vocabulary = vectorize_layer.get_vocabulary(include_special_tokens=False)

print(vocabulary)
```

```
import tensorflow as tf

sentences = [
    'I love my dog',
    'I love my cat'
]

vectorize_layer = tf.keras.layers.TextVectorization()

vectorize_layer.adapt(sentences)

vocabulary = vectorize_layer.get_vocabulary()

print(vocabulary)
```

```
['', '[UNK]', 'my', 'love', 'i', 'dog', 'you', 'cat']
```

```
import tensorflow as tf

sentences = [
    'I love my dog',
    'I love my cat',
    'You love my dog!',
    'Do you think my dog is amazing?'
]

vectorize_layer = tf.keras.layers.TextVectorization()

vectorize_layer.adapt(sentences)

vocabulary = vectorize_layer.get_vocabulary()

for index, word in enumerate(vocabulary):
    print(index, word)
```

```
import tensorflow as tf

sentences = [
    'I love my dog',
    'I love my cat',
    'You love my dog!',
    'Do you think my dog is amazing?'
]

vectorize_layer = tf.keras.layers.TextVectorization()

vectorize_layer.adapt(sentences)

vocabulary = vectorize_layer.get_vocabulary()

for index, word in enumerate(vocabulary):
    print(index, word)
```

```
import tensorflow as tf

sentences = [
    'I love my dog',
    'I love my cat',
    'You love my dog!',
    'Do you think my dog is amazing?'
]

vectorize_layer = tf.keras.layers.TextVectorization()

vectorize_layer.adapt(sentences)

vocabulary = vectorize_layer.get_vocabulary()

for index, word in enumerate(vocabulary):
    print(index, word)
```

```python
for index, word in enumerate(vocabulary):
    print(index, word)
```

```
0 
1 [UNK]
2 my
3 love
4 dog
5 you
6 i
7 think
8 is
9 do
10 cat
11 amazing
```

```python
import tensorflow as tf

sentences = [
    'I love my dog',
    'I love my cat',
    'You love my dog!',
    'Do you think my dog is amazing?'
]

vectorize_layer = tf.keras.layers.TextVectorization()

vectorize_layer.adapt(sentences)

vocabulary = vectorize_layer.get_vocabulary()

sequence = vectorize_layer('I love my dog')

for index, word in enumerate(vocabulary):
    print(index, word)

print(sequence)
```

```
for index, word in enumerate(vocabulary):
    print(index, word)

0
1 [UNK]
2 my
3 love
4 dog
5 you
6 i
7 think
8 is
9 do
10 cat
11 amazing

print(sequence)

tf.Tensor([6 3 2 4], shape=(4,), dtype=int64)
```

```python
import tensorflow as tf

sentences = [
    'I love my dog',
    'I love my cat',
    'You love my dog!',
    'Do you think my dog is amazing?'
]

vectorize_layer = tf.keras.layers.TextVectorization()

vectorize_layer.adapt(sentences)

vocabulary = vectorize_layer.get_vocabulary()

sequences = vectorize_layer(sentences)

for index, word in enumerate(vocabulary):
    print(index, word)

print(sequences)
```

```
for index, word in enumerate(vocabulary):
    print(index, word)

0
1 [UNK]
2 my
3 love
4 dog
5 you
6 i
7 think
8 is
9 do
10 cat
11 amazing
```

```
print(sequences)

tf.Tensor(
[[ 6  3  2  4  0  0  0]
 [ 6  3  2 10  0  0  0]
 [ 5  3  2  4  0  0  0]
 [ 9  5  7  2  4  8 11]], shape=(4, 7), dtype=int64)
```

```
for index, word in enumerate(vocabulary):
    print(index, word)
```

```
0
1 [UNK]
2 my
3 love
4 dog
5 you
6 i
7 think
8 is
9 do
10 cat
11 amazing
```

```
print(sequences)
```

```
tf.Tensor(
[[ 6  3  2  4  0  0  0]
 [ 6  3  2 10  0  0  0]
 [ 5  3  2  4  0  0  0]
 [ 9  5  7  2  4  8 11]], shape=(4, 7), dtype=int64)
```

