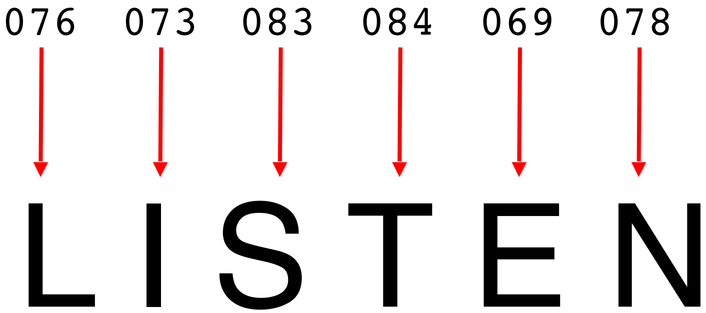
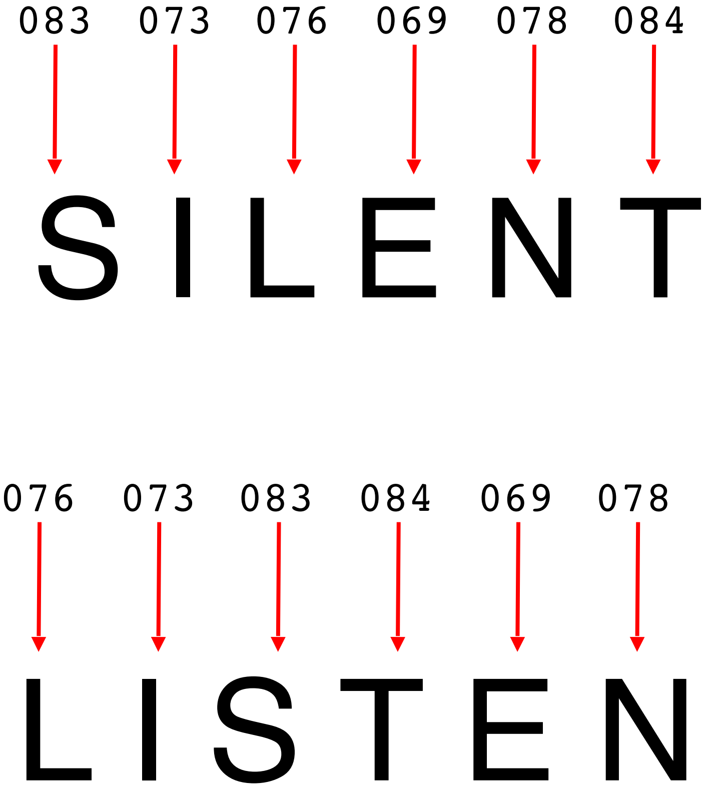
Copyright Notice

LISTEN



076 073 083 084 069 078

LISTEN



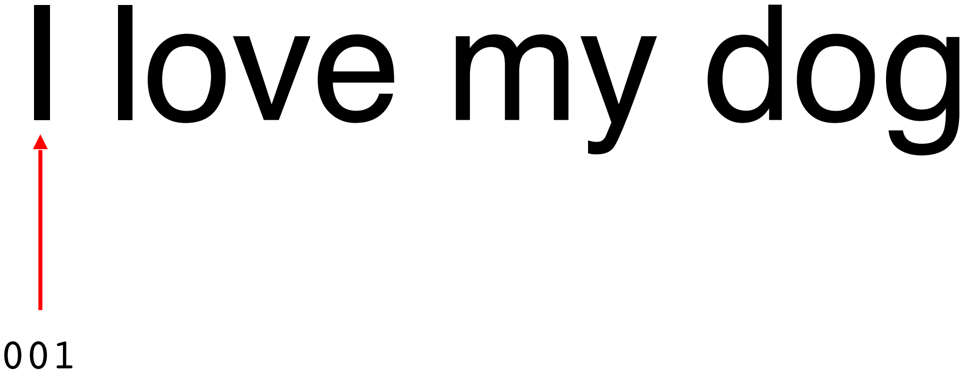
083 073 076 069 078 084

SILENT

076 073 083 084 069 078

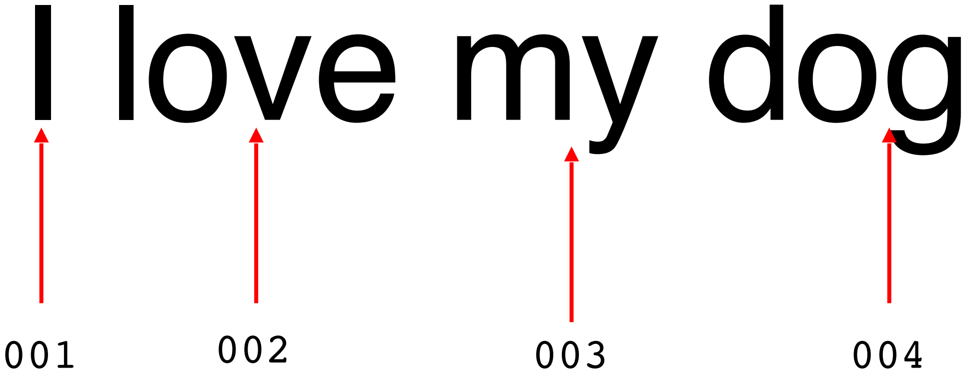
LISTEN

I love my dog



I love my dog

001



I love my dog

001

002

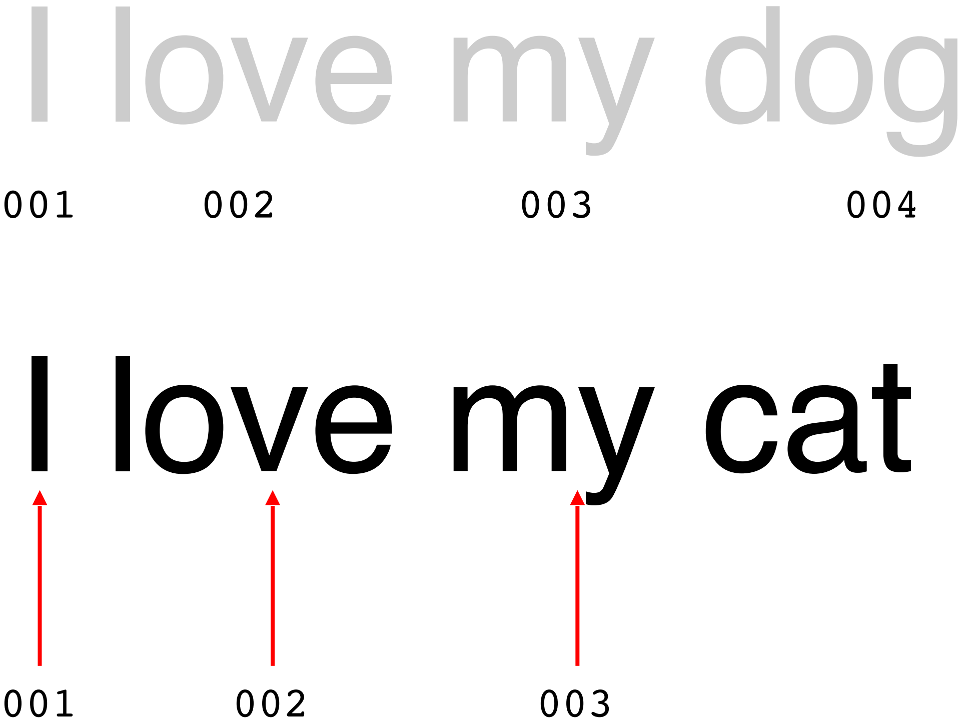
003

004

I love my dog

001 002 003 004

I love my cat

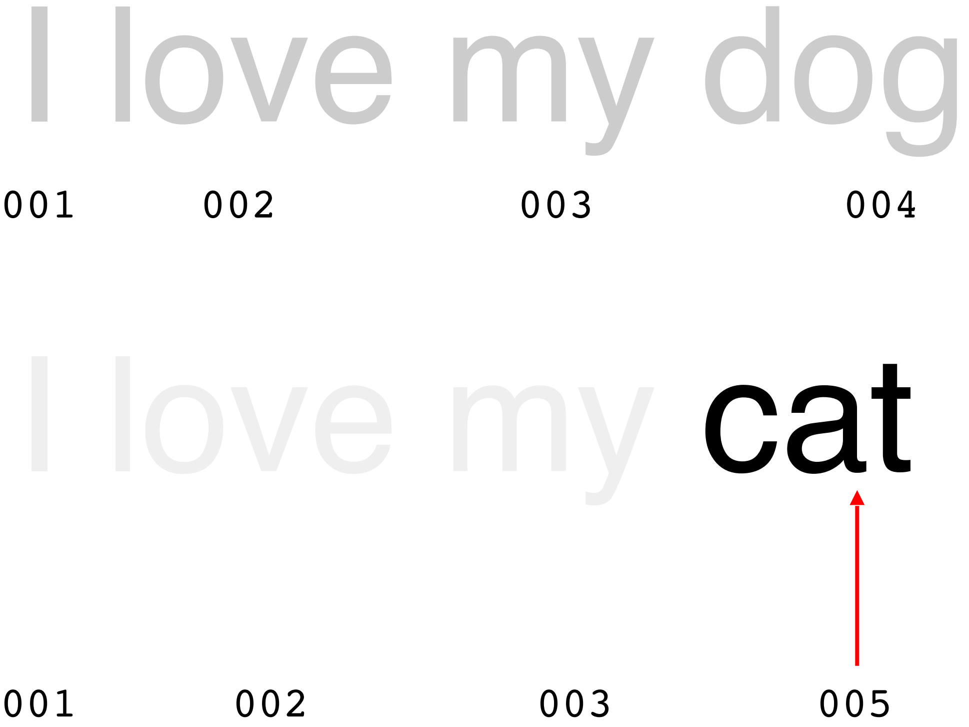


I love my dog

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

```
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)
```

```
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)
```

```
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)
```

```
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)
```

```
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}
```

```
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!'
]
```

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

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

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

```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)
```

```
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)
```

```
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)
```

```
{'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]]
```

```
{'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]]
```

```
{'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]]
```

```
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)
```

```
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}
```

```
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)
```

```
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)
```

```
[[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}
```

```python
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)
```

```
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)
```

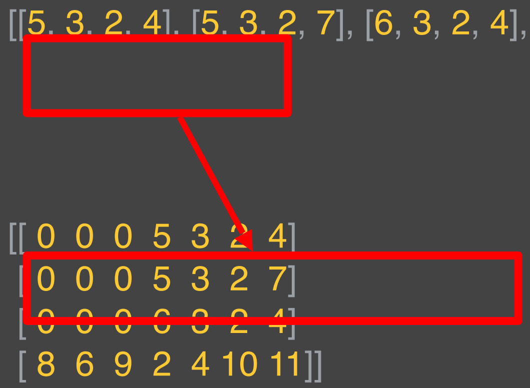
```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

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)
```

```
{'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]]
```

```
{'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]]
```

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

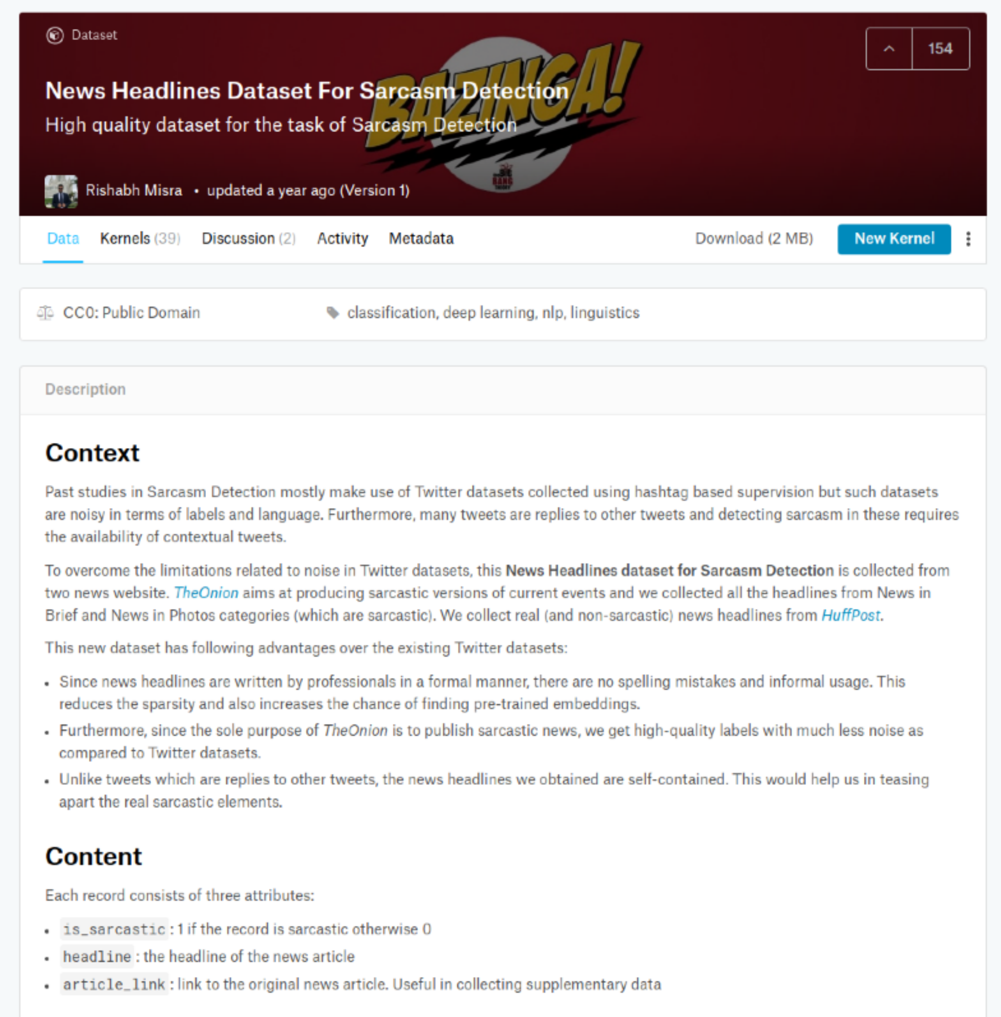
```
[[5, 3, 2, 4], [5, 3, 2, 7], [6, 3, 2, 4], [8, 6, 9, 2, 4, 10, 11]]
```

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

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-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}

```
[

{"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'])
```

```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'])
```

```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'])
```

```
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)
```

```
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)
```

```
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)
```

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

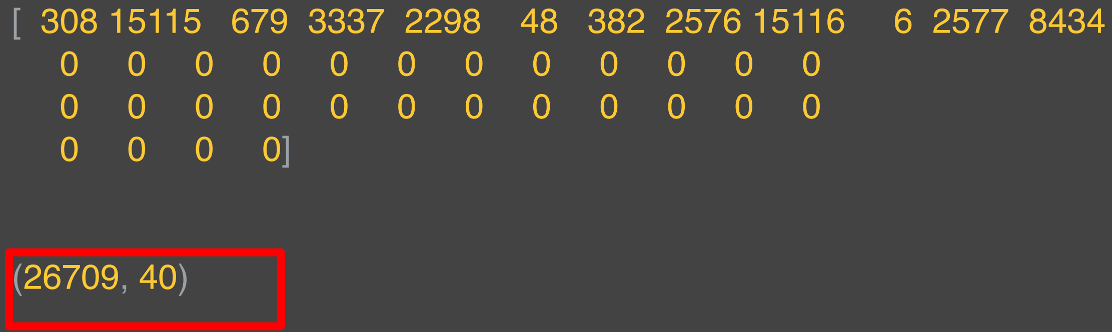
```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
tokenizer = Tokenizer(oov_token="<OOV>")
tokenizer.fit_on_texts(sentences)
word_index = tokenizer.word_index

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

```
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)
```

```
[  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)
```



```
[  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)
```

Copyright Notice

audio

"nsynth"

image

"abstract_reasoning"
"caltech101"
"cats_vs_dogs"
"celeb_a"
"celeb_a_hq"
"cifar10"
"cifar100"
"cifar10_corrupted"
"coco2014"
"colorectal_histology"
"cycle_gan"
"diabetic_retinopathy..."
"dsprites"
"dtd"
"emnist"
"fashion_mnist"
"horses_or_humans"
"image_label_folder"
"imagenet2012"
"imagenet2012_corrupted"
"kmnist"
"lsun"
"mnist"
"omniglot"
"open_images_v4"
"oxford_iiit_pet"
"quickdraw_bitmap"
"rock_paper_scissors"
"shapes3d"
"smallnorb"
"sun397"
"svhn_cropped"
"tf_flowers"

structured

"higgs"
"iris"
"titanic"

text

"cnn_dailymail"
"glue"
"imdb_reviews"
"lm1b"
"multi_nli"
"squad"
"wikipedia"
"xnli"

translate

"flores"
"para_crawl"
"ted_hrlr_translate"
"ted_multi_translate"
"wmt15_translate"
"wmt16_translate"
"wmt17_translate"
"wmt18_translate"
"wmt19_translate"

audio

"nsynth"

image

"abstract_reasoning"
"caltech101"
"cats_vs_dogs"
"celeb_a"
"celeb_a_hq"
"cifar10"
"cifar100"
"cifar10_corrupted"
"coco2014"
"colorectal_histology"
"cycle_gan"
"diabetic_retinopathy..."
"dsprites"
"dtd"
"emnist"
"fashion_mnist"
"horses_or_humans"
"image_label_folder"
"imagenet2012"
"imagenet2012_corrupted"
"kmnist"
"lsun"
"mnist"
"omniglot"
"open_images_v4"
"oxford_iiit_pet"
"quickdraw_bitmap"
"rock_paper_scissors"
"shapes3d"
"smallnorb"
"sun397"
"svhn_cropped"
"tf_flowers"

structured

"higgs"
"iris"
"titanic"

text

"cnn_dailymail"
"glue"
"imdb_reviews"
"lm1b"
"multi_nli"
"squad"
"wikipedia"
"xnli"

translate

"flores"
"para_crawl"
"ted_hrlr_translate"
"ted_multi_translate"
"wmt15_translate"
"wmt16_translate"
"wmt17_translate"
"wmt18_translate"
"wmt19_translate"

http://ai.stanford.edu/~amaas/data/sentiment/

```
@InProceedings{maas-EtAl:2011:ACL-HLT2011,
  author    = {Maas, Andrew L.  and  Daly, Raymond E.  and  Pham, Peter T.  and  Huang, Dan  and
Ng, Andrew Y.  and  Potts, Christopher},
  title     = {Learning Word Vectors for Sentiment Analysis},
  booktitle = {Proceedings of the 49th Annual Meeting of the Association for Computational
Linguistics: Human Language Technologies},
  month     = {June},
  year      = {2011},
  address   = {Portland, Oregon, USA},
  publisher = {Association for Computational Linguistics},
  pages     = {142--150},
  url       = {http://www.aclweb.org/anthology/P11-1015}
}
```

```
import tensorflow as tf
print(tf.__version__)
```

```
import tensorflow_datasets as tfds
imdb, info = tfds.load("imdb_reviews", with_info=True, as_supervised=True)
```

```python
import numpy as np

train_data, test_data = imdb['train'], imdb['test']
```

```
training_sentences = []
training_labels = []

testing_sentences = []
testing_labels = []

for s,l in train_data:
  training_sentences.append(str(s.numpy()))
  training_labels.append(l.numpy())

for s,l in test_data:
  testing_sentences.append(str(s.numpy()))
  testing_labels.append(l.numpy())
```

