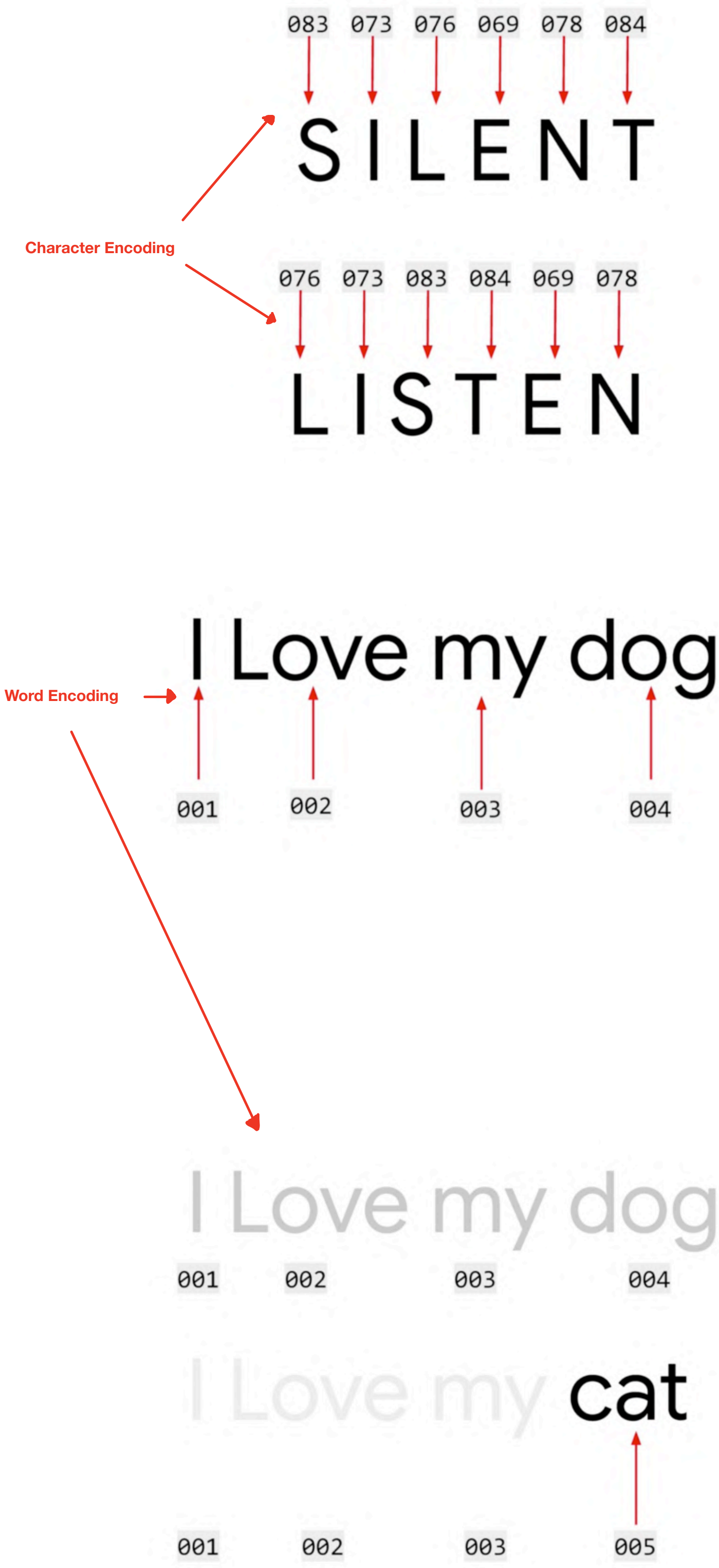

083 073 076 069 078 084
SILENT
Character Encoding
076 073 083 084 069 078
LISTEN
I Love my dog
Word Encoding
001 002 003 004
I Love my dog
001 002 003 004
I Love my cat
001 002 003 005

I	Love	My	Dog
001	002	003	004
001	002	003	005
I	Love	My	Cat

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

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

tokenizer = Tokenizer(num_words = 100)
tokenizer.fit_on_texts(sentences)
word_index = tokenizer.word_index
print(word_index)
```

Tensorflow and keras give us a number of ways to encode words, but the one I'm going to focus on is the tokenizer. This will handle the heavy lifting for us, generating the dictionary of word encodings and creating vectors out of the sentences.

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

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

tokenizer = Tokenizer(num_words = 100)
tokenizer.fit_on_texts(sentences)
word_index = tokenizer.word_index
print(word_index)
```