```
for index, word in enumerate(vocabulary):
    print(index, word)
```

```
0
1 [UNK]
2 my
3 love
4 dog
5 you
6 i
7 think
8 is
9 do
10 cat
11 amazing
```

```
print(sequences)
```

```
tf.Tensor(
[[ 6  3  2  4  0  0  0]
 [ 6  3  2 10  0  0  0]
 [ 5  3  2  4  0  0  0]
 [ 9  5  7  2  4  8 11]], shape=(4, 7), dtype=int64)
```

```
for index, word in enumerate(vocabulary):
    print(index, word)
```

```
0
1 [UNK]
2 my
3 love
4 dog
5 you
6 i
7 think
8 is
9 do
10 cat
11 amazing
```

```
print(sequences)
```

```
tf.Tensor(
[[ 6  3  2  4  0  0  0]
 [ 6  3  2 10  0  0  0]
 [ 5  3  2  4  0  0  0]
 [ 9  5  7  2  4  8 11]], shape=(4, 7), dtype=int64)
```

```
sequences_pre = tf.keras.utils.pad_sequences(sequences, padding='pre')



print(sequences)

OUTPUT:
[[ 0  0  0  6  3  2  4]
 [ 0  0  0  6  3  2 10]
 [ 0  0  0  5  3  2  4]
 [ 9  5  7  2  4  8 11]]
```

```
sentences = [
    'I love my dog',
    'I love my cat',
    'You love my dog!',
    'Do you think my dog is amazing?'
]

vectorize_layer = tf.keras.layers.TextVectorization()
vectorize_layer.adapt(sentences)
vocabulary = vectorize_layer.get_vocabulary()
```

```
sequences = vectorize_layer(sentences)

print(sequences)

tf.Tensor(
[[ 6  3  2  4  0  0  0]
 [ 6  3  2 10  0  0  0]
 [ 5  3  2  4  0  0  0]
 [ 9  5  7  2  4  8 11]], shape=(4, 7), dtype=int64)
```

```
sentences = [
    'I love my dog',
    'I love my cat',
    'You love my dog!',
    'Do you think my dog is amazing?'
]

vectorize_layer = tf.keras.layers.TextVectorization()
vectorize_layer.adapt(sentences)
vocabulary = vectorize_layer.get_vocabulary()

sentences_dataset = tf.data.Dataset.from_tensor_slices(sentences)

sequences = sentences_dataset.map(vectorize_layer)
```

```
sentences = [
    'I love my dog',
    'I love my cat',
    'You love my dog!',
    'Do you think my dog is amazing?'
]

vectorize_layer = tf.keras.layers.TextVectorization()
vectorize_layer.adapt(sentences)
vocabulary = vectorize_layer.get_vocabulary()

sentences_dataset = tf.data.Dataset.from_tensor_slices(sentences)

sequences = sentences_dataset.map(vectorize_layer)
```

```
sentences = [
    'I love my dog',
    'I love my cat',
    'You love my dog!',
    'Do you think my dog is amazing?'
]

vectorize_layer = tf.keras.layers.TextVectorization()
vectorize_layer.adapt(sentences)
vocabulary = vectorize_layer.get_vocabulary()

sentences_dataset = tf.data.Dataset.from_tensor_slices(sentences)

sequences = sentences_dataset.map(vectorize_layer)

print(sequences)
```

```
<_MapDataset element_spec=TensorSpec(shape=(None,), dtype=tf.int64, name=None)>
```

```
sentences = [
    'I love my dog',
    'I love my cat',
    'You love my dog!',
    'Do you think my dog is amazing?'
]

vectorize_layer = tf.keras.layers.TextVectorization()
vectorize_layer.adapt(sentences)
vocabulary = vectorize_layer.get_vocabulary()

sentences_dataset = tf.data.Dataset.from_tensor_slices(sentences)

sequences = sentences_dataset.map(vectorize_layer)

print(sequences)
```

```
<_MapDataset element_spec=TensorSpec(shape=(None,), dtype=tf.int64, name=None)>
```