```
training_sentences = []
training_labels = []

testing_sentences = []
testing_labels = []

for s,l in train_data:
  training_sentences.append(str(s.numpy()))
  training_labels.append(l.numpy())

for s,l in test_data:
  testing_sentences.append(str(s.numpy()))
  testing_labels.append(l.numpy())
```

```
training_sentences = []
training_labels = []

testing_sentences = []
testing_labels = []

for s,l in train_data:
  training_sentences.append(str(s.numpy()))
  training_labels.append(l.numpy())

for s,l in test_data:
  testing_sentences.append(str(s.numpy()))
  testing_labels.append(l.numpy())
```

```python
training_sentences = []
training_labels = []

testing_sentences = []
testing_labels = []

for s,l in train_data:
  training_sentences.append(str(s.numpy()))
  training_labels.append(l.numpy())

for s,l in test_data:
  testing_sentences.append(str(s.numpy()))
  testing_labels.append(l.numpy())
```

```
tf.Tensor(b"As a lifelong fan of Dickens, I have invariably been disappointed
by adaptations of his novels.<br /><br />Although his works presented an
extremely accurate re-telling of human life at every level in Victorian Britain,
throughout them all was a pervasive thread of humour that could be both playful
or sarcastic as the narrative dictated. In a way, he was a literary
caricaturist and cartoonist. He could be serious and hilarious in the same
sentence. He pricked pride, lampooned arrogance, celebrated modesty,
and empathised with loneliness and poverty. It may be a clich\xc3\xa9, but
he was a people's writer.<br /><br />And it is the comedy that is so often
missing from his interpretations. At the time of writing, Oliver Twist
is being dramatised in serial form on BBC television. All of the misery
and cruelty is their, but non of the humour, irony, and savage lampoonery.",
shape=(), dtype=string)
```

```
tf.Tensor(1, shape=(), dtype=int64)
tf.Tensor(1, shape=(), dtype=int64)
tf.Tensor(1, shape=(), dtype=int64)
tf.Tensor(0, shape=(), dtype=int64)
tf.Tensor(0, shape=(), dtype=int64)
tf.Tensor(1, shape=(), dtype=int64)
```

```
training_labels_final = np.array(training_labels)
testing_labels_final = np.array(testing_labels)
```

```
vocab_size = 10000
embedding_dim = 16
max_length = 120
trunc_type='post'
oov_tok = "<OOV>"

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

tokenizer = Tokenizer(num_words = vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(training_sentences)
word_index = tokenizer.word_index
sequences = tokenizer.texts_to_sequences(training_sentences)
padded = pad_sequences(sequences,maxlen=max_length, truncating=trunc_type)

testing_sequences = tokenizer.texts_to_sequences(testing_sentences)
testing_padded = pad_sequences(testing_sequences,maxlen=max_length)
```

```
vocab_size = 10000
embedding_dim = 16
max_length = 120
trunc_type='post'
oov_tok = "<OOV>"

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

tokenizer = Tokenizer(num_words = vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(training_sentences)
word_index = tokenizer.word_index
sequences = tokenizer.texts_to_sequences(training_sentences)
padded = pad_sequences(sequences,maxlen=max_length, truncating=trunc_type)

testing_sequences = tokenizer.texts_to_sequences(testing_sentences)
testing_padded = pad_sequences(testing_sequences,maxlen=max_length)
```

```
vocab_size = 10000
embedding_dim = 16
max_length = 120
trunc_type='post'
oov_tok = "<OOV>"

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

tokenizer = Tokenizer(num_words = vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(training_sentences)
word_index = tokenizer.word_index
sequences = tokenizer.texts_to_sequences(training_sentences)
padded = pad_sequences(sequences,maxlen=max_length, truncating=trunc_type)

testing_sequences = tokenizer.texts_to_sequences(testing_sentences)
testing_padded = pad_sequences(testing_sequences,maxlen=max_length)
```

```
vocab_size = 10000
embedding_dim = 16
max_length = 120
trunc_type='post'
oov_tok = "<OOV>"

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

tokenizer = Tokenizer(num_words = vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(training_sentences)
word_index = tokenizer.word_index
sequences = tokenizer.texts_to_sequences(training_sentences)
padded = pad_sequences(sequences,maxlen=max_length, truncating=trunc_type)

testing_sequences = tokenizer.texts_to_sequences(testing_sentences)
testing_padded = pad_sequences(testing_sequences,maxlen=max_length)
```

```python
vocab_size = 10000
embedding_dim = 16
max_length = 120
trunc_type='post'
oov_tok = "<OOV>"

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

tokenizer = Tokenizer(num_words = vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(training_sentences)
word_index = tokenizer.word_index
sequences = tokenizer.texts_to_sequences(training_sentences)
padded = pad_sequences(sequences,maxlen=max_length, truncating=trunc_type)

testing_sequences = tokenizer.texts_to_sequences(testing_sentences)
testing_padded = pad_sequences(testing_sequences,maxlen=max_length)
```

```
vocab_size = 10000
embedding_dim = 16
max_length = 120
trunc_type='post'
oov_tok = "<OOV>"

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

tokenizer = Tokenizer(num_words = vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(training_sentences)
word_index = tokenizer.word_index
sequences = tokenizer.texts_to_sequences(training_sentences)
padded = pad_sequences(sequences,maxlen=max_length, truncating=trunc_type)

testing_sequences = tokenizer.texts_to_sequences(testing_sentences)
testing_padded = pad_sequences(testing_sequences,maxlen=max_length)
```

```
vocab_size = 10000
embedding_dim = 16
max_length = 120
trunc_type='post'
oov_tok = "<OOV>"

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

tokenizer = Tokenizer(num_words = vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(training_sentences)
word_index = tokenizer.word_index
sequences = tokenizer.texts_to_sequences(training_sentences)
padded = pad_sequences(sequences,maxlen=max_length, truncating=trunc_type)

testing_sequences = tokenizer.texts_to_sequences(testing_sentences)
testing_padded = pad_sequences(testing_sequences,maxlen=max_length)
```

```
vocab_size = 10000
embedding_dim = 16
max_length = 120
trunc_type='post'
oov_tok = "<OOV>"

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

tokenizer = Tokenizer(num_words = vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(training_sentences)
word_index = tokenizer.word_index
sequences = tokenizer.texts_to_sequences(training_sentences)
padded = pad_sequences(sequences,maxlen=max_length, truncating=trunc_type)

testing_sequences = tokenizer.texts_to_sequences(testing_sentences)
testing_padded = pad_sequences(testing_sequences,maxlen=max_length)
```

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(6, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(6, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(6, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(6, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_9 (Embedding)      (None, 120, 16)           160000
_________________________________________________________________
flatten_3 (Flatten)          (None, 1920)              0
_________________________________________________________________
dense_14 (Dense)             (None, 6)                 11526
_________________________________________________________________
dense_15 (Dense)             (None, 1)                 7
=================================================================
Total params: 171,533
Trainable params: 171,533
Non-trainable params: 0
```

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(6, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_11 (Embedding)     (None, 120, 16)           160000
_________________________________________________________________
global_average_pooling1d_3 ( (None, 16)                0
_________________________________________________________________
dense_16 (Dense)             (None, 6)                 102
_________________________________________________________________
dense_17 (Dense)             (None, 1)                 7
=================================================================
Total params: 160,109
Trainable params: 160,109
Non-trainable params: 0
```

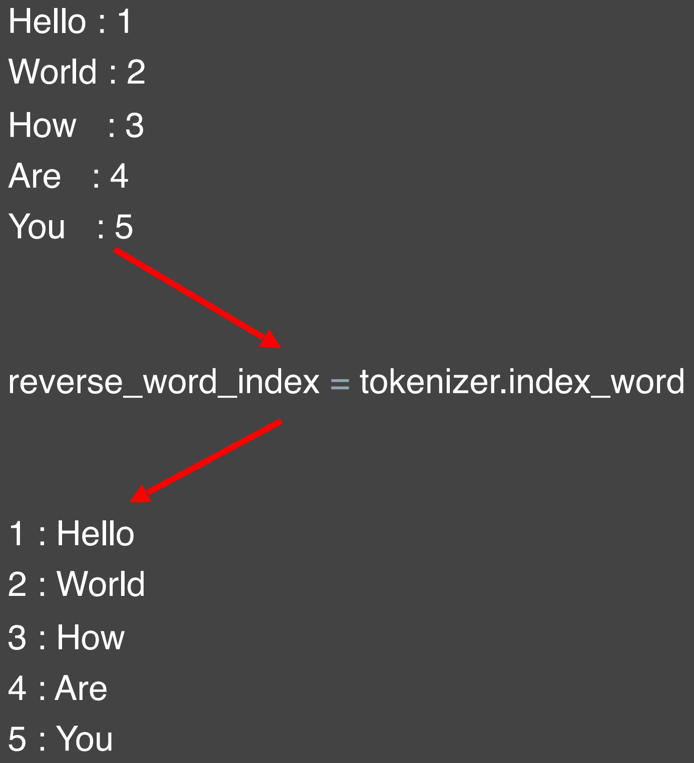
```
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
model.summary()
```

```
num_epochs = 10
model.fit(padded,
          training_labels_final,
          epochs=num_epochs,
          validation_data=(testing_padded, testing_labels_final))
```

```
Epoch 8/10
25000/25000 [==============================] -
6s 256us/sample - loss: 5.2086e-04 - acc: 1.0000 - val_loss: 0.7252 - val_acc: 0.8270
Epoch 9/10
25000/25000 [==============================] -
6s 222us/sample - loss: 3.0199e-04 - acc: 1.0000 - val_loss: 0.7628 - val_acc: 0.8269
Epoch 10/10
25000/25000 [==============================] -
6s 224us/sample - loss: 1.7872e-04 - acc: 1.0000 - val_loss: 0.7997 - val_acc: 0.8259
```

```
e = model.layers[0]
weights = e.get_weights()[0]
print(weights.shape) # shape: (vocab_size, embedding_dim)
```

```
(10000, 16)
```



Hello : 1

World : 2

How : 3

Are : 4

You : 5

```
reverse_word_index = tokenizer.index_word
```

1 : Hello

2 : World

3 : How

4 : Are

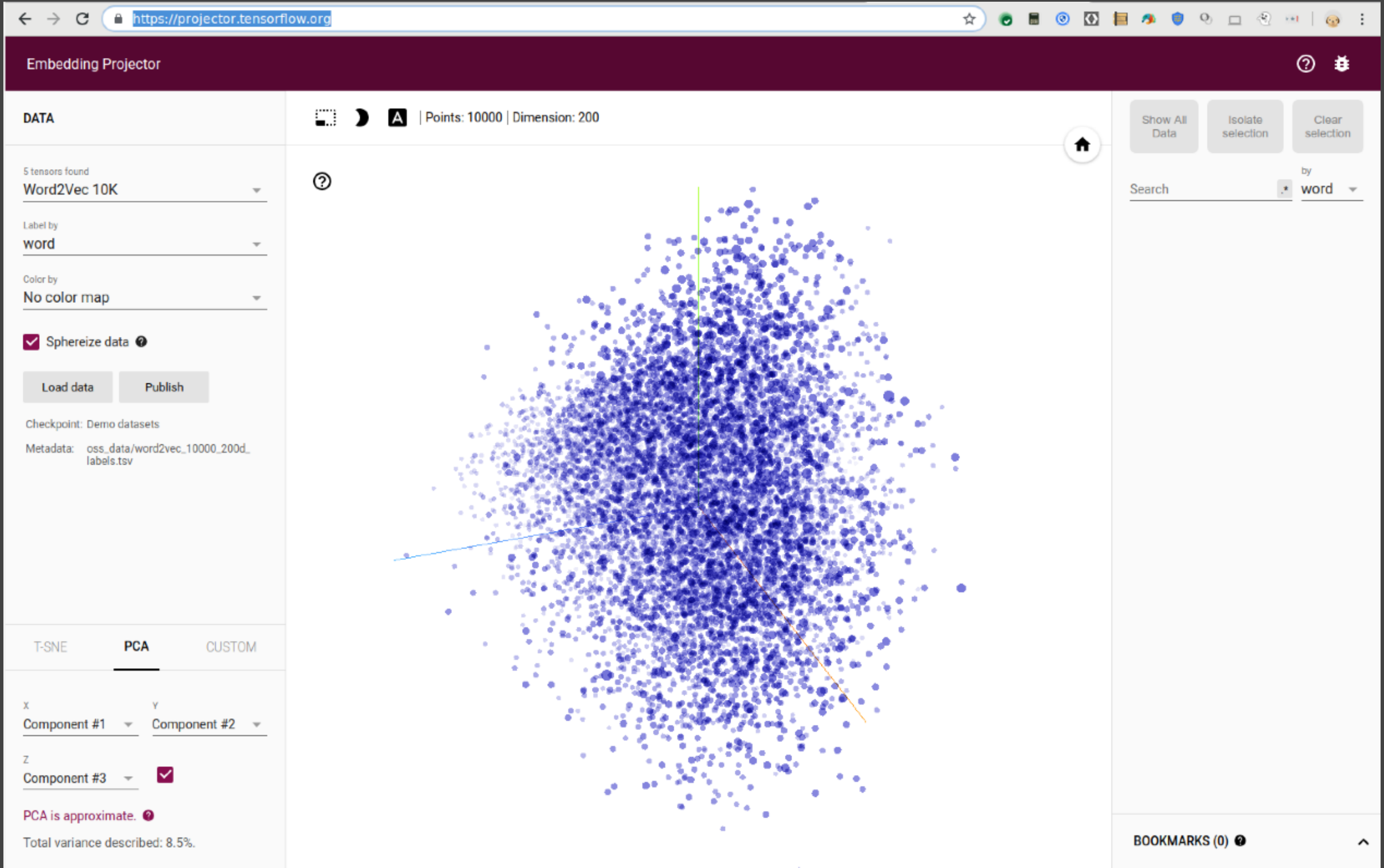
5 : You

```
import io

out_v = io.open('vecs.tsv', 'w', encoding='utf-8')
out_m = io.open('meta.tsv', 'w', encoding='utf-8')
for word_num in range(1, vocab_size):
  word = reverse_word_index[word_num]
  embeddings = weights[word_num]
  out_m.write(word + "\n")
  out_v.write('\t'.join([str(x) for x in embeddings]) + "\n")
out_v.close()
out_m.close()
```

```
import io

out_v = io.open('vecs.tsv', 'w', encoding='utf-8')
out_m = io.open('meta.tsv', 'w', encoding='utf-8')
for word_num in range(1, vocab_size):
  word = reverse_word_index[word_num]
  embeddings = weights[word_num]
  out_m.write(word + "\n")
  out_v.write('\t'.join([str(x) for x in embeddings]) + "\n")
out_v.close()
out_m.close()
```

```
import io

out_v = io.open('vecs.tsv', 'w', encoding='utf-8')
out_m = io.open('meta.tsv', 'w', encoding='utf-8')
for word_num in range(1, vocab_size):
  word = reverse_word_index[word_num]
  embeddings = weights[word_num]
  out_m.write(word + "\n")
  out_v.write('\t'.join([str(x) for x in embeddings]) + "\n")
out_v.close()
out_m.close()
```