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

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

tokenizer = Tokenizer(num_words = 100)
tokenizer.fit_on_texts(sentences)
word_index = tokenizer.word_index
print(word_index)
```

I then create an instance of the tokenizer. A passive parameter num wards to it. In this case, I'm using 100 which is way too big, as there are only five distinct words in this data. If you're creating a training set based on lots of text, you usually don't know how many unique distinct words there are in that text.

So by setting this hyperparameter, what the tokenizer will do is take the top 100 words by volume and just encode those. It's a handy shortcut when dealing with lots of data, and worth experimenting with when you train with real data later in this course.

Sometimes the impact of less words can be minimal and training accuracy, but huge in training time,

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

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

tokenizer = Tokenizer(num_words = 100)
tokenizer.fit_on_texts(sentences)
word_index = tokenizer.word_index
print(word_index)
```

The fit on texts method of the tokenizer then takes in the data and encodes it.

Strips Punctuation

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

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

tokenizer = Tokenizer(num_words = 100)
tokenizer.fit_on_texts(sentences)
word_index = tokenizer.word_index
print(word_index)
```

The tokenizer provides a word index property which returns a dictionary containing key value pairs, where the key is the word, and the value is the token for that word,

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

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

tokenizer = Tokenizer(num_words = 100)
tokenizer.fit_on_texts(sentences)
word_index = tokenizer.word_index
print(word_index)
```

```
{'i': 1, 'my': 3, 'dog': 4, 'cat': 5, 'love': 2}
```

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

```
{'i': 3, 'my': 2, 'you': 6, 'love': 1, 'cat': 5, 'dog': 4}
```

```
{'i': 3, 'my': 2, 'you': 6, 'love': 1, 'cat': 5, 'dog': 4}
```

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

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

tokenizer = Tokenizer(num_words = 100)
tokenizer.fit_on_texts(sentences)
word_index = tokenizer.word_index

sequences = tokenizer.texts_to_sequences(sentences)

print(word_index)
print(sequences)
```

Text to sequence

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

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

tokenizer = Tokenizer(num_words = 100)
tokenizer.fit_on_texts(sentences)
word_index = tokenizer.word_index

sequences = tokenizer.texts_to_sequences(sentences)

print(word_index)
print(sequences)
```

The next piece of code is this one, where I simply call on the tokenizer to get texts to sequences, and it will turn them into a set of sequences for me. So if I run this code, this will be the output.

```
{'amazing': 10, 'dog': 3, 'you': 5, 'cat': 6,
 'think': 8, 'i': 4, 'is': 9, 'my': 1, 'do': 7,
 'love': 2}
```

At the top is the new dictionary. With new tokens for my new words like amazing, think, is, and do.

```
[[4, 2, 1, 3], [4, 2, 1, 6], [5, 2, 1, 3], [7, 5,
8, 1, 3, 9, 10]]
```

```
{'amazing': 10, 'dog': 3, 'you': 5, 'cat': 6,
 'think': 8, 'i': 4, 'is': 9, 'my': 1, 'do': 7,
 'love': 2}
```

```
[[4, 2, 1, 3], [4, 2, 1, 6], [5, 2, 1, 3], [7, 5,
8, 1, 3, 9, 10]]
```

At the bottom is my list of sentences that have been encoded into integer lists, with the tokens replacing the words. So for example, I love my dog becomes 4, 2, 1, 3.

```python
from tensorflow.keras.preprocessing.text import Tokenizer

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

tokenizer = Tokenizer(num_words = 100)
tokenizer.fit_on_texts(sentences)
word_index = tokenizer.word_index

sequences = tokenizer.texts_to_sequences(sentences)

print(word_index)
print(sequences)
```

One really handy thing about this that you'll use later is the fact that the text to sequences called can take any set of sentences, so it can encode them based on the word set that it learned from the one that was passed into fit on texts.

This is very significant if you think ahead a little bit. If you train a neural network on a corpus of texts, and the text has a word index generated from it, then when you want to do inference with the train model, you'll have to encode the text that you want to infer on with the same word index, otherwise it would be meaningless.