```
sentences = [
    'I love my dog',
    'I love my cat',
    'You love my dog!',
    'Do you think my dog is amazing?'
]

vectorize_layer = tf.keras.layers.TextVectorization()
vectorize_layer.adapt(sentences)
vocabulary = vectorize_layer.get_vocabulary()

sentences_dataset = tf.data.Dataset.from_tensor_slices(sentences)

sequences = sentences_dataset.map(vectorize_layer)
```

```
for sentence, sequence in zip(sentences, sequences):
    print(f'{sentence} ---> {sequence}')
```

```python
vectorize_layer = tf.keras.layers.TextVectorization()
vectorize_layer.adapt(sentences)
vocabulary = vectorize_layer.get_vocabulary()

sentences_dataset = tf.data.Dataset.from_tensor_slices(sentences)

sequences = sentences_dataset.map(vectorize_layer)



for sentence, sequence in zip(sentences, sequences):
    print(f'{sentence} ---> {sequence}')
```

```
I love my dog ---> [6 3 2 4]
I love my cat ---> [6 3 2 10]
You love my dog! ---> [5 3 2 4]
Do you think my dog is amazing? ---> [9 5 7 2 4 8 11]
```

```
vectorize_layer = tf.keras.layers.TextVectorization()
vectorize_layer.adapt(sentences)
vocabulary = vectorize_layer.get_vocabulary()

sentences_dataset = tf.data.Dataset.from_tensor_slices(sentences)

sequences = sentences_dataset.map(vectorize_layer)



for sentence, sequence in zip(sentences, sequences):
    print(f'{sentence} ---> {sequence}')
```

```
I love my dog ---> [6 3 2 4]
I love my cat ---> [6 3 2 10]
You love my dog! ---> [5 3 2 4]
Do you think my dog is amazing? ---> [9 5 7 2 4 8 11]
```

```python
vectorize_layer = tf.keras.layers.TextVectorization()
vectorize_layer.adapt(sentences)
vocabulary = vectorize_layer.get_vocabulary()

sentences_dataset = tf.data.Dataset.from_tensor_slices(sentences)

sequences = sentences_dataset.map(vectorize_layer)

sequences_pre = tf.keras.utils.pad_sequences(sequences, padding='pre')

print(sequences_pre)
```

```
[[ 0  0  0  6  3  2  4]
 [ 0  0  0  6  3  2 10]
 [ 0  0  0  5  3  2  4]
 [ 9  5  7  2  4  8 11]]
```

```
vectorize_layer = tf.keras.layers.TextVectorization()
vectorize_layer.adapt(sentences)
vocabulary = vectorize_layer.get_vocabulary()

sentences_dataset = tf.data.Dataset.from_tensor_slices(sentences)

sequences = sentences_dataset.map(vectorize_layer)

sequences_pre = tf.keras.utils.pad_sequences(sequences, padding='pre')

print(sequences_pre)
```

```
[[ 0  0  0  6  3  2  4]
 [ 0  0  0  6  3  2 10]
 [ 0  0  0  5  3  2  4]
 [ 9  5  7  2  4  8 11]]
```

```
vectorize_layer = tf.keras.layers.TextVectorization()
vectorize_layer.adapt(sentences)
vocabulary = vectorize_layer.get_vocabulary()

sentences_dataset = tf.data.Dataset.from_tensor_slices(sentences)

sequences = sentences_dataset.map(vectorize_layer)

sequences_pre = tf.keras.utils.pad_sequences(sequences, padding='pre')

print(sequences_pre)
```

```
[[ 0  0  0  6  3  2  4]
 [ 0  0  0  6  3  2 10]
 [ 0  0  0  5  3  2  4]
 [ 9  5  7  2  4  8 11]]
```

```python
import tensorflow as tf

sentences = [
    'I love my dog',
    'I love my cat',
    'You love my dog!',
    'Do you think my dog is amazing?'
]

vectorize_layer = tf.keras.layers.TextVectorization(ragged=True)

vectorize_layer.adapt(sentences)

vocabulary = vectorize_layer.get_vocabulary()

ragged_sequences = vectorize_layer(sentences)

for index, word in enumerate(vocabulary):
    print(index, word)

print(ragged_sequences)
```

```
for index, word in enumerate(vocabulary):
    print(index, word)
```

```
0
1 [UNK]
2 my
3 love
4 dog
5 you
6 i
7 think
8 is
9 do
10 cat
11 amazing
```