```
try:
  from google.colab import files
except ImportError:
  pass
else:
  files.download('vecs.tsv')
  files.download('meta.tsv')
```



5 tensors found
Word2Vec 10K

Label by
word

Color by
No color map

☑ Sphereize data

Load data
Publish

Checkpoint: Demo datasets

Metadata: oss_data/word2vec_10000_200d_labels.tsv

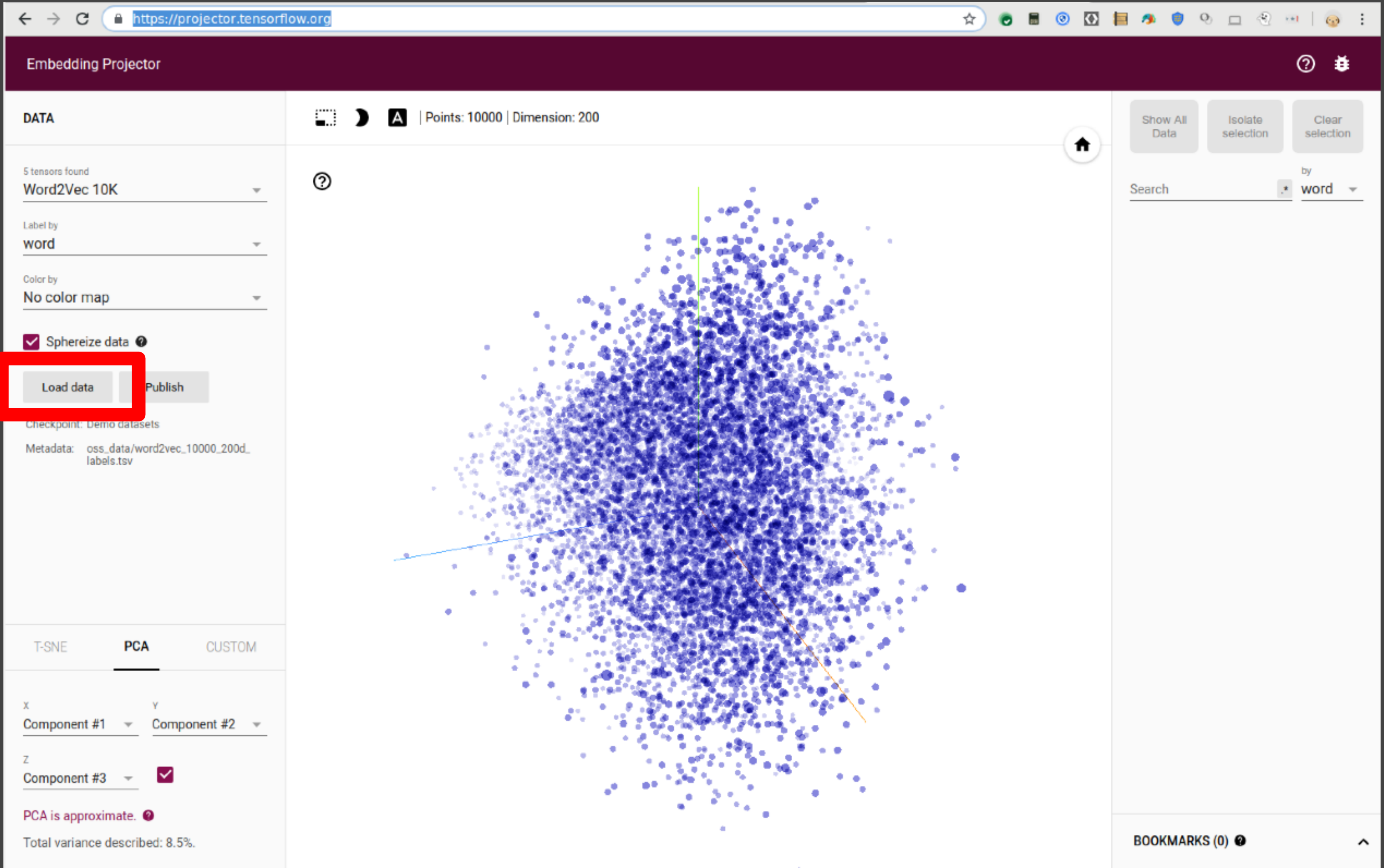
X
Component #1

Y
Component #2

Z
Component #3 ☑

PCA is approximate.

Total variance described: 8.5%.



Points: 10000 | Dimension: 200

5 tensors found
Word2Vec 10K

Label by
word

Color by
No color map

☑ Sphereize data

Checkpoint: Demo datasets

Metadata: oss_data/word2vec_10000_200d_labels.tsv

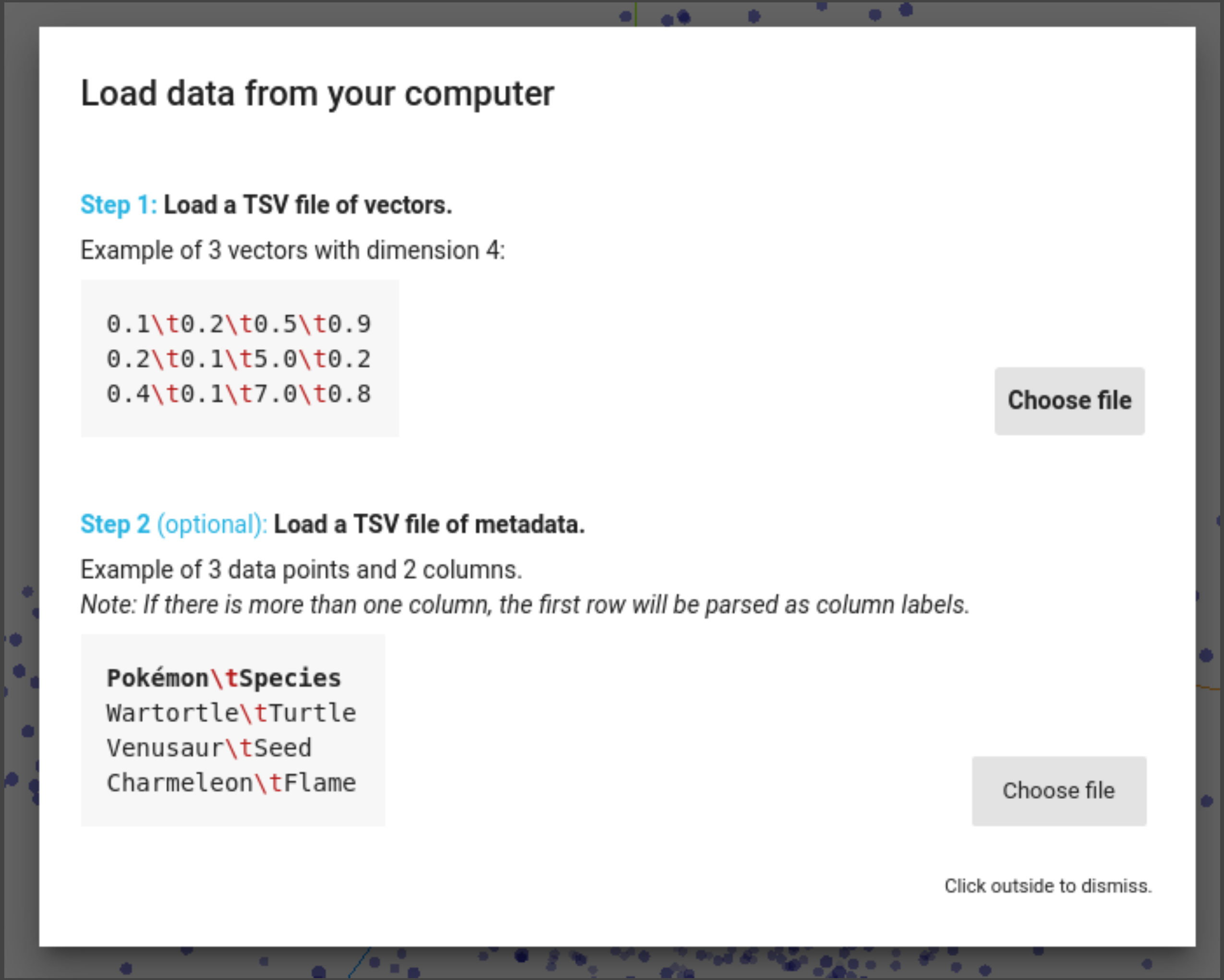
X
Component #1

Y
Component #2

Z
Component #3 ☑

PCA is approximate.

Total variance described: 8.5%.



Load data from your computer

Step 1: Load a TSV file of vectors.

Example of 3 vectors with dimension 4:

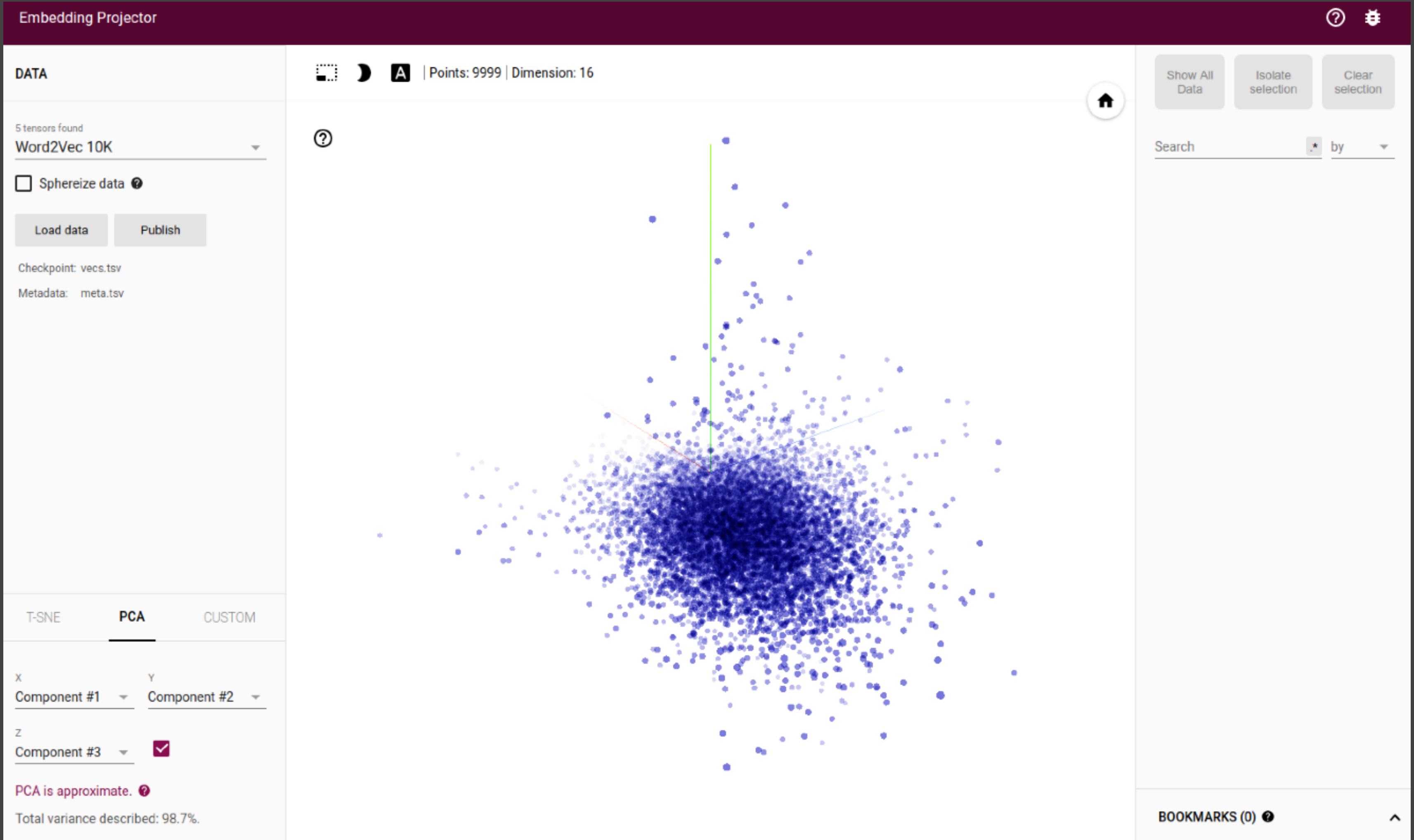
```
0.1\t0.2\t0.5\t0.9
0.2\t0.1\t5.0\t0.2
0.4\t0.1\t7.0\t0.8
```

Step 2 (optional): Load a TSV file of metadata.

Example of 3 data points and 2 columns.
Note: If there is more than one column, the first row will be parsed as column labels.

```
Pokémon\tSpecies
Wartortle\tTurtle
Venusaur\tSeed
Charmeleon\tFlame
```

Click outside to dismiss.



5 tensors found

Word2Vec 10K

□ Sphereize data

Checkpoint: vecs.tsv

Metadata: meta.tsv

Points: 9999 | Dimension: 16

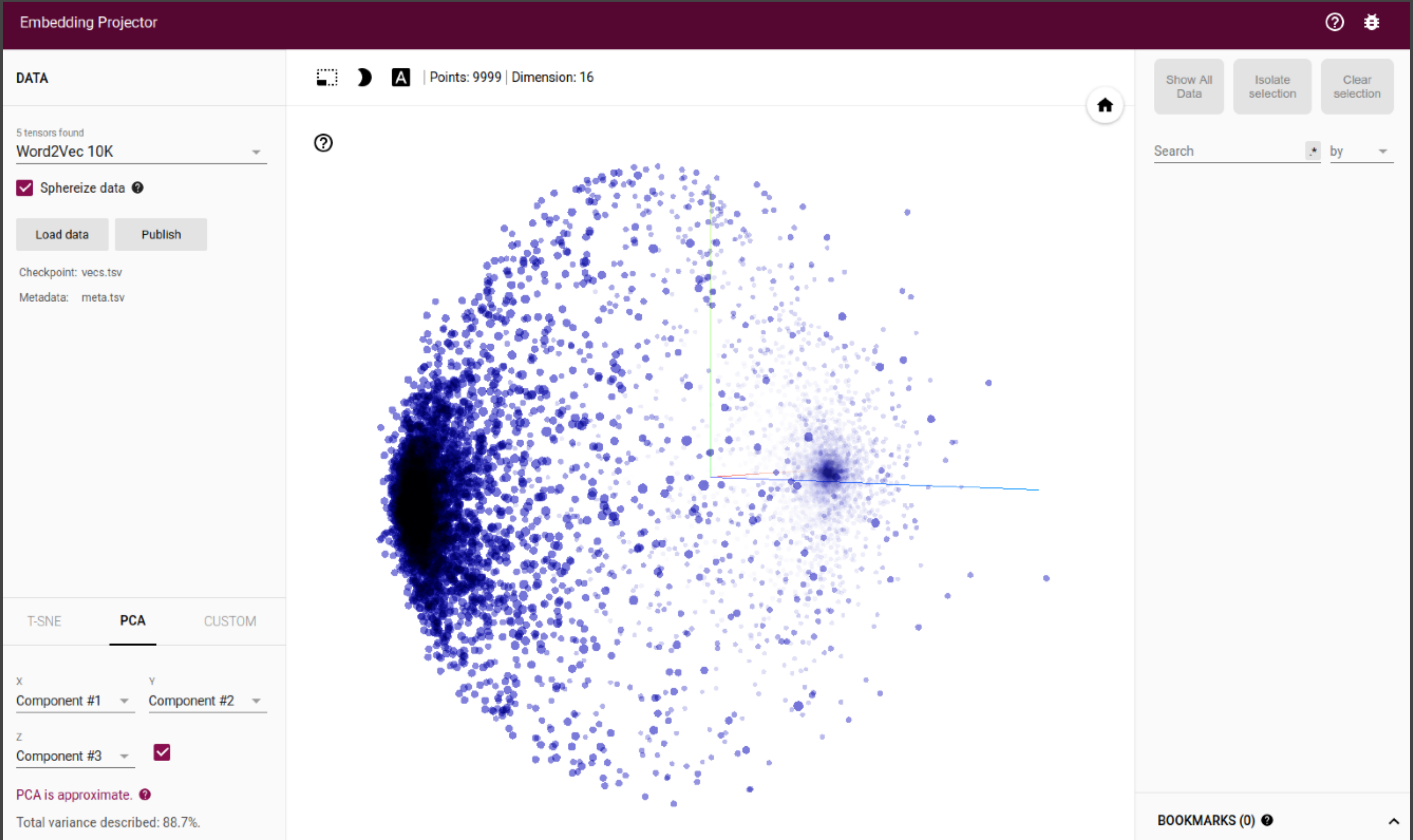
X Component #1

Y Component #2

Z Component #3 ☑

PCA is approximate.

Total variance described: 98.7%.



Points: 9999 | Dimension: 16

5 tensors found
Word2Vec 10K

☑ Sphereize data

Checkpoint: vecs.tsv

Metadata: meta.tsv

X
Component #1

Y
Component #2

Z
Component #3 ☑

PCA is approximate.

Total variance described: 88.7%.

```python
import json
import tensorflow as tf

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

```
vocab_size = 10000
embedding_dim = 16
max_length = 32
trunc_type='post'
padding_type='post'
oov_tok = "<OOV>"
training_size = 20000
```

```
!wget --no-check-certificate \
    https://storage.googleapis.com/laurencemoroney-blog.appspot.com/sarcasm.json \
    -O /tmp/sarcasm.json
```

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

sentences = []
labels = []

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

```
training_sentences = sentences[0:training_size]
testing_sentences = sentences[training_size:]
training_labels = labels[0:training_size]
testing_labels = labels[training_size:]
```

```
training_sentences = sentences[0:training_size]
testing_sentences = sentences[training_size:]
training_labels = labels[0:training_size]
testing_labels = labels[training_size:]
```

```
training_sentences = sentences[0:training_size]
testing_sentences = sentences[training_size:]
training_labels = labels[0:training_size]
testing_labels = labels[training_size:]
```

```
training_sentences = sentences[0:training_size]
testing_sentences = sentences[training_size:]
training_labels = labels[0:training_size]
testing_labels = labels[training_size:]
```

```
tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(training_sentences)

word_index = tokenizer.word_index

training_sequences = tokenizer.texts_to_sequences(training_sentences)
training_padded = pad_sequences(training_sequences, maxlen=max_length,
                  padding=padding_type, truncating=trunc_type)

testing_sequences = tokenizer.texts_to_sequences(testing_sentences)
testing_padded = pad_sequences(testing_sequences, maxlen=max_length,
                 padding=padding_type, truncating=trunc_type)
```

```
tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(training_sentences)

word_index = tokenizer.word_index

training_sequences = tokenizer.texts_to_sequences(training_sentences)
training_padded = pad_sequences(training_sequences, maxlen=max_length,
                   padding=padding_type, truncating=trunc_type)

testing_sequences = tokenizer.texts_to_sequences(testing_sentences)
testing_padded = pad_sequences(testing_sequences, maxlen=max_length,
                  padding=padding_type, truncating=trunc_type)
```

```
tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(training_sentences)

word_index = tokenizer.word_index

training_sequences = tokenizer.texts_to_sequences(training_sentences)
training_padded = pad_sequences(training_sequences, maxlen=max_length,
                                padding=padding_type, truncating=trunc_type)

testing_sequences = tokenizer.texts_to_sequences(testing_sentences)
testing_padded = pad_sequences(testing_sequences, maxlen=max_length,
                               padding=padding_type, truncating=trunc_type)
```

```
tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(training_sentences)

word_index = tokenizer.word_index

training_sequences = tokenizer.texts_to_sequences(training_sentences)
training_padded = pad_sequences(training_sequences, maxlen=max_length,
                                padding=padding_type, truncating=trunc_type)

testing_sequences = tokenizer.texts_to_sequences(testing_sentences)
testing_padded = pad_sequences(testing_sequences, maxlen=max_length,
                               padding=padding_type, truncating=trunc_type)
```

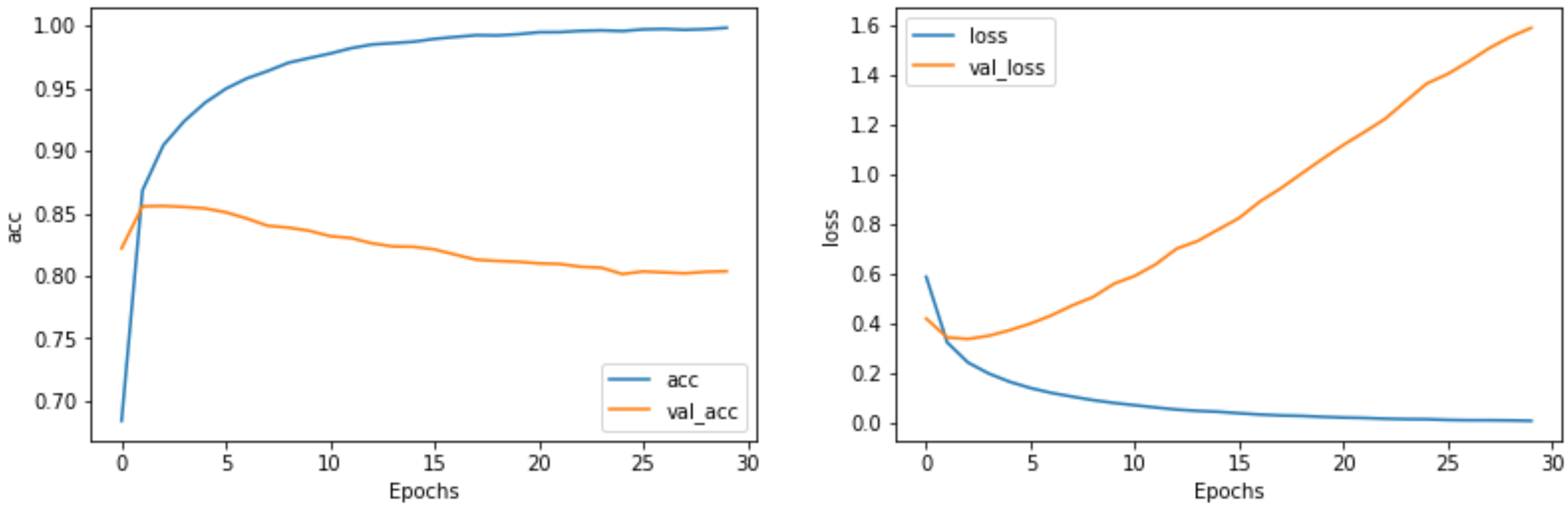
```
tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(training_sentences)

word_index = tokenizer.word_index

training_sequences = tokenizer.texts_to_sequences(training_sentences)
training_padded = pad_sequences(training_sequences, maxlen=max_length,
                                padding=padding_type, truncating=trunc_type)

testing_sequences = tokenizer.texts_to_sequences(testing_sentences)
testing_padded = pad_sequences(testing_sequences, maxlen=max_length,
                               padding=padding_type, truncating=trunc_type)
```

```
tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(training_sentences)

word_index = tokenizer.word_index

training_sequences = tokenizer.texts_to_sequences(training_sentences)
training_padded = pad_sequences(training_sequences, maxlen=max_length,
                                padding=padding_type, truncating=trunc_type)

testing_sequences = tokenizer.texts_to_sequences(testing_sentences)
testing_padded = pad_sequences(testing_sequences, maxlen=max_length,
                               padding=padding_type, truncating=trunc_type)
```