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

test_seq = tokenizer.texts_to_sequences(test_data)
print(test_seq)
```

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

test_seq = tokenizer.texts_to_sequences(test_data)
print(test_seq)
```

```
[[4, 2, 1, 3], [1, 3, 1]]

{'think': 8, 'amazing': 10, 'my': 1, 'love': 2, 'dog': 3, 'is': 9, 'you': 5, 'do': 7,
'cat': 6, 'i': 4}
```

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

test_seq = tokenizer.texts_to_sequences(test_data)
print(test_seq)
```

```
[[4, 2, 1, 3], [1, 3, 1]]

{'think': 8, 'amazing': 10, 'my': 1, 'love': 2, 'dog': 3, 'is': 9, 'you': 5, 'do': 7,
'cat': 6, 'i': 4}
```

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

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

tokenizer = Tokenizer(num_words = 100, oov_token="<OOV>")
tokenizer.fit_on_texts(sentences)
word_index = tokenizer.word_index

sequences = tokenizer.texts_to_sequences(sentences)

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

test_seq = tokenizer.texts_to_sequences(test_data)
print(test_seq)
```

In many cases, it's a good idea to instead of just ignoring unseen words, to put a special value in when an unseen word is encountered. You can do this with a property on the tokenizer.

Let's take a look. Here's the complete code showing both the original sentences and the test data. What I've changed is to add a property oov token to the tokenizer constructor. You can see now that I've specified that I want the token oov for outer vocabulary to be used for words that aren't in the word index.

You can use whatever you like here, but remember that it should be something unique and distinct that isn't confused with a real word

```
[[5, 1, 3, 2, 4], [2, 4, 1, 2, 1]]


{'think': 9, 'amazing': 11, 'dog': 4, 'do': 8, 'i': 5, 'cat': 7,
 'you': 6, 'love': 3, '<OOV>': 1, 'my': 2, 'is': 10}
```

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

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

tokenizer = Tokenizer(num_words = 100, oov_token="<OOV>")
tokenizer.fit_on_texts(sentences)
word_index = tokenizer.word_index

sequences = tokenizer.texts_to_sequences(sentences)

padded = pad_sequences(sequences)
print(word_index)
print(sequences)
print(padded)
```

In order to use the padding functions you'll have to import pad sequences from tensorflow.carastoppreprocessing.sequence

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

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

tokenizer = Tokenizer(num_words = 100, oov_token="<OOV>")
tokenizer.fit_on_texts(sentences)
word_index = tokenizer.word_index

sequences = tokenizer.texts_to_sequences(sentences)

padded = pad_sequences(sequences)
print(word_index)
print(sequences)
print(padded)
```

Once the tokenizer has created the sequences, these sequences can be passed to pad sequences in order to have them padded like this.

```
{'do': 8, 'you': 6, 'love': 3, 'i': 5, 'amazing': 11, 'my': 2, 'is': 10, 'think': 9,
'dog': 4, '<OOV>': 1, 'cat': 7}

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


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

You can now see that the list of sentences has been padded out into a matrix and that each row in the matrix has the same length. It achieved this by putting the appropriate number of zeros before the sentence.

```
{'do': 8, 'you': 6, 'love': 3, 'i': 5, 'amazing': 11, 'my': 2, 'is': 10, 'think': 9,
'dog': 4, '<OOV>': 1, 'cat': 7}

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


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

```
padded = pad_sequences(sequences, padding='post')
```

Often you'll see examples where the padding is after the sentence and not before as you just saw. If you, like me, are more comfortable with that, you can change the code to this, adding the parameter padding equals post.

```
padded = pad_sequences(sequences, padding='post', maxlen=5)
```

You may have noticed that the matrix width was the same as the longest sentence. But you can override that with the maxlen parameter. So for example if you only want your sentences to have a maximum of five words. You can say maxlen equals five like this.