```
print(ragged_sequences)
```

```
<tf.RaggedTensor [[6, 3, 2, 4], [6, 3, 2, 10], [5, 3, 2, 4], [9, 5, 7, 2, 4, 8, 11]]>
```

```
for index, word in enumerate(vocabulary):
    print(index, word)

0
1 [UNK]
2 my
3 love
4 dog
5 you
6 i
7 think
8 is
9 do
10 cat
11 amazing
```

```
print(ragged_sequences)

<tf.RaggedTensor [[6, 3, 2, 4]  [6, 3, 2, 10], [5, 3, 2, 4], [9, 5, 7, 2, 4, 8, 11]]>
```

```python
import tensorflow as tf

sentences = [
    'I love my dog',
    'I love my cat',
    'You love my dog!',
    'Do you think my dog is amazing?'
]

vectorize_layer = tf.keras.layers.TextVectorization(ragged=True)

vectorize_layer.adapt(sentences)

vocabulary = vectorize_layer.get_vocabulary()

ragged_sequences = vectorize_layer(sentences)

pre_padded_sequences = tf.keras.utils.pad_sequences(ragged_sequences.numpy())

print(pre_padded_sequences)
print(pre_padded_sequences)

[[ 0  0  0  6  3  2  4]
 [ 0  0  0  6  3  2 10]
 [ 0  0  0  5  3  2  4]
 [ 9  5  7  2  4  8 11]]
```

```
sentences = [
    'I love my dog',
    'I love my cat',
    'You love my dog!',
    'Do you think my dog is amazing?'
]

vectorize_layer = tf.keras.layers.TextVectorization()
vectorize_layer.adapt(sentences)

test_data = [
    'i really love my dog',
    'my dog loves my manatee'
]
```

```
sentences = [
    'I love my dog',
    'I love my cat',
    'You love my dog!',
    'Do you think my dog is amazing?'
]

vectorize_layer = tf.keras.layers.TextVectorization()
vectorize_layer.adapt(sentences)

test_data = [
    'i really love my dog',
    'my dog loves my manatee'
]
```

```
test_data = [
    'i really love my dog',
    'my dog loves my manatee'
]

test_seq = vectorize_layer(test_data)

print(test_seq)
```

```
test_data = [
    'i really love my dog',
    'my dog loves my manatee'
]

test_seq = vectorize_layer(test_data)