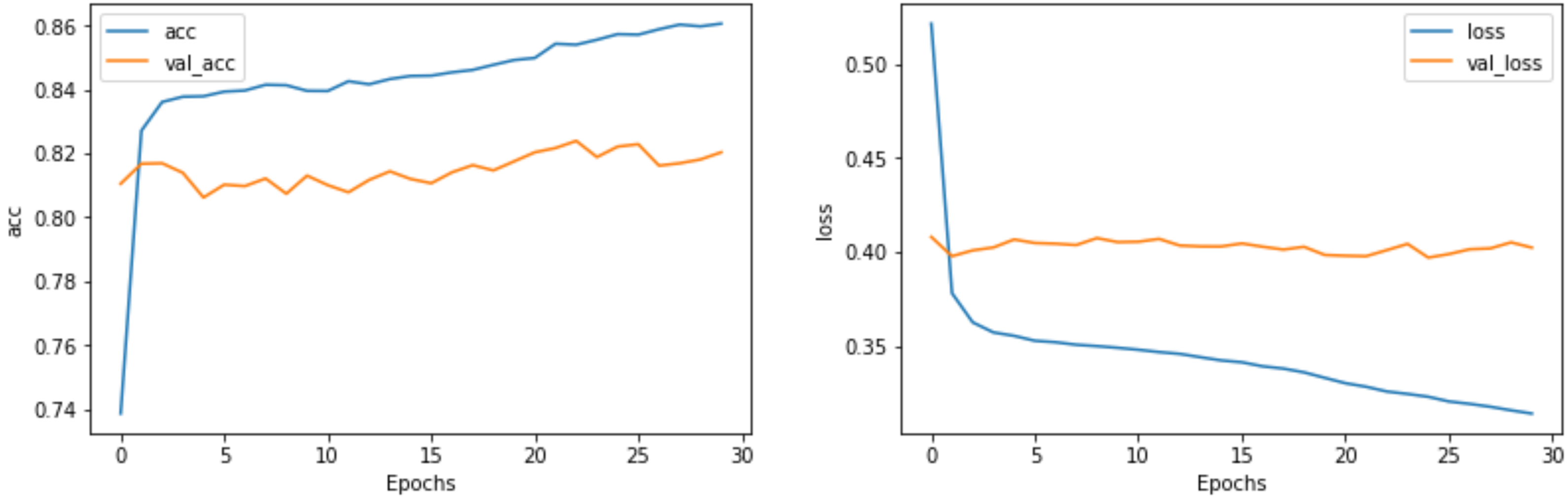
```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(24, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
model.summary()



Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 32, 16)            160000
_________________________________________________________________
global_average_pooling1d_2 ( (None, 16)                0
_________________________________________________________________
dense_4 (Dense)              (None, 24)                408
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 25
=================================================================
Total params: 160,433
Trainable params: 160,433
Non-trainable params: 0
```

```
num_epochs = 30

history = model.fit(training_padded, training_labels, epochs=num_epochs,
              validation_data=(testing_padded, testing_labels), verbose=2)
```

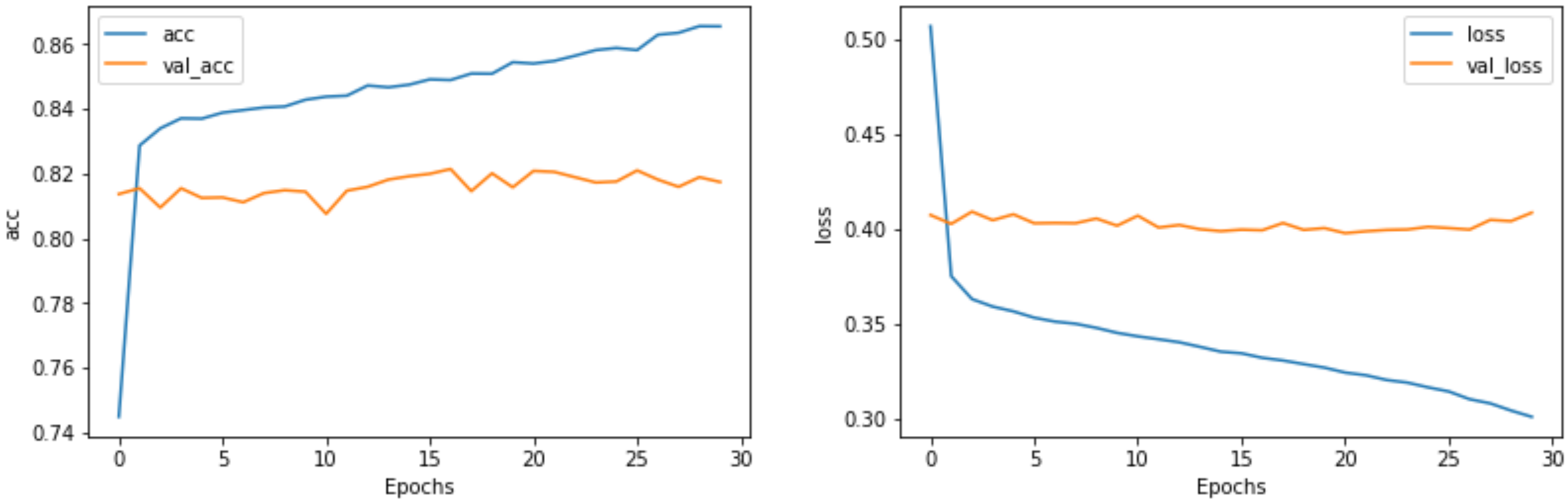
```
import matplotlib.pyplot as plt


def plot_graphs(history, string):
  plt.plot(history.history[string])
  plt.plot(history.history['val_'+string])
  plt.xlabel("Epochs")
  plt.ylabel(string)
  plt.legend([string, 'val_'+string])
  plt.show()

plot_graphs(history, "acc")
plot_graphs(history, "loss")
```


acc
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0
5
10
15
20
25
30
Epochs
acc
val_acc
loss
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
30
Epochs
loss
val_loss

```
vocab_size = 1000   (was 10,000)
embedding_dim = 16
max_length = 16     (was 32)
trunc_type='post'
padding_type='post'
oov_tok = "<OOV>"
training_size = 20000
```

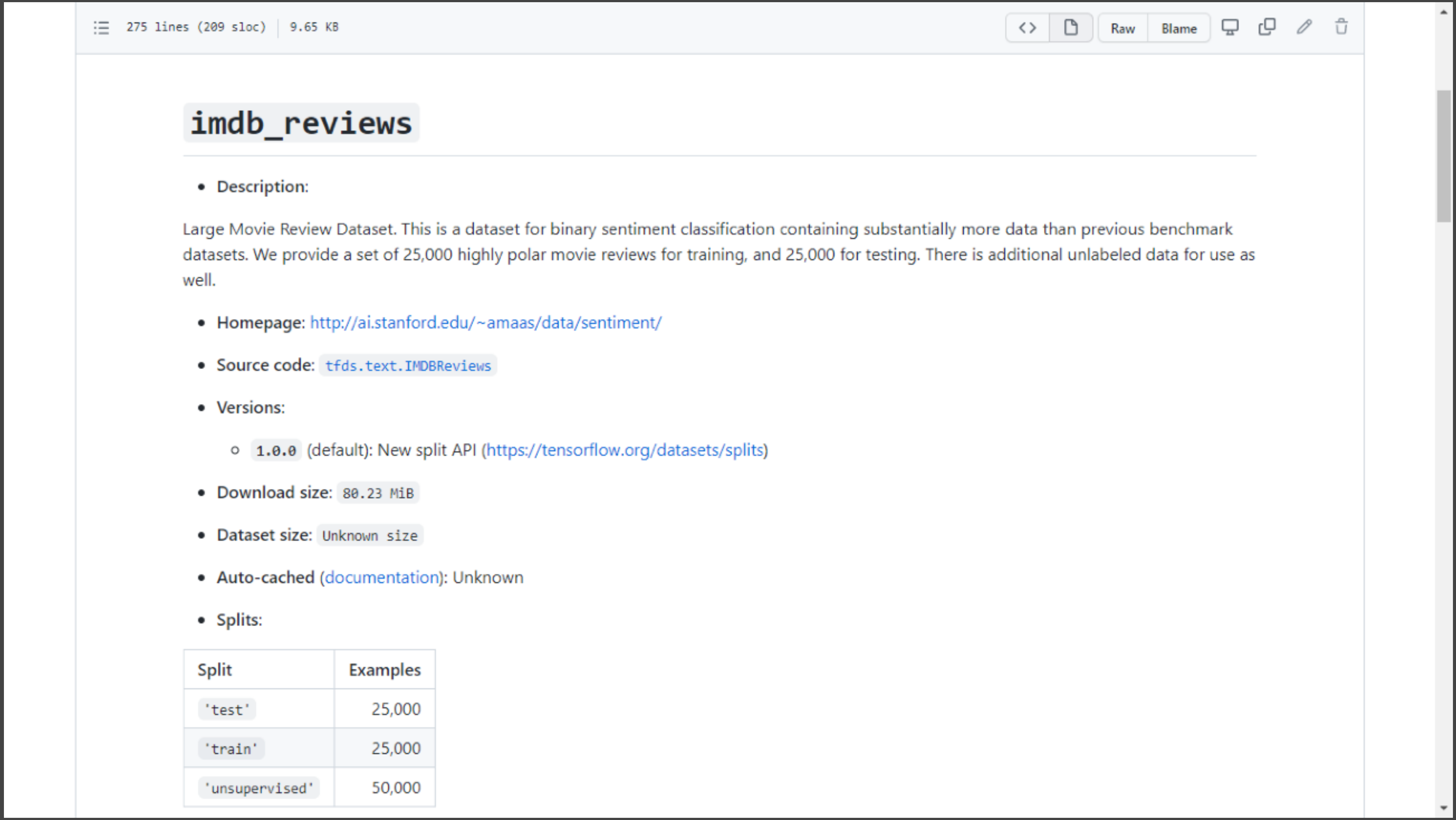

acc
val_acc
acc
0.86
0.84
0.82
0.80
0.78
0.76
0.74
0
5
10
15
20
25
30
Epochs
loss
val_loss
loss
0.50
0.45
0.40
0.35
0
5
10
15
20
25
30
Epochs

```
vocab_size = 1000   (was 10,000)
embedding_dim = 32  (was 16)
max_length = 16     (was 32)
trunc_type='post'
padding_type='post'
oov_tok = "<OOV>"
training_size = 20000
```


acc
val_acc
0.86
0.84
0.82
0.80
0.78
0.76
0.74
acc
0
5
10
15
20
25
30
Epochs
loss
val_loss
0.50
0.45
0.40
0.35
0.30
loss
0
5
10
15
20
25
30
Epochs

https://github.com/tensorflow/datasets/tree/master/docs/catalog

https://github.com/tensorflow/datasets/blob/master/docs/catalog/imdb_reviews.md



imdb_reviews

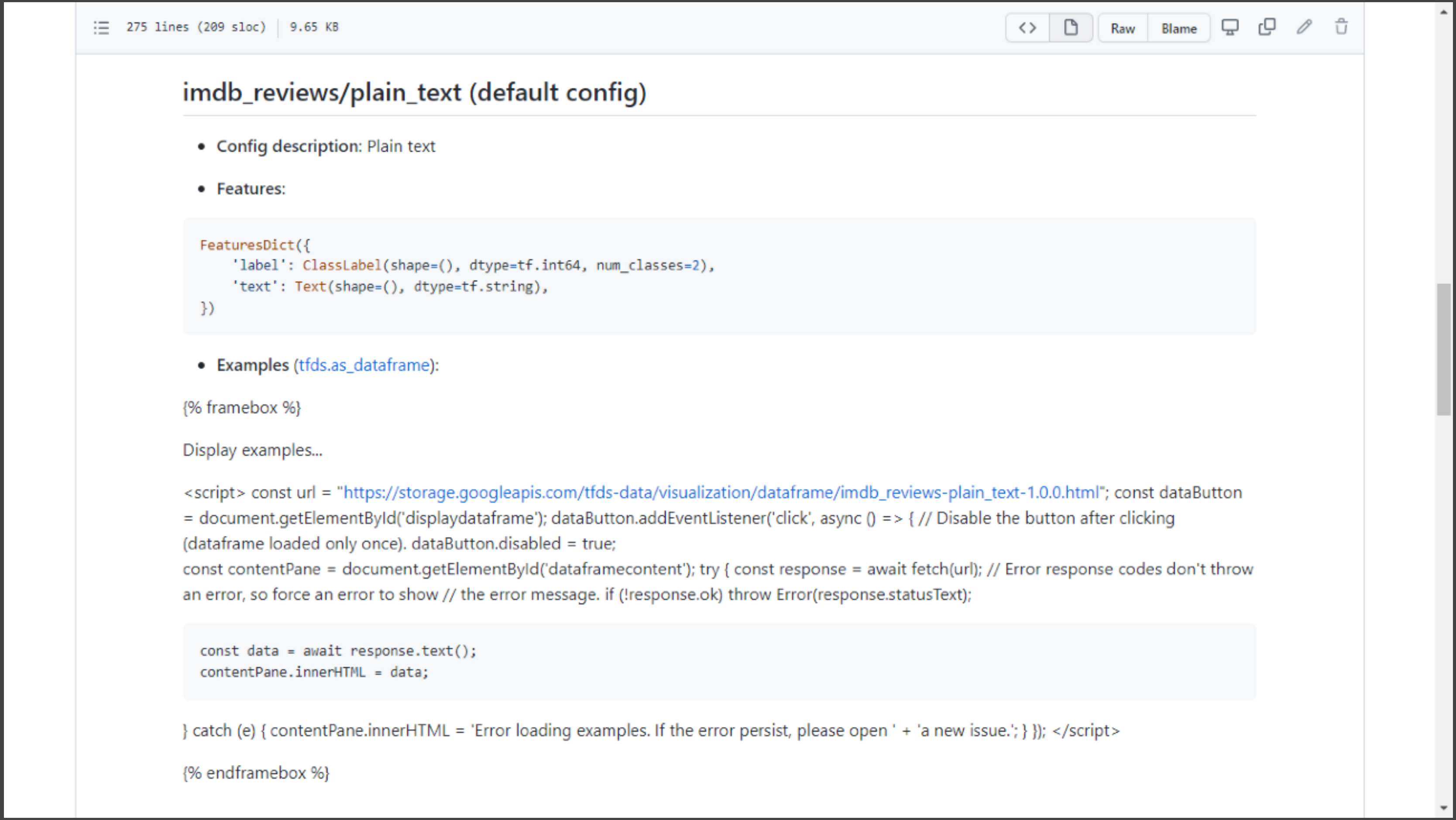
- **Description**:

Large Movie Review Dataset. This is a dataset for binary sentiment classification containing substantially more data than previous benchmark datasets. We provide a set of 25,000 highly polar movie reviews for training, and 25,000 for testing. There is additional unlabeled data for use as well.

- **Homepage**: http://ai.stanford.edu/~amaas/data/sentiment/
- **Source code**: `tfds.text.IMDBReviews`
- **Versions**:
  - `1.0.0` (default): New split API (https://tensorflow.org/datasets/splits)
- **Download size**: `80.23 MiB`
- **Dataset size**: `Unknown size`
- **Auto-cached** (documentation): Unknown
- **Splits**:

Split	Examples
`'test'`	25,000
`'train'`	25,000
`'unsupervised'`	50,000

https://github.com/tensorflow/datasets/blob/master/docs/catalog/imdb_reviews.md



imdb_reviews/plain_text (default config)

- **Config description**: Plain text

- **Features**:

```
FeaturesDict({
    'label': ClassLabel(shape=(), dtype=tf.int64, num_classes=2),
    'text': Text(shape=(), dtype=tf.string),
})
```

- **Examples** (tfds.as_dataframe):

{% framebox %}

Display examples...

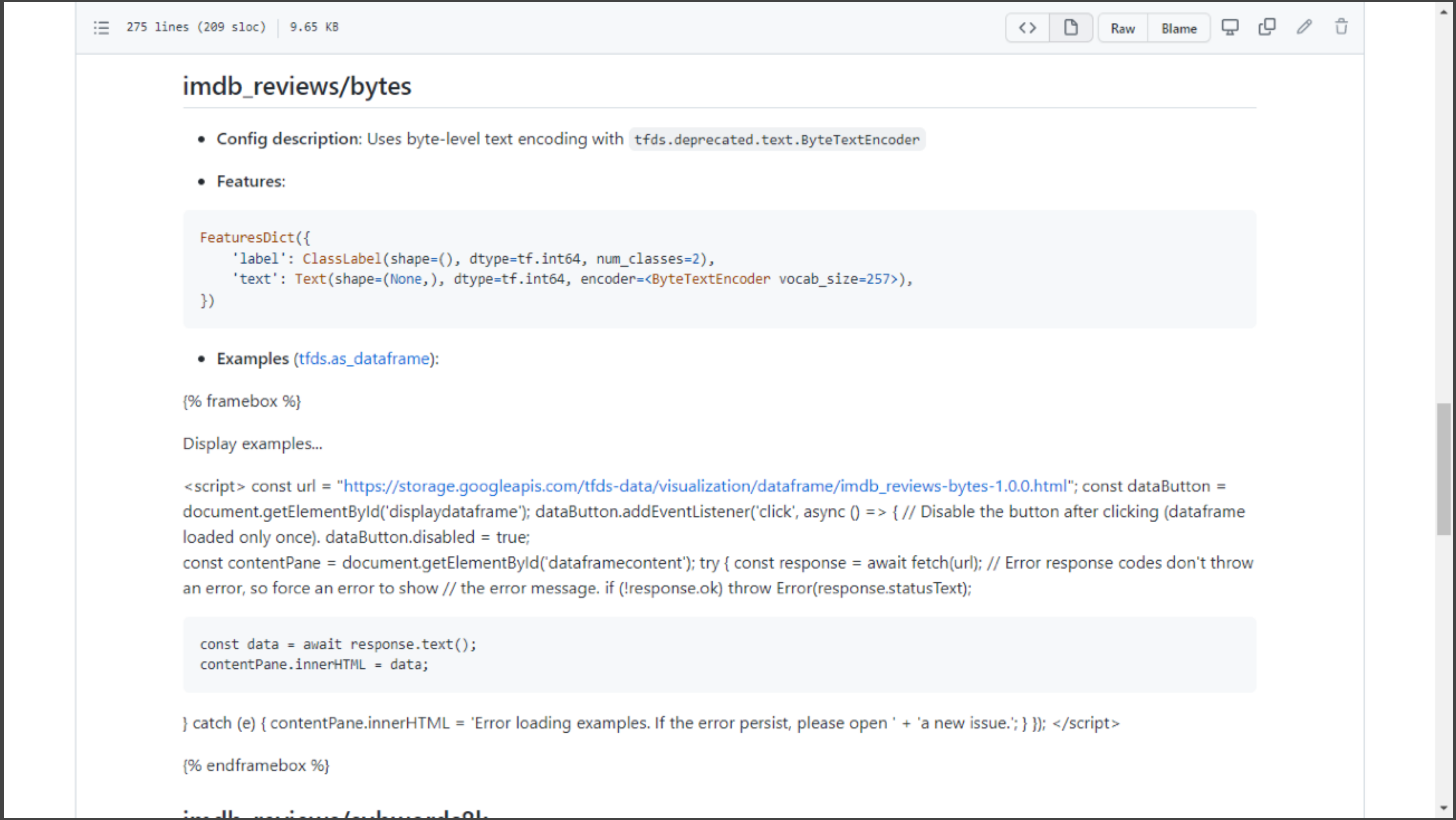
<script> const url = "https://storage.googleapis.com/tfds-data/visualization/dataframe/imdb_reviews-plain_text-1.0.0.html"; const dataButton = document.getElementById('displaydataframe'); dataButton.addEventListener('click', async () => { // Disable the button after clicking (dataframe loaded only once). dataButton.disabled = true;
const contentPane = document.getElementById('dataframecontent'); try { const response = await fetch(url); // Error response codes don't throw an error, so force an error to show // the error message. if (!response.ok) throw Error(response.statusText);

```
const data = await response.text();
contentPane.innerHTML = data;
```

} catch (e) { contentPane.innerHTML = 'Error loading examples. If the error persist, please open ' + 'a new issue.'; } }); </script>

{% endframebox %}

https://github.com/tensorflow/datasets/blob/master/docs/catalog/imdb_reviews.md



imdb_reviews/bytes

- **Config description**: Uses byte-level text encoding with `tfds.deprecated.text.ByteTextEncoder`
- **Features**:

```
FeaturesDict({
    'label': ClassLabel(shape=(), dtype=tf.int64, num_classes=2),
    'text': Text(shape=(None,), dtype=tf.int64, encoder=<ByteTextEncoder vocab_size=257>),
})
```

- **Examples** (tfds.as_dataframe):

{% framebox %}

Display examples...