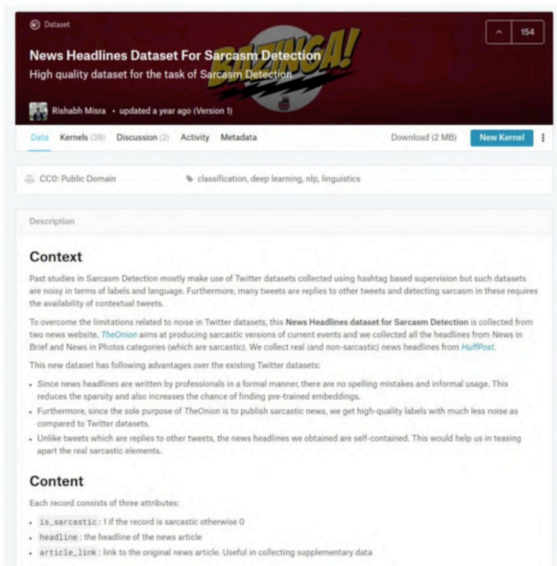
This of course will lead to the question. If I have sentences longer than the maxlength, then I'll lose information but from where. Like with the padding the default is pre, which means that you will lose from the beginning of the sentence.

If you want to override this so that you lose from the end instead, you can do so with the truncating parameter like this.

```
padded = pad_sequences(sequences, padding='post',
                       truncating='post', maxlen=5)
```

Sarcasm in News Headlines Dataset by Rishabh Misra

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



Dataset

154

News Headlines Dataset For Sarcasm Detection

High quality dataset for the task of Sarcasm Detection

BAZINGA!

Rishabh Misra • updated a year ago (Version 1)

Data Kernels (39) Discussion (2) Activity Metadata Download (2 MB) New Kernel

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-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}
```


To make it much easier to load this data into Python, I made a little tweak to the data to look like this


```
[
{"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'])
```

This allows you to load data in JSON format and automatically create a Python data structure from it.

```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'])
```

To do that you simply open the file, and pass it to json.load and you'll get a list containing lists of the three types of data: headlines, URLs, and is_sarcastic labels.

```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'])
```

Because I want the sentences as a list of their own to pass to the tokenizer, I can then create a list of sentences and later, if I want the labels for creating a neural network, I can create a list of them too. While I'm at it, I may as well do URLs even though I'm not going to use them here but you might want to.

```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'])
```

Now I can iterate through the list that was created with a for item in data store loop. For each item, I can then copy the headline to my sentences, the is_sarcastic to my labels and the article_link to my URLs.

```python
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
tokenizer = Tokenizer(oov_token="<OOV>")
tokenizer.fit_on_texts(sentences)
word_index = tokenizer.word_index

sequences = tokenizer.texts_to_sequences(sentences)
padded = pad_sequences(sequences, padding='post')
print(padded[0])
print(padded.shape)
```

We've just created sentences less from the headlines, in the sarcasm data set. So by calling tokenizer.fit on texts, will generate the word index and we'll initialize the tokenizer.

```python
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
tokenizer = Tokenizer(oov_token="<OOV>")
tokenizer.fit_on_texts(sentences)
word_index = tokenizer.word_index