print(test_seq)
```

```
tf.Tensor(
[[6 1 3 2 4]
 [2 4 1 2 1]], shape=(2, 5), dtype=int64)
```

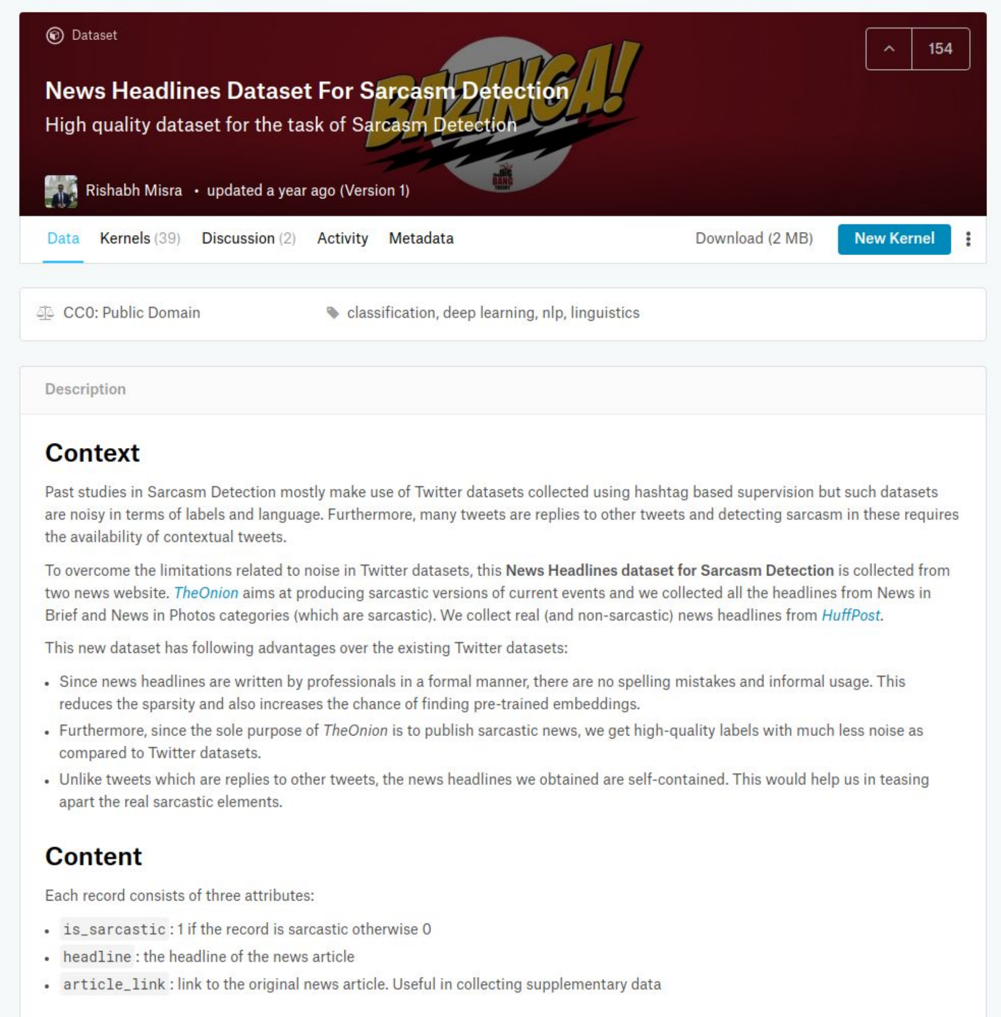
```
test_data = [
    'i really love my dog',
    'my dog loves my manatee'
]

test_seq = vectorize_layer(test_data)

print(test_seq)
```

```
tf.Tensor(
[[6 1 3 2 4]
 [2 4 1 2 1]], shape=(2, 5), dtype=int64)
```

```
0
1 [UNK]
2 my
3 love
4 dog
5 you
6 i
7 think
8 is
9 do
10 cat
11 amazing
```

Sarcasm in News Headlines Dataset by Rishabh Misra

https://rishabhmisra.github.io/publications/



News Headlines Dataset For Sarcasm Detection

High quality dataset for the task of Sarcasm Detection

Rishabh Misra • updated a year ago (Version 1)

CC0: Public Domain   classification, deep learning, nlp, linguistics

Description

Context

Past studies in Sarcasm Detection mostly make use of Twitter datasets collected using hashtag based supervision but such datasets are noisy in terms of labels and language. Furthermore, many tweets are replies to other tweets and detecting sarcasm in these requires the availability of contextual tweets.

To overcome the limitations related to noise in Twitter datasets, this **News Headlines dataset for Sarcasm Detection** is collected from two news website. *TheOnion* aims at producing sarcastic versions of current events and we collected all the headlines from News in Brief and News in Photos categories (which are sarcastic). We collect real (and non-sarcastic) news headlines from *HuffPost*.

This new dataset has following advantages over the existing Twitter datasets:

- Since news headlines are written by professionals in a formal manner, there are no spelling mistakes and informal usage. This reduces the sparsity and also increases the chance of finding pre-trained embeddings.
- Furthermore, since the sole purpose of *TheOnion* is to publish sarcastic news, we get high-quality labels with much less noise as compared to Twitter datasets.
- Unlike tweets which are replies to other tweets, the news headlines we obtained are self-contained. This would help us in teasing apart the real sarcastic elements.

Content

Each record consists of three attributes:

- `is_sarcastic` : 1 if the record is sarcastic otherwise 0
- `headline` : the headline of the news article
- `article_link` : link to the original news article. Useful in collecting supplementary data

is_sarcastic: 1 if the record is sarcastic otherwise 0

headline: the headline of the news article

article_link: link to the original news article. Useful in collecting supplementary data