<script> const url = "https://storage.googleapis.com/tfds-data/visualization/dataframe/imdb_reviews-bytes-1.0.0.html"; const dataButton = document.getElementById('displaydataframe'); dataButton.addEventListener('click', async () => { // Disable the button after clicking (dataframe loaded only once). dataButton.disabled = true;
const contentPane = document.getElementById('dataframecontent'); try { const response = await fetch(url); // Error response codes don't throw an error, so force an error to show // the error message. if (!response.ok) throw Error(response.statusText);

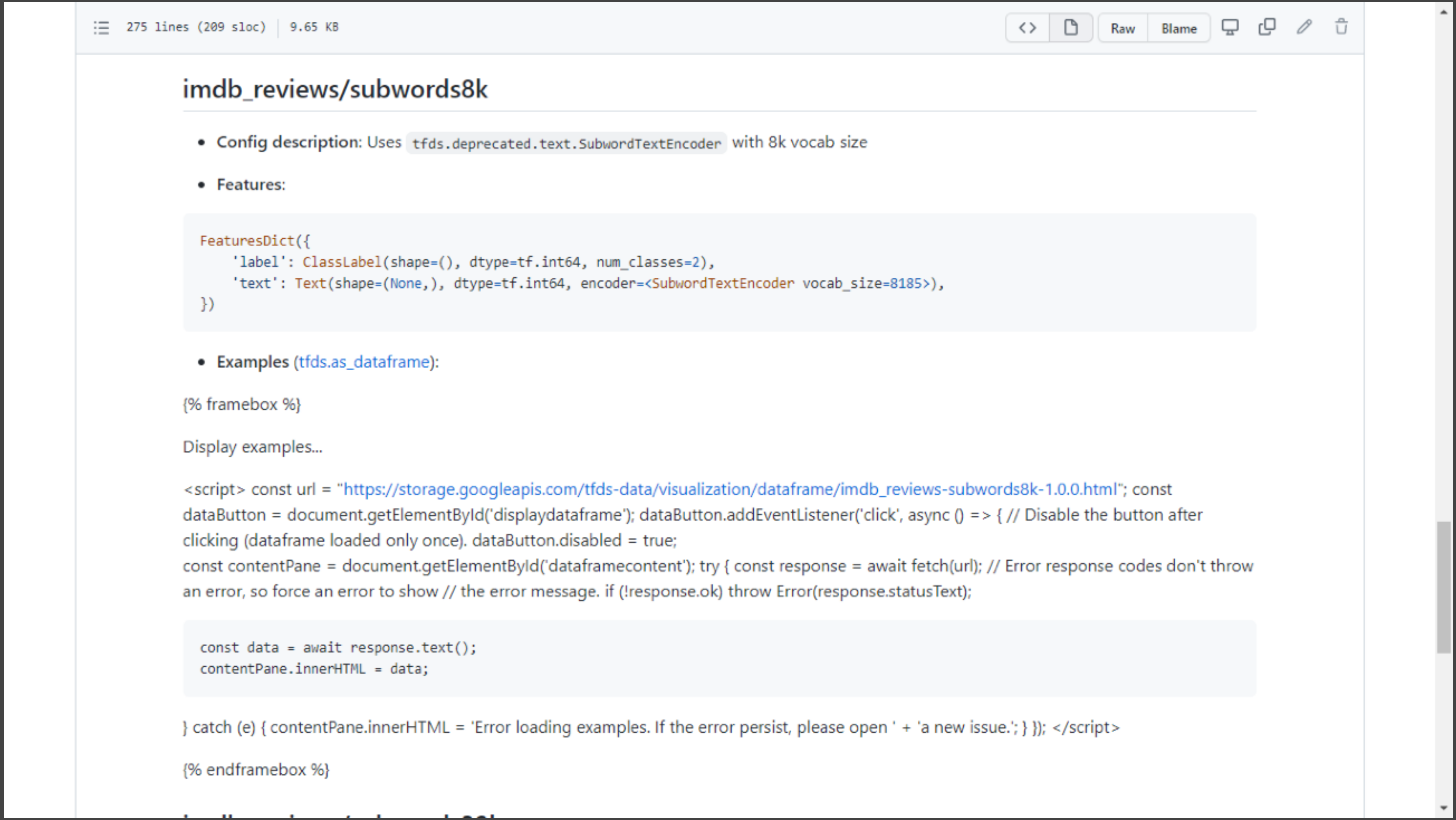
```
const data = await response.text();
contentPane.innerHTML = data;
```

} catch (e) { contentPane.innerHTML = 'Error loading examples. If the error persist, please open ' + 'a new issue.'; } }); </script>

{% endframebox %}

imdb_reviews/subwords8k

https://github.com/tensorflow/datasets/blob/master/docs/catalog/imdb_reviews.md



imdb_reviews/subwords8k

- **Config description**: Uses `tfds.deprecated.text.SubwordTextEncoder` with 8k vocab size
- **Features**:

```
FeaturesDict({
    'label': ClassLabel(shape=(), dtype=tf.int64, num_classes=2),
    'text': Text(shape=(None,), dtype=tf.int64, encoder=<SubwordTextEncoder vocab_size=8185>),
})
```

- **Examples** (tfds.as_dataframe):

{% framebox %}

Display examples...

<script> const url = "https://storage.googleapis.com/tfds-data/visualization/dataframe/imdb_reviews-subwords8k-1.0.0.html"; const dataButton = document.getElementById('displaydataframe'); dataButton.addEventListener('click', async () => { // Disable the button after clicking (dataframe loaded only once). dataButton.disabled = true;
const contentPane = document.getElementById('dataframecontent'); try { const response = await fetch(url); // Error response codes don't throw an error, so force an error to show // the error message. if (!response.ok) throw Error(response.statusText);

```
const data = await response.text();
contentPane.innerHTML = data;
```

} catch (e) { contentPane.innerHTML = 'Error loading examples. If the error persist, please open ' + 'a new issue.'; } }); </script>

{% endframebox %}

```
import tensorflow_datasets as tfds
imdb, info = tfds.load("imdb_reviews/subwords8k", with_info=True, as_supervised=True)
```

```
train_data, test_data = imdb['train'], imdb['test']
```

```
tokenizer = info.features['text'].encoder
```

tensorflow.org/datasets/api_docs/python/tfds/features/text/SubwordTextEncoder

```
print(tokenizer_subwords.subwords)


['the_', ', ', '. ', 'a_', 'and_', 'of_', 'to_', 's_', 'is_', 'br', 'in_', 'I_', 'that_', 'this_', 'it_', … ]
```

```
sample_string = 'TensorFlow, from basics to mastery'

tokenized_string = tokenizer_subwords.encode(sample_string)
print ('Tokenized string is {}'.format(tokenized_string))

original_string = tokenizer_subwords.decode(tokenized_string)
print ('The original string: {}'.format(original_string))
```

```
Tokenized string is [6307, 2327, 4043, 2120, 2, 48, 4249, 4429, 7, 2652, 8050]

The original string: TensorFlow, from basics to mastery
```

```python
sample_string = 'TensorFlow, from basics to mastery'

tokenized_string = tokenizer_subwords.encode(sample_string)
print ('Tokenized string is {}'.format(tokenized_string))

original_string = tokenizer_subwords.decode(tokenized_string)
print ('The original string: {}'.format(original_string))
```

```
Tokenized string is [6307, 2327, 4043, 2120, 2, 48, 4249, 4429, 7, 2652, 8050]

The original string: TensorFlow, from basics to mastery
```

```
sample_string = 'TensorFlow, from basics to mastery'

tokenized_string = tokenizer_subwords.encode(sample_string)
print ('Tokenized string is {}'.format(tokenized_string))

original_string = tokenizer_subwords.decode(tokenized_string)
print ('The original string: {}'.format(original_string))
```

```
Tokenized string is [6307, 2327, 4043, 2120, 2, 48, 4249, 4429, 7, 2652, 8050]

The original string: TensorFlow, from basics to mastery
```

```
sample_string = 'TensorFlow, from basics to mastery'

tokenized_string = tokenizer_subwords.encode(sample_string)
print ('Tokenized string is {}'.format(tokenized_string))

original_string = tokenizer_subwords.decode(tokenized_string)
print ('The original string: {}'.format(original_string))
```

```
Tokenized string is [6307, 2327, 4043, 2120, 2, 48, 4249, 4429, 7, 2652, 8050]

The original string: TensorFlow, from basics to mastery
```

```python
for ts in tokenized_string:
  print ('{} ----> {}'.format(ts, tokenizer_subwords.decode([ts])))
```

```
6307 ----> Ten
2327 ----> sor
4043 ----> Fl
2120 ----> ow
2 ----> ,
48 ----> from
4249 ----> basi
4429 ----> cs
7 ----> to
2652 ----> master
8050 ----> y
```

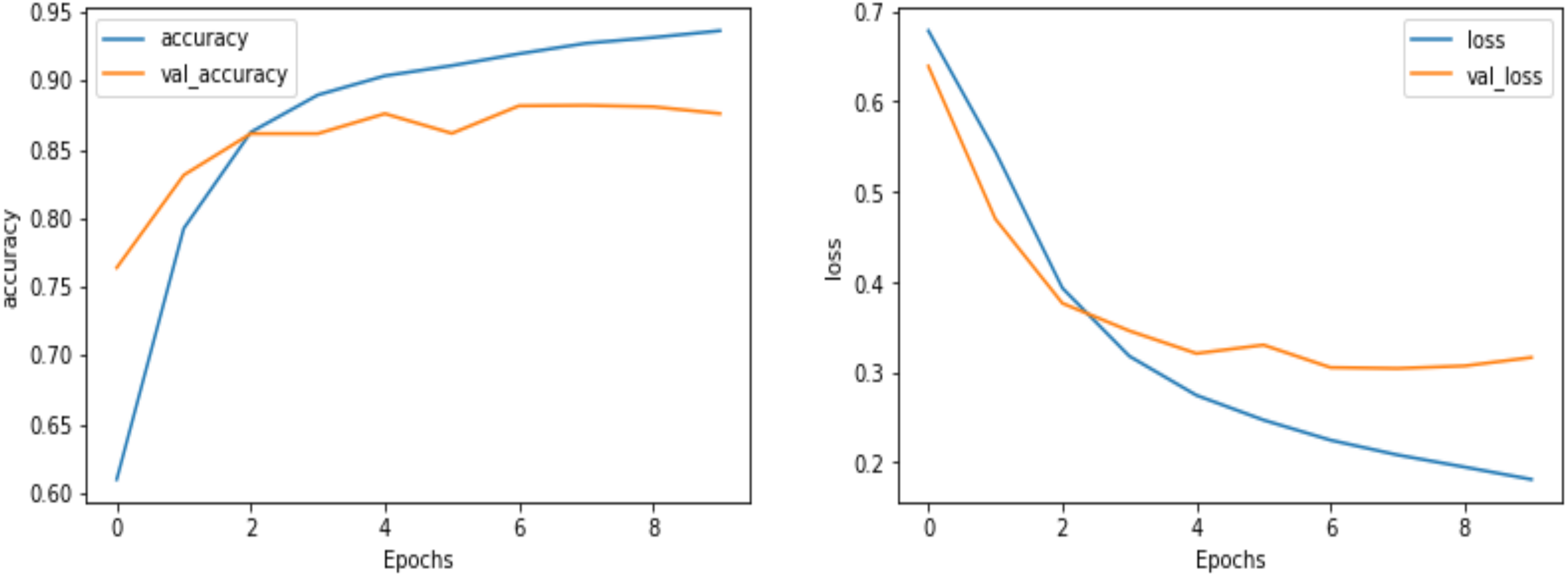
```
embedding_dim = 64
model = tf.keras.Sequential([
  tf.keras.layers.Embedding(tokenizer_subwords.vocab_size, embedding_dim),
  tf.keras.layers.GlobalAveragePooling1D(),
  tf.keras.layers.Dense(6, activation='relu'),
  tf.keras.layers.Dense(1, activation='sigmoid')
])

model.summary()
```

```
embedding_dim = 64
model = tf.keras.Sequential([
  tf.keras.layers.Embedding(tokenizer_subwords.vocab_size, embedding_dim),
  tf.keras.layers.GlobalAveragePooling1D(),
  tf.keras.layers.Dense(6, activation='relu'),
  tf.keras.layers.Dense(1, activation='sigmoid')
])

model.summary()
```

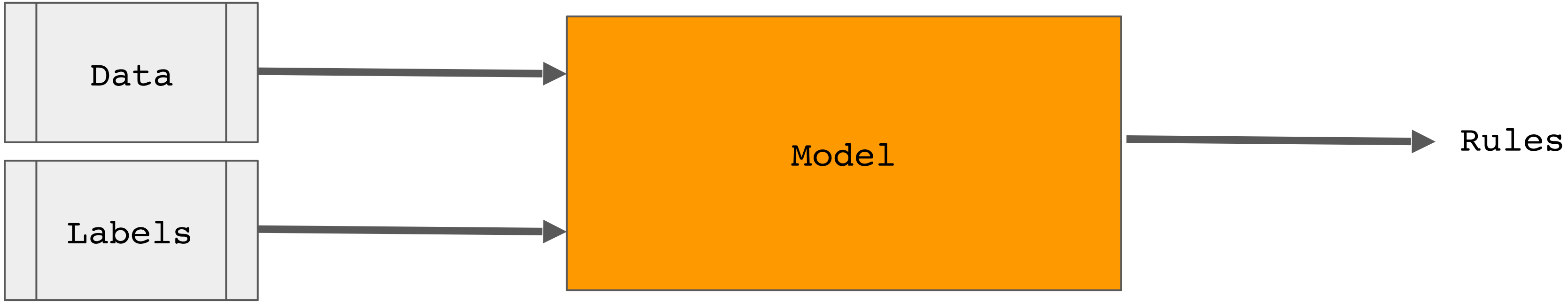
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, None, 64)          523840
_________________________________________________________________
global_average_pooling1d_1 ( (None, 64)                0
_________________________________________________________________
dense_4 (Dense)              (None, 6)                 390
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 7
=================================================================
Total params: 524,237
Trainable params: 524,237
Non-trainable params: 0
```

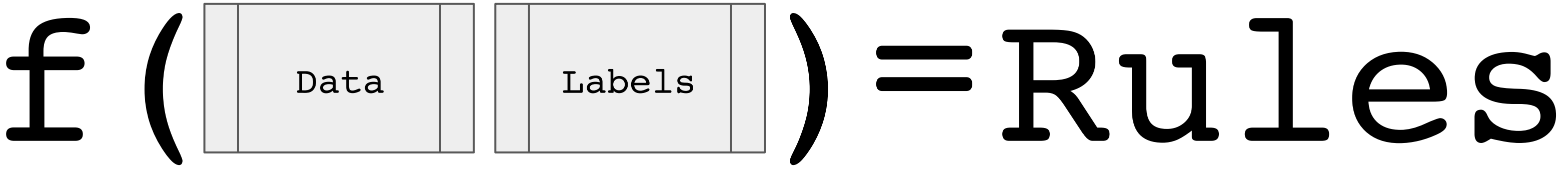
```
num_epochs = 10

model.compile(loss='binary_crossentropy',
       optimizer='adam',
       metrics=['accuracy'])

history = model.fit(train_dataset,
          epochs=num_epochs,
          validation_data=test_data)
```

```python
import matplotlib.pyplot as plt



def plot_graphs(history, string):
  plt.plot(history.history[string])
  plt.plot(history.history['val_'+string])
  plt.xlabel("Epochs")
  plt.ylabel(string)
  plt.legend([string, 'val_'+string])
  plt.show()