sequences = tokenizer.texts_to_sequences(sentences)
padded = pad_sequences(sequences, padding='post')
print(padded[0])
print(padded.shape)
```

We can see the word index as before by calling the word index property. Note that this returns all words that the tokenizer saw when tokenizing the sentences. If you specify num words to get the top 1000 or whatever, you may be confused by seeing something greater than that here. It's an easy mistake to make.

The key thing to remember, is that when it takes the top 1000 or whatever you specified, it does that in the text to sequence this process.

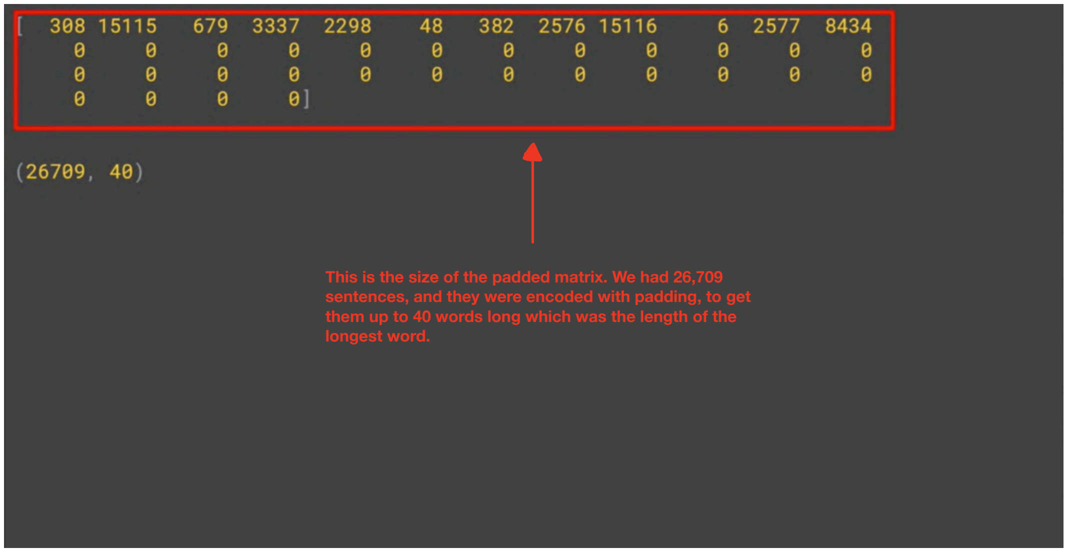
```
{'underwood': 24127, 'skillingsbolle': 23055, 'grabs': 12293, 'mobility': 8909,
"'assassin's": 12648, 'visualize': 23973, 'hurting': 4992, 'orphaned': 9173,
"'agreed'": 24365, 'narration': 28470
```

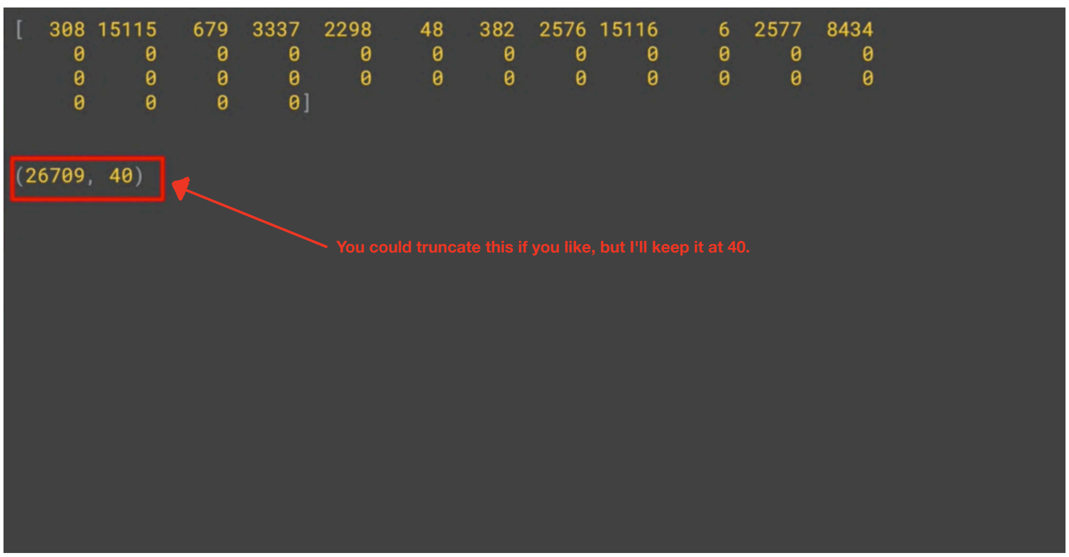
Our word index is much larger than with the previous example. So we'll see a greater variety of words in it.

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
tokenizer = Tokenizer(oov_token="<OOV>")
tokenizer.fit_on_texts(sentences)
word_index = tokenizer.word_index

sequences = tokenizer.texts_to_sequences(sentences)
padded = pad_sequences(sequences, padding='post')
print(padded[0])
print(padded.shape)
```

Create the sequences from the text, as well as padding them.

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
tokenizer = Tokenizer(oov_token="<OOV>")
tokenizer.fit_on_texts(sentences)
word_index = tokenizer.word_index

sequences = tokenizer.texts_to_sequences(sentences)
padded = pad_sequences(sequences, padding='post')
print(padded[0])
print(padded.shape)
```


[308 15115 679 3337 2298 48 382 2576 15116 6 2577 8434
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]
(26709, 40)
This is the size of the padded matrix. We had 26,709 sentences, and they were encoded with padding, to get them up to 40 words long which was the length of the longest word.


[308 15115 679 3337 2298 48 382 2576 15116 6 2577 8434
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]
(26709, 40)
You could truncate this if you like, but I'll keep it at 40.