```
{"article_link":
"https://politics.theonion.com/boehner-just-wants-wife-to-listen-not-come-up-with-alt-18195
74302", "headline": "boehner just wants wife to listen, not come up with alternative
debt-reduction ideas", "is_sarcastic": 1}

{"article_link":
"https://www.huffingtonpost.com/entry/roseanne-revival-review_us_5ab3a497e4b054d118e04365",
"headline": "the 'roseanne' revival catches up to our thorny political mood, for better and
worse", "is_sarcastic": 0}

{"article_link":
"https://local.theonion.com/mom-starting-to-fear-son-s-web-series-closest-thing-she-1819576
697", "headline": "mom starting to fear son's web series closest thing she will have to
grandchild", "is_sarcastic": 1}
```

```
[

{"article_link": "https://politics.theonion.com/boehner-just-wants-wife-to-listen-not-come-up-with-alt-1819574302", "headline": "boehner just wants wife to listen, not come up with alternative debt-reduction ideas", "is_sarcastic": 1},

{"article_link": "https://www.huffingtonpost.com/entry/roseanne-revival-review_us_5ab3a497e4b054d118e04365", "headline": "the 'roseanne' revival catches up to our thorny political mood, for better and worse", "is_sarcastic": 0},

{"article_link": "https://local.theonion.com/mom-starting-to-fear-son-s-web-series-closest-thing-she-1819576697", "headline": "mom starting to fear son's web series closest thing she will have to grandchild", "is_sarcastic": 1}

]
```

```python
import json

with open("sarcasm.json", 'r') as f:
    datastore = json.load(f)

sentences = []
labels = []
urls = []
for item in datastore:
    sentences.append(item['headline'])
    labels.append(item['is_sarcastic'])
    urls.append(item['article_link'])
```

```python
import json

with open("sarcasm.json", 'r') as f:
    datastore = json.load(f)

sentences = []
labels = []
urls = []
for item in datastore:
    sentences.append(item['headline'])
    labels.append(item['is_sarcastic'])
    urls.append(item['article_link'])
```

```
import json

with open("sarcasm.json", 'r') as f:
    datastore = json.load(f)

sentences = []
labels = []
urls = []
for item in datastore:
    sentences.append(item['headline'])
    labels.append(item['is_sarcastic'])
    urls.append(item['article_link'])
```

```
import json

with open("sarcasm.json", 'r') as f:
    datastore = json.load(f)

sentences = []
labels = []
urls = []
for item in datastore:
    sentences.append(item['headline'])
    labels.append(item['is_sarcastic'])
    urls.append(item['article_link'])
```

```python
import json

with open("sarcasm.json", 'r') as f:
    datastore = json.load(f)

sentences = []
labels = []
urls = []
for item in datastore:
    sentences.append(item['headline'])
    labels.append(item['is_sarcastic'])
    urls.append(item['article_link'])
```

```
import tensorflow as tf

vectorize_layer = tf.keras.layers.TextVectorization()

vectorize_layer.adapt(sentences)

vocabulary = vectorize_layer.get_vocabulary()

post_padded_sequences = vectorize_layer(sentences)

print(f'padded sequence: {post_padded_sequences[2]}')

padded sequence: [140 825 2 813 1100 2048 571 5057 199 139 39 46 2
13050 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]
```

```python
import tensorflow as tf

vectorize_layer = tf.keras.layers.TextVectorization()

vectorize_layer.adapt(sentences)

vocabulary = vectorize_layer.get_vocabulary()

post_padded_sequences = vectorize_layer(sentences)

print(f'padded sequence: {post_padded_sequences[2]}')

padded sequence: [140 825 2 813 1100 2048 571 5057 199 139 39 46 2
13050 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]
```

```
import tensorflow as tf

vectorize_layer = tf.keras.layers.TextVectorization()

vectorize_layer.adapt(sentences)

vocabulary = vectorize_layer.get_vocabulary()

post_padded_sequences = vectorize_layer(sentences)

print(f'padded sequence: {post_padded_sequences[2]}')

padded sequence: [140 825 2 813 1100 2048 571 5057 199 139 39 46 2
13050 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]
```