plot_graphs(history, "accuracy")
plot_graphs(history, "loss")
```


accuracy
val_accuracy
accuracy
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0
2
4
6
8
Epochs
loss
val_loss
loss
0.7
0.6
0.5
0.4
0.3
0.2
0
2
4
6
8
Epochs

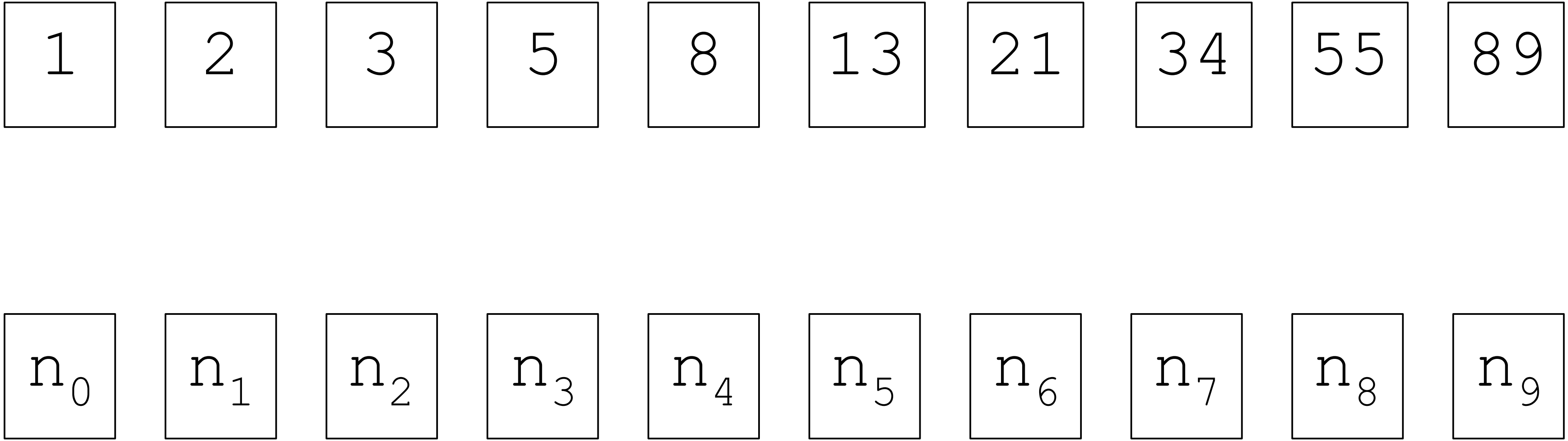
Copyright Notice


Data
Labels
Model
Rules



f(Data Labels)=Rules

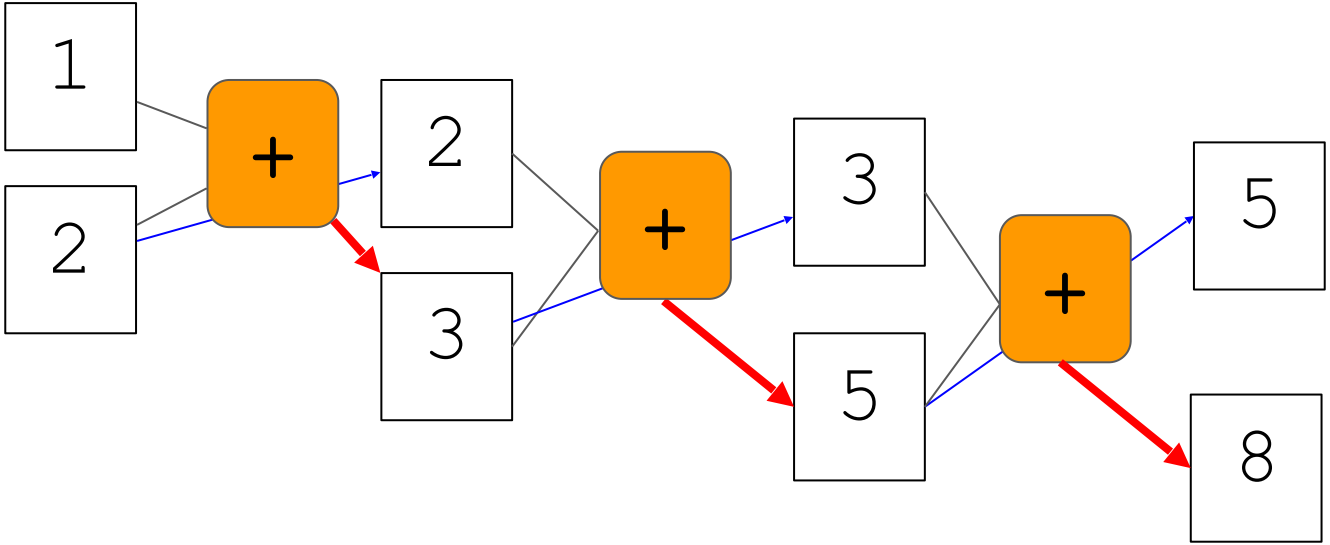
1	2	3	5	8	13	21	34	55	89

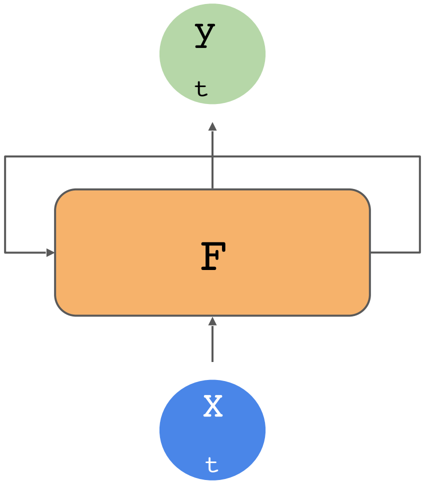


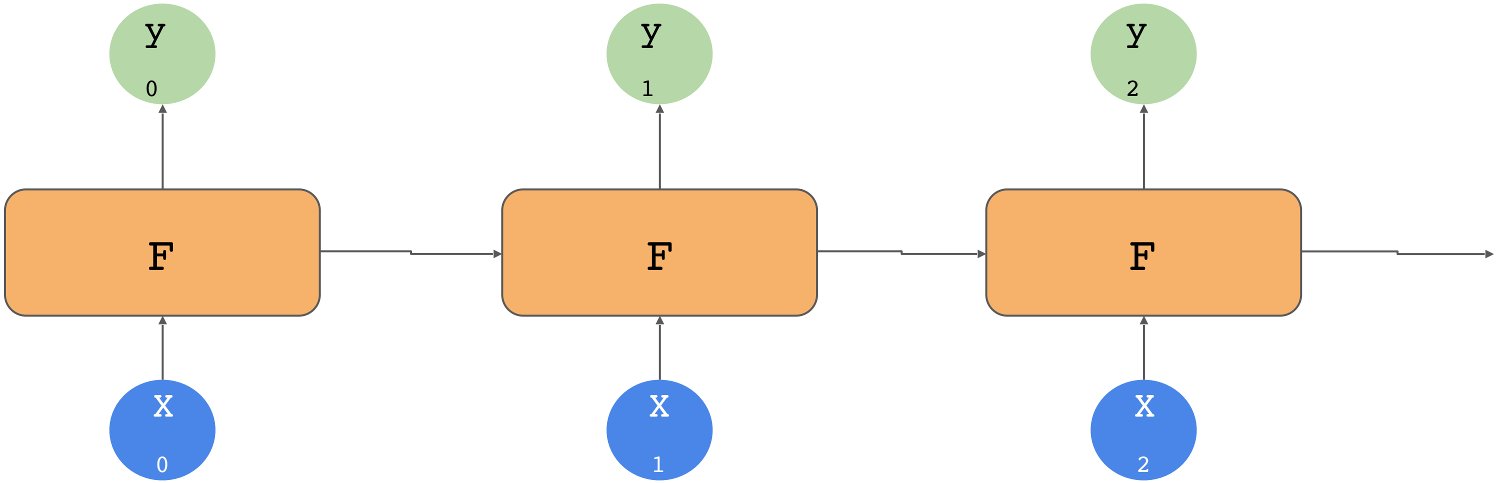
1	2	3	5	8	13	21	34	55	89
n_0	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9

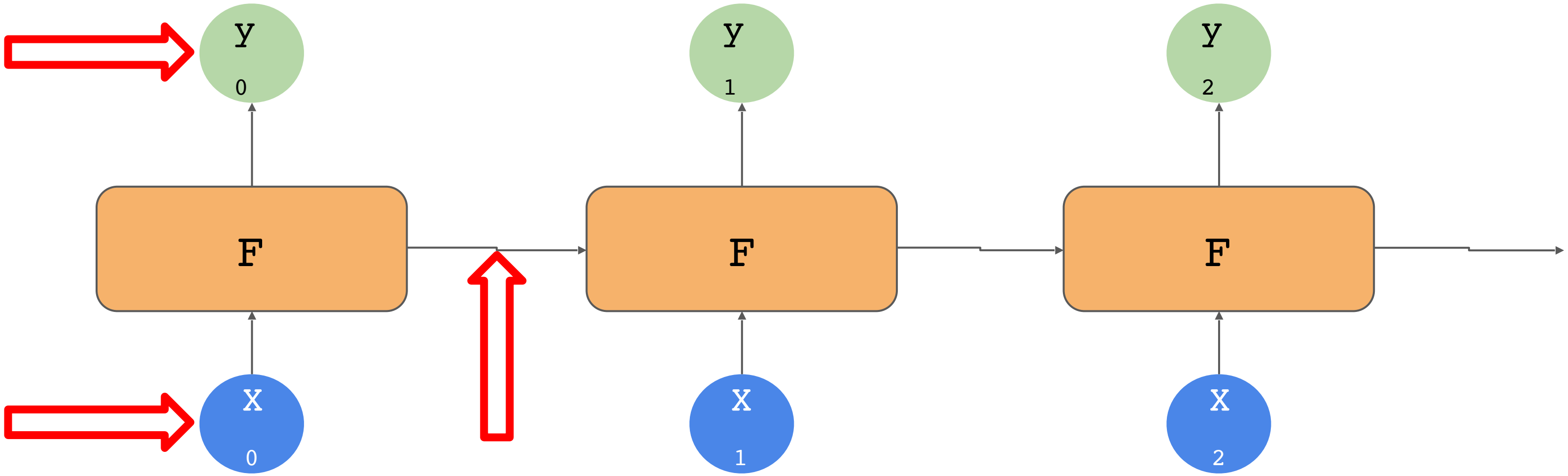
1	2	3	5	8	13	21	34	55	89
n_0	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9

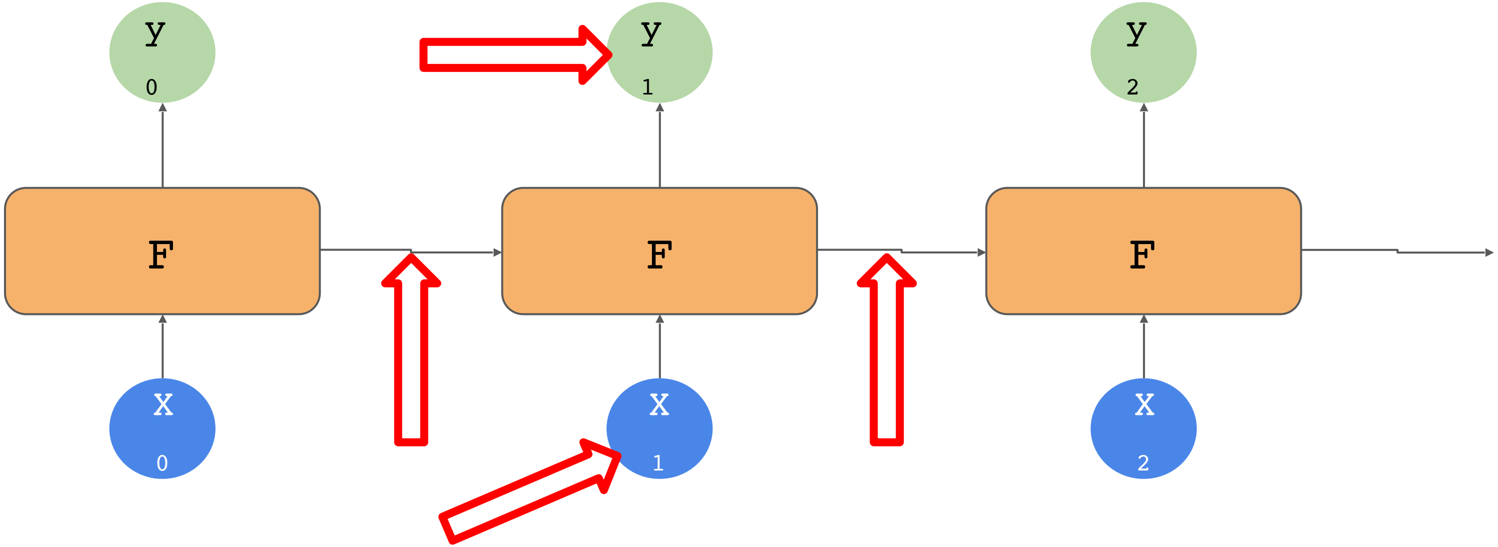
$$n_x = n_{x-1} + n_{x-2}$$

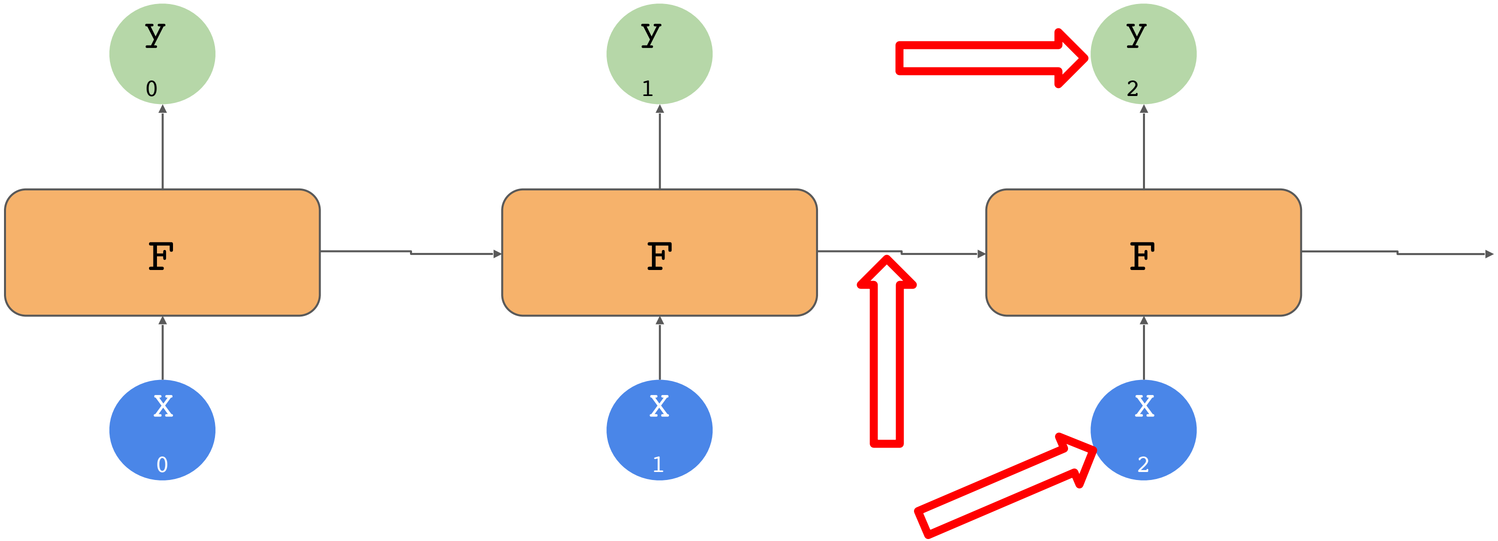

1
2
+
2
3
+
3
5
+
5
8

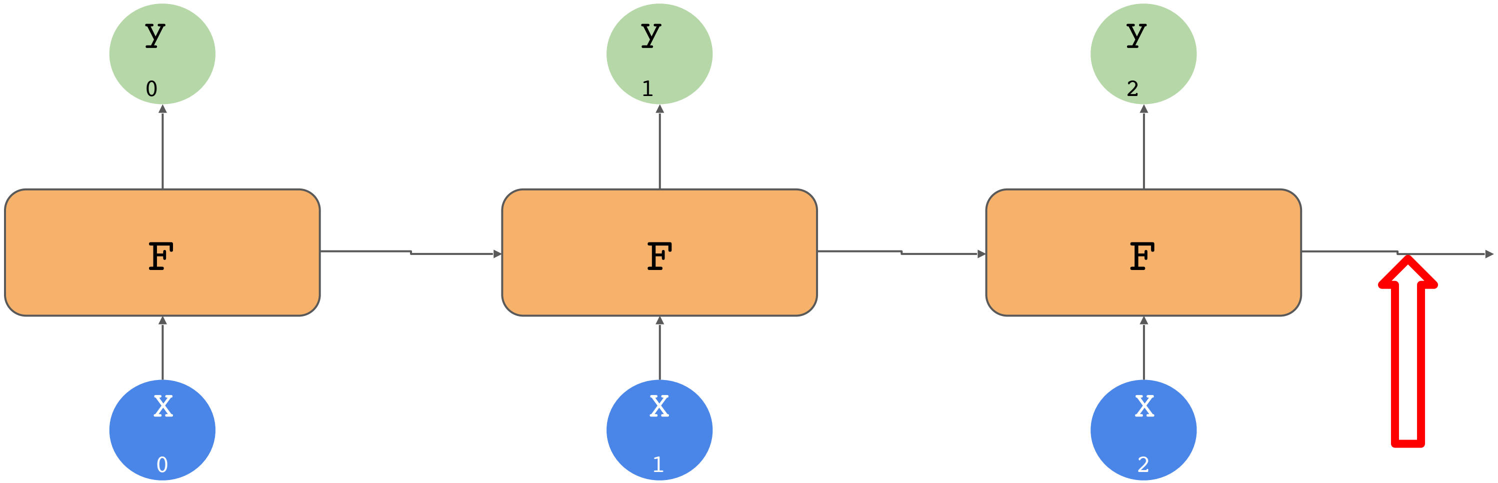

y
t
F
x
t


y
0
y
1
y
2
F
F
F
x
0
x
1
x
2

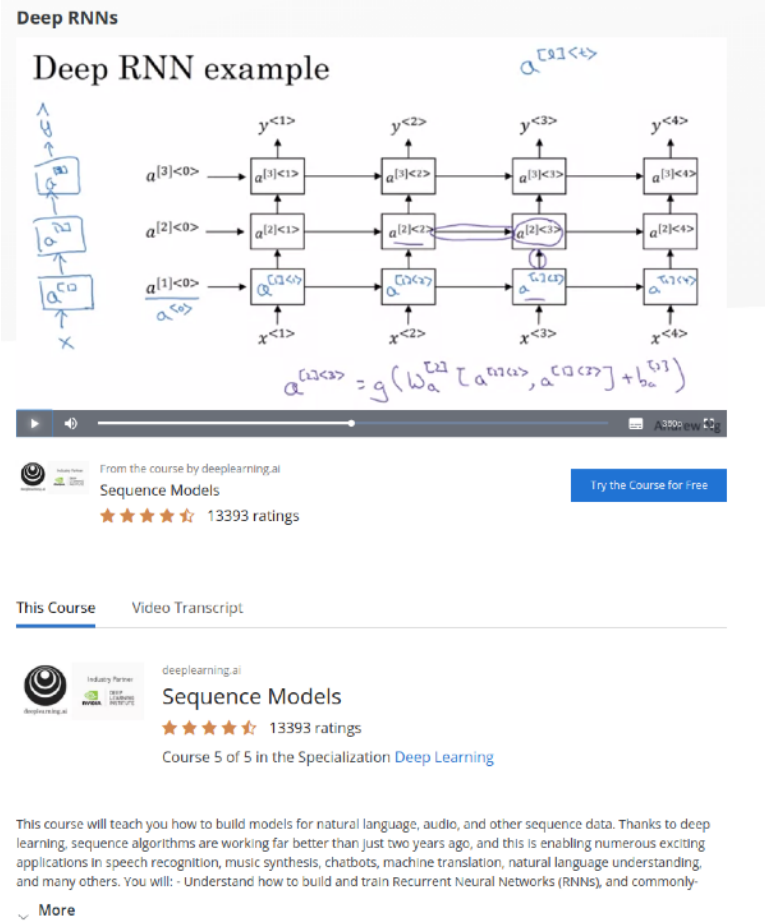

y
0
y
1
y
2
F
F
F
X
0
X
1
X
2


y
0
y
1
y
2
F
F
F
X
0
X
1
X
2


y
0
y
1
y
2
F
F
F
X
0
X
1
X
2


y
0
y
1
y
2
F
F
F
X
0
X
1
X
2

https://www.coursera.org/lecture/nlp-sequence-models/deep-rnns-ehs0S



Deep RNNs

Deep RNN example

$$a^{[2]<3>} = g(W_a^{[2]}[a^{[2]<2>}, a^{[1]<3>}] + b_a^{[2]})$$

From the course by deeplearning.ai

Sequence Models

★★★★☆ 13393 ratings

This Course

Video Transcript

deeplearning.ai

Sequence Models

★★★★☆ 13393 ratings

Course 5 of 5 in the Specialization Deep Learning

This course will teach you how to build models for natural language, audio, and other sequence data. Thanks to deep learning, sequence algorithms are working far better than just two years ago, and this is enabling numerous exciting applications in speech recognition, music synthesis, chatbots, machine translation, natural language understanding, and many others. You will: - Understand how to build and train Recurrent Neural Networks (RNNs), and commonly-

More

Today has a beautiful blue <...>

Today has a beautiful blue <...>

Today has a beautiful blue sky

Today has a beautiful blue <...>

Today has a beautiful blue sky

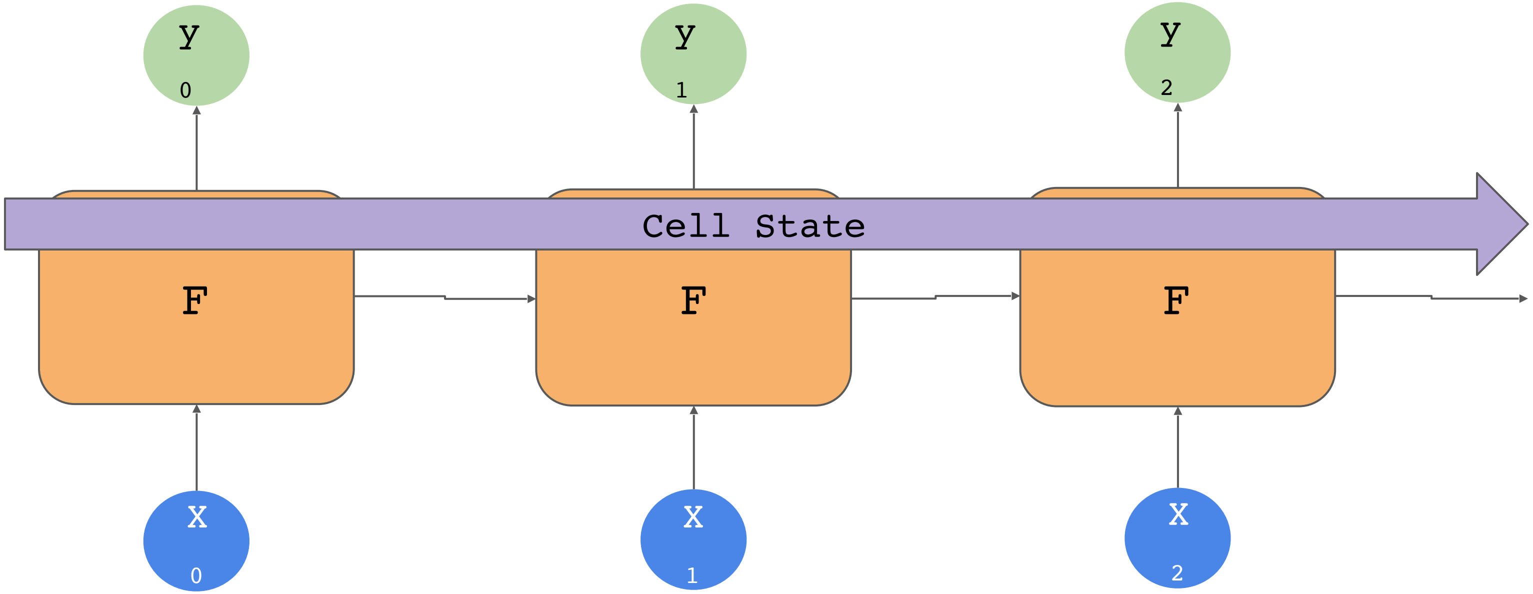
I lived in Ireland, so at school they made me learn how to speak <...>

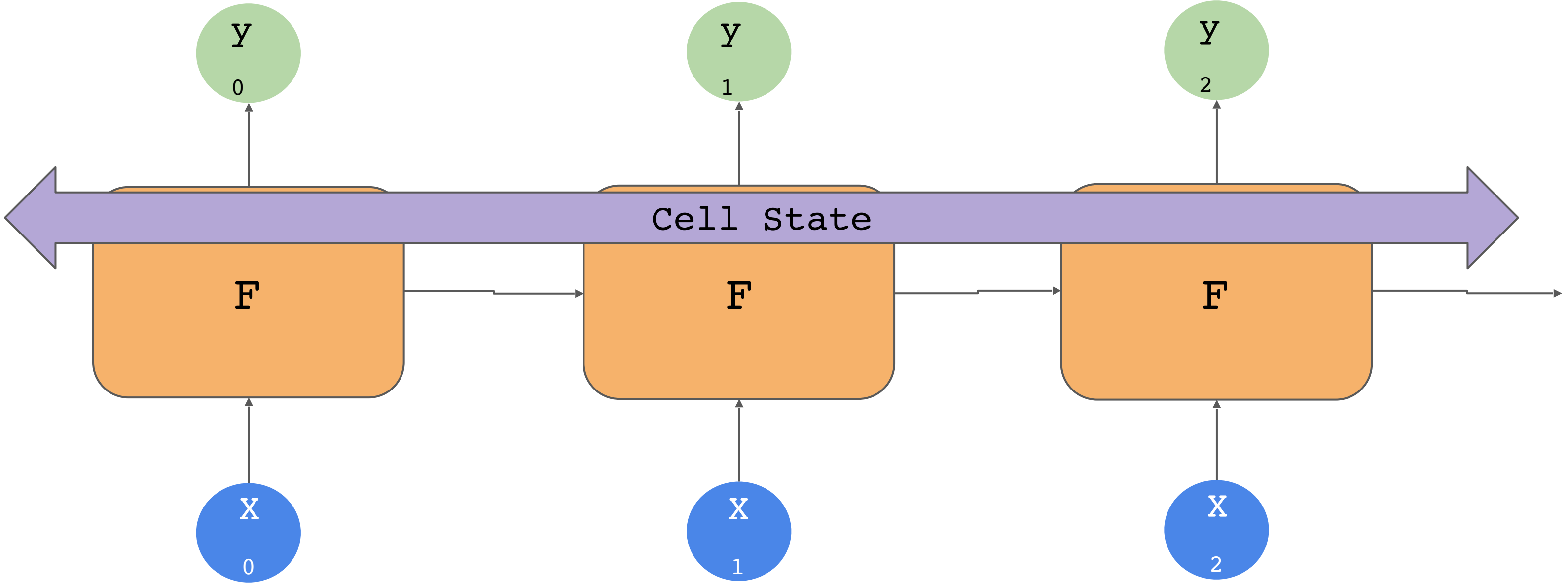
I lived in Ireland, so at school they made me learn how to speak <...>

I lived in Ireland, so at school they made me learn how to speak Gaelic

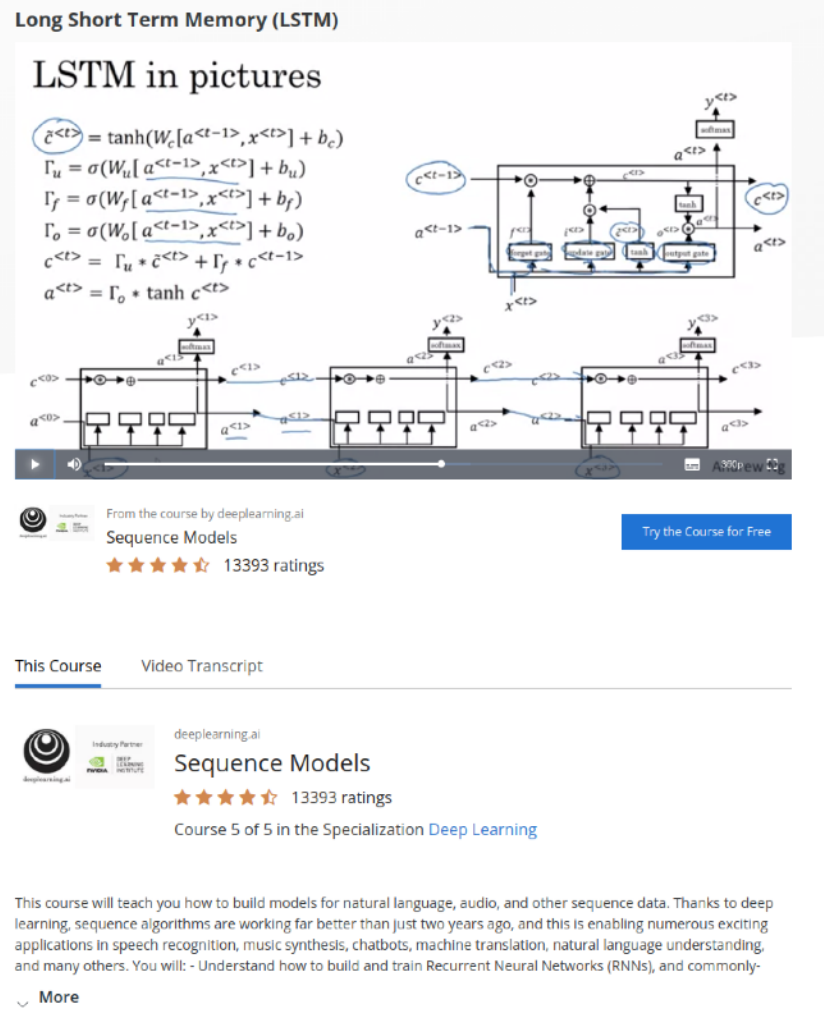
I lived in Ireland, so at school they made me learn how to speak <...>

I lived in Ireland so at school they made me learn how to speak Gaelic


y 0
y 1
y 2
Cell State
F
F
F
X 0
X 1
X 2


y 0
y 1
y 2
Cell State
F
F
F
X 0
X 1
X 2

https://www.coursera.org/lecture/nlp-sequence-models/long-short-term-memory-lstm-KXoay



Long Short Term Memory (LSTM)

LSTM in pictures

$$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$

$$\Gamma_u = \sigma(W_u[\, a^{<t-1>}, x^{<t>}] + b_u)$$

$$\Gamma_f = \sigma(W_f[\, a^{<t-1>}, x^{<t>}] + b_f)$$

$$\Gamma_o = \sigma(W_o[\, a^{<t-1>}, x^{<t>}] + b_o)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$$

$$a^{<t>} = \Gamma_o * \tanh c^{<t>}$$

From the course by deeplearning.ai

Sequence Models

★★★★☆ 13393 ratings

deeplearning.ai

Sequence Models

★★★★☆ 13393 ratings

Course 5 of 5 in the Specialization Deep Learning

This course will teach you how to build models for natural language, audio, and other sequence data. Thanks to deep learning, sequence algorithms are working far better than just two years ago, and this is enabling numerous exciting applications in speech recognition, music synthesis, chatbots, machine translation, natural language understanding, and many others. You will: - Understand how to build and train Recurrent Neural Networks (RNNs), and commonly-

```python
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(tokenizer.vocab_size, 64),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(tokenizer.vocab_size, 64),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(tokenizer.vocab_size, 64),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
_________________________________________________________________
Layer (type)               Output Shape             Param #
=================================================================
embedding_2 (Embedding)    (None, None, 64)          523840
_________________________________________________________________
bidirectional_1 (Bidirection (None, 128)             66048
_________________________________________________________________
dense_4 (Dense)            (None, 64)                8256
_________________________________________________________________
dense_5 (Dense)            (None, 1)                 65
=================================================================
Total params: 598,209
Trainable params: 598,209
Non-trainable params: 0
```

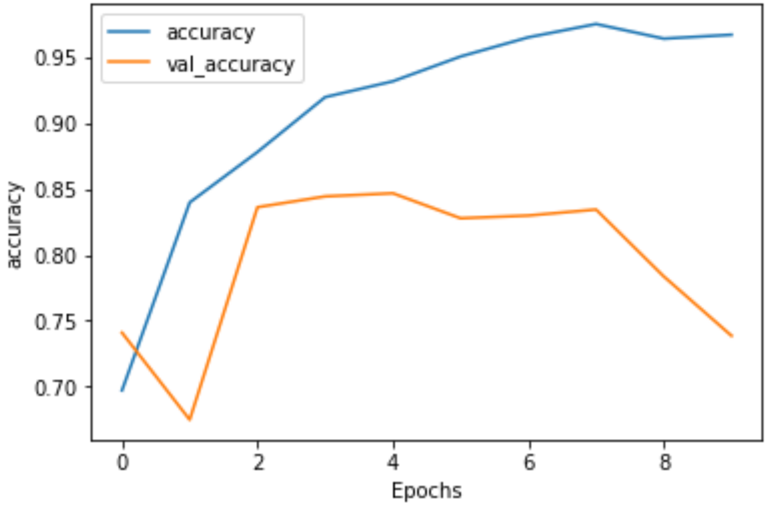
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, None, 64)          523840
_________________________________________________________________
bidirectional_1 (Bidirection (None, 128)               66048
_________________________________________________________________
dense_4 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 65
=================================================================
Total params: 598,209
Trainable params: 598,209
Non-trainable params: 0
```

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(tokenizer.vocab_size, 64),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64, return_sequences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

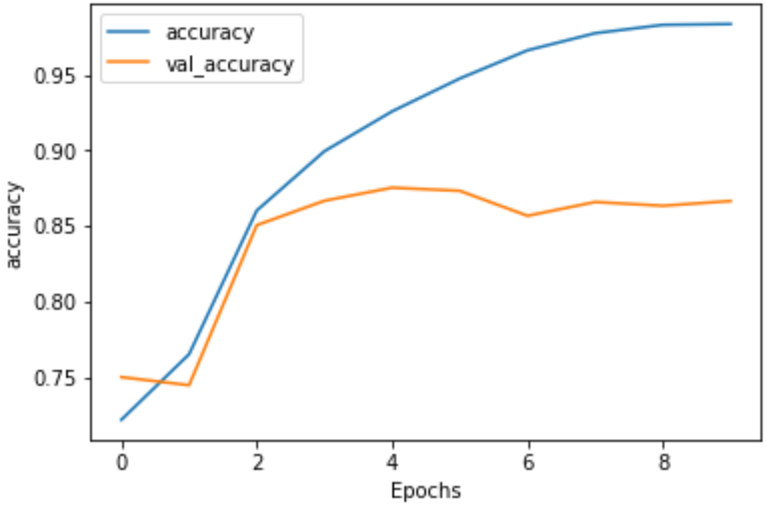
```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(tokenizer.vocab_size, 64),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64, return_sequences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, None, 64)          523840
_________________________________________________________________
bidirectional_2 (Bidirection (None, None, 128)         66048
_________________________________________________________________
bidirectional_3 (Bidirection (None, 64)                41216
_________________________________________________________________
dense_6 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 65
=================================================================
Total params: 635,329
Trainable params: 635,329
Non-trainable params: 0
```

10 Epochs : Accuracy Measurement

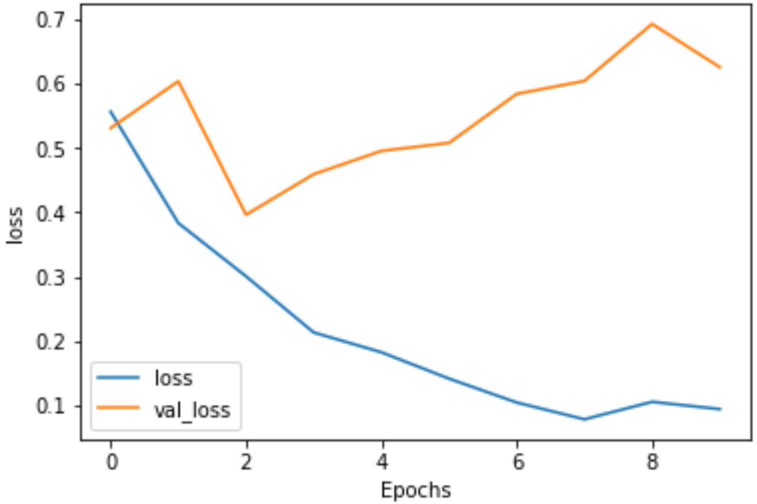


1 Layer
LSTM

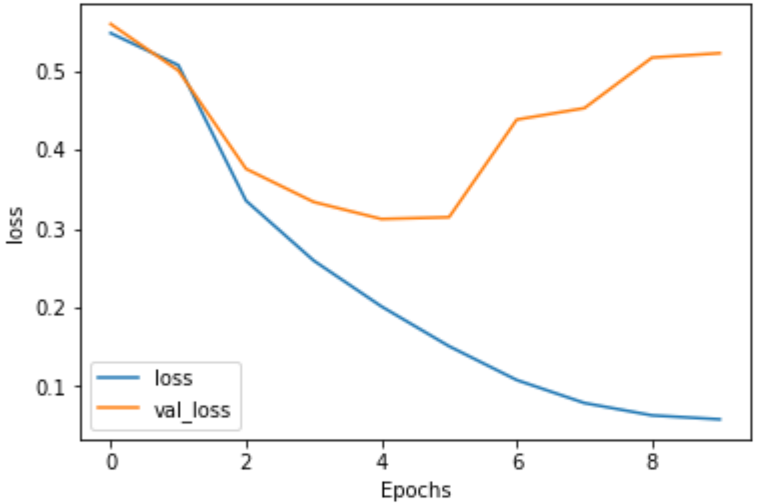


2 Layer
LSTM

10 Epochs : Loss Measurement

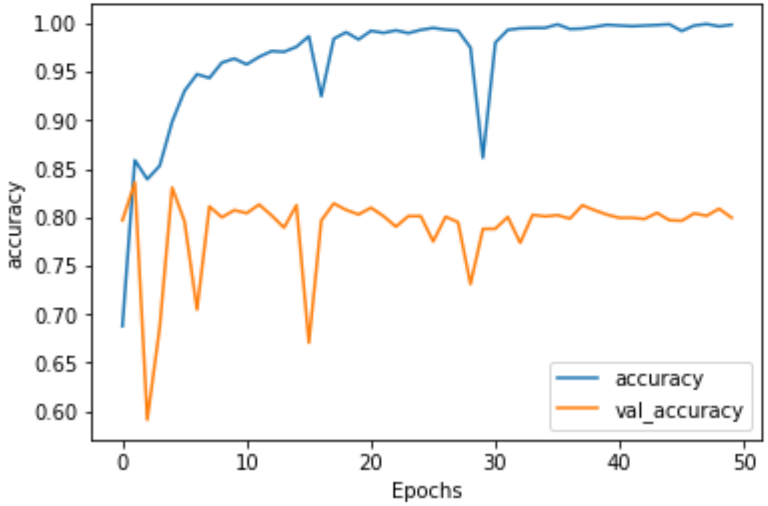


1 Layer
LSTM

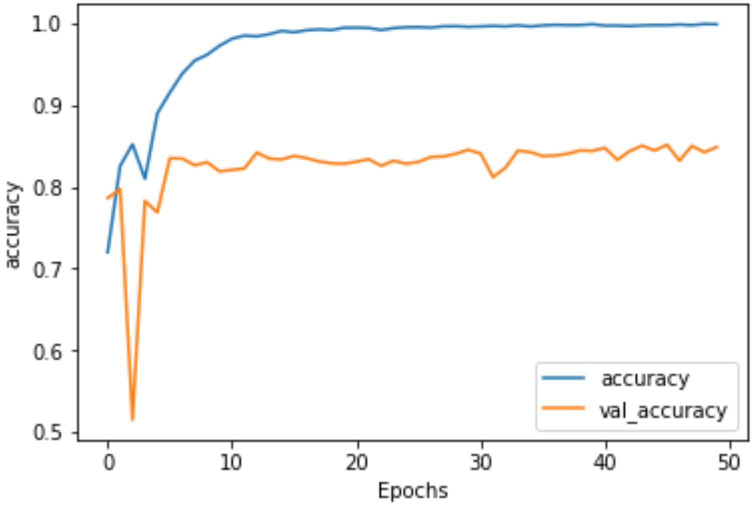


2 Layer
LSTM

50 Epochs : Accuracy Measurement

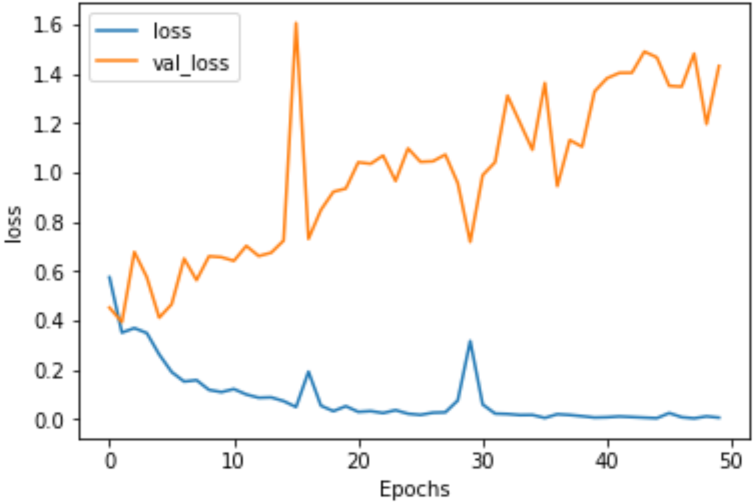


1 Layer
LSTM

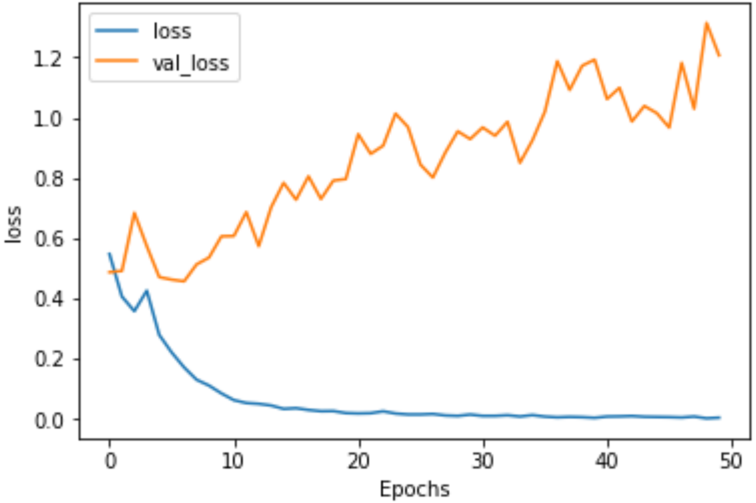


2 Layer
LSTM

50 Epochs : Loss Measurement



1 Layer
LSTM

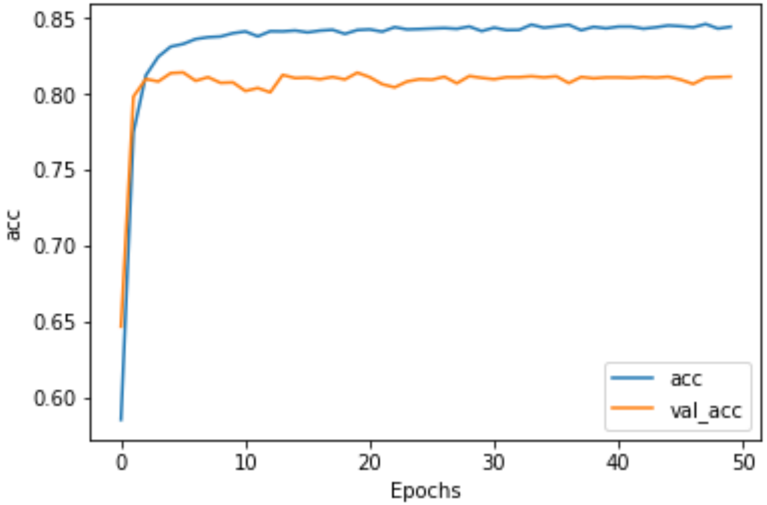


2 Layer
LSTM

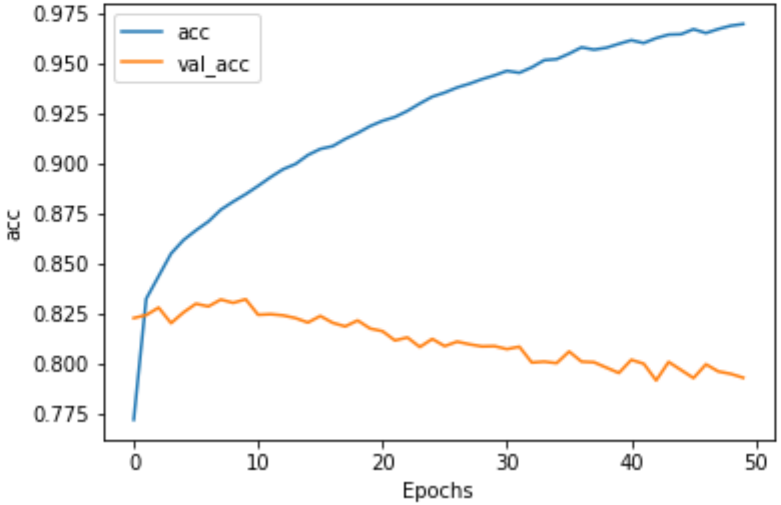
```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim,
                     input_length=max_length),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(24, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim,
                              input_length=max_length),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(24, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

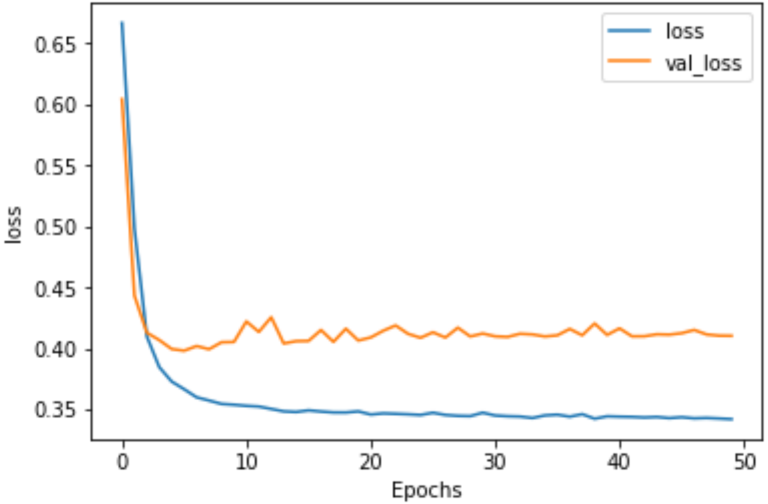
```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim,
                              input_length=max_length),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Dense(24, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```



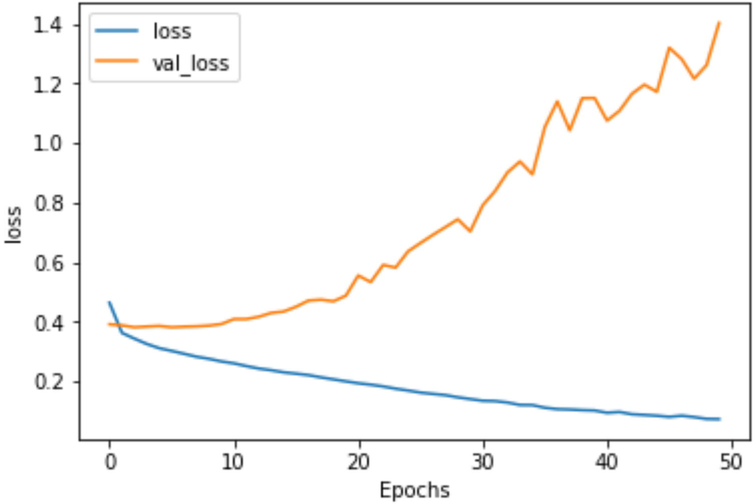
Without LSTM



With LSTM



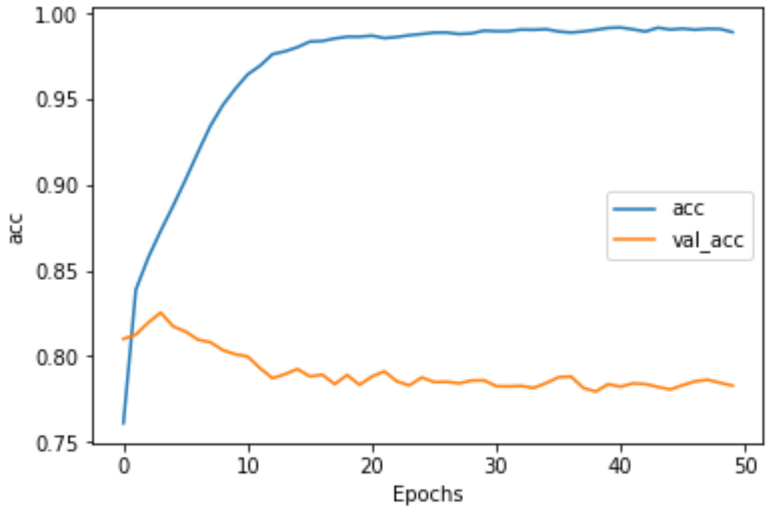
Without
LSTM

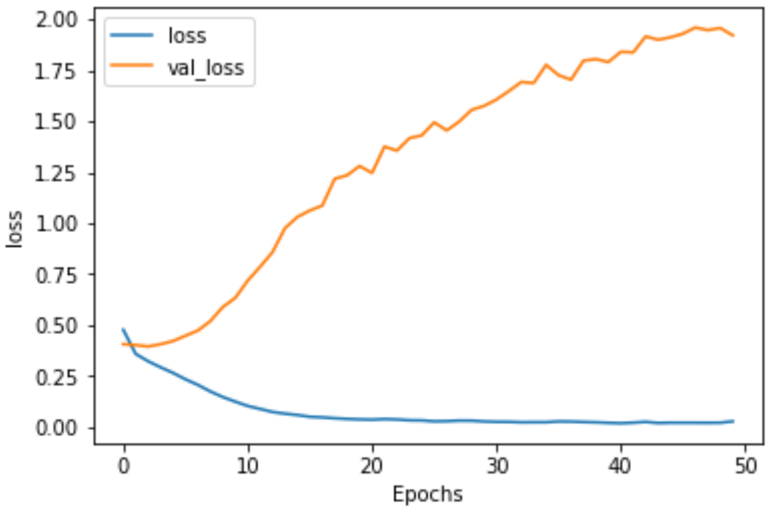


With LSTM

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim,
                    input_length=max_length),
    tf.keras.layers.Conv1D(128, 5, activation='relu'),
    tf.keras.layers.GlobalMaxPooling1D(),
    tf.keras.layers.Dense(24, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim,
                    input_length=max_length),
    tf.keras.layers.Conv1D(128, 5, activation='relu'),
    tf.keras.layers.GlobalMaxPooling1D(),
    tf.keras.layers.Dense(24, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```


1.00
0.95
0.90
0.85
0.80
0.75
acc
0
10
20
30
40
50
Epochs
acc
val_acc


2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
loss
0
10
20
30
40
50
Epochs
loss
val_loss

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim,
                              input_length=max_length),
    tf.keras.layers.Conv1D(128, 5, activation='relu'),
    tf.keras.layers.GlobalMaxPooling1D(),
    tf.keras.layers.Dense(24, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
max_length = 120

tf.keras.layers.Conv1D(128, 5, activation='relu'),
```

```
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 120, 16)           16000
_________________________________________________________________
conv1d (Conv1D)              (None, 116, 128)          10368
_________________________________________________________________
global_max_pooling1d (Global (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 24)                3096
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 25
=================================================================
Total params: 29,489
Trainable params: 29,489
Non-trainable params: 0
```

```
max_length = 120

tf.keras.layers.Conv1D(128, 5, activation='relu'),
```

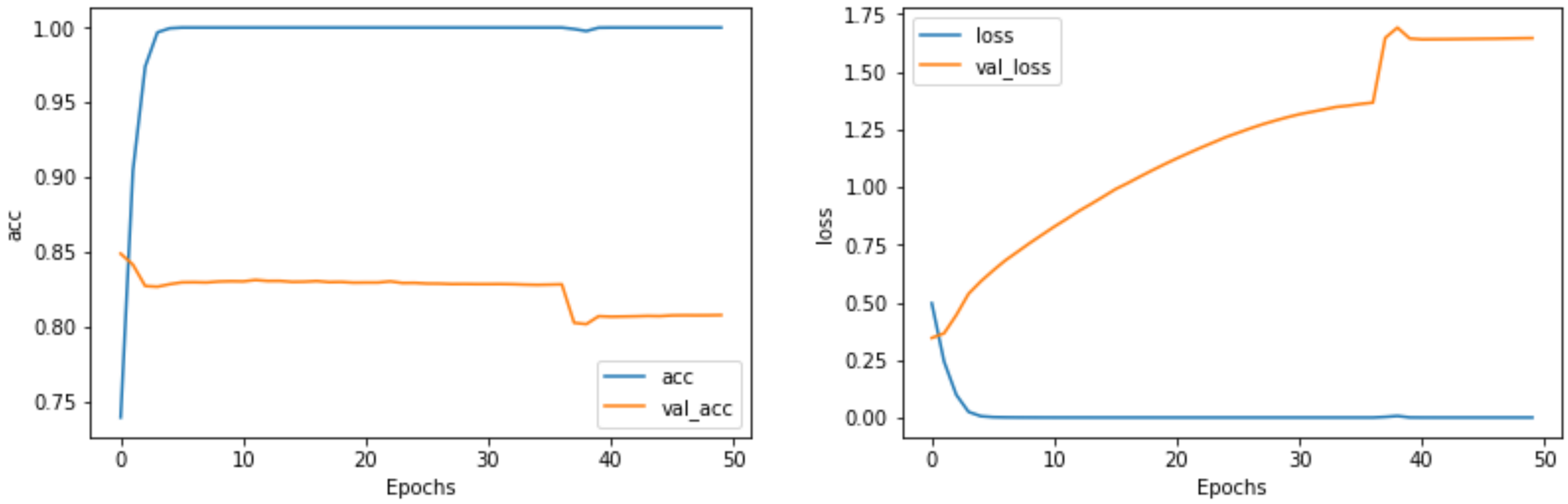
```
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 120, 16)           16000
_________________________________________________________________
conv1d (Conv1D)              (None, 116, 128)          10368
_________________________________________________________________
global_max_pooling1d (Global (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 24)                3096
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 25
=================================================================
Total params: 29,489
Trainable params: 29,489
Non-trainable params: 0
```

```
max_length = 120

tf.keras.layers.Conv1D(128, 5, activation='relu'),
```

```
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 120, 16)           16000
_________________________________________________________________
conv1d (Conv1D)              (None, 116, 128)          10368
_________________________________________________________________
global_max_pooling1d (Global (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 24)                3096
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 25
=================================================================
Total params: 29,489
Trainable params: 29,489
Non-trainable params: 0
```

```
imdb, info = tfds.load("imdb_reviews", with_info=True, as_supervised=True)




model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(6, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])

model.summary()
```

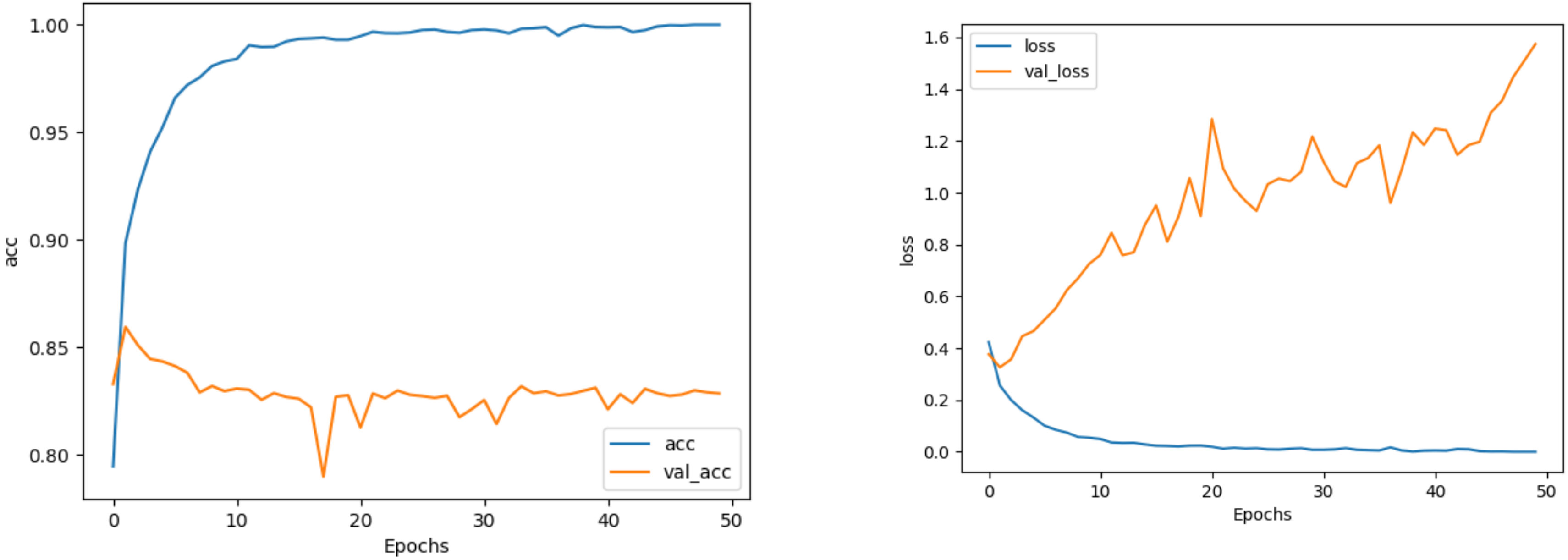
```
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 120, 16)           160000
_________________________________________________________________
flatten (Flatten)            (None, 1920)              0
_________________________________________________________________
dense (Dense)                (None, 6)                 11526
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 7
=================================================================
Total params: 171,533
Trainable params: 171,533
Non-trainable params: 0
```



IMDB with Embedding-only : ~ 5s per epoch

```
imdb, info = tfds.load("imdb_reviews", with_info=True, as_supervised=True)



# Model Definition with LSTM
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Dense(6, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
model.summary()
```

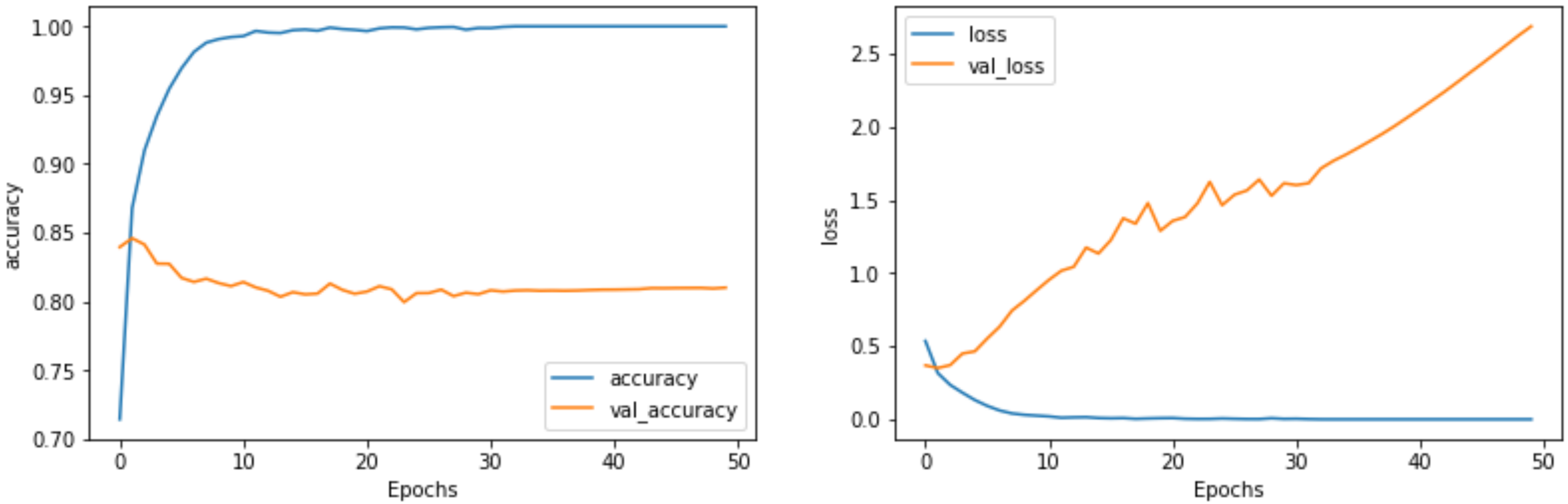
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_7 (Embedding)      (None, 120, 16)           160000
_________________________________________________________________
bidirectional_7 (Bidirection (None, 64)                12544
_________________________________________________________________
dense_14 (Dense)             (None, 6)                 390
_________________________________________________________________
dense_15 (Dense)             (None, 1)                 7
=================================================================
Total params: 173,941
Trainable params: 172,941
Non-trainable params: 0
```



IMDB with LSTM ~43s per epoch

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.Bidirectional(tf.keras.layers.GRU(32)),
    tf.keras.layers.Dense(6, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])

model.summary()
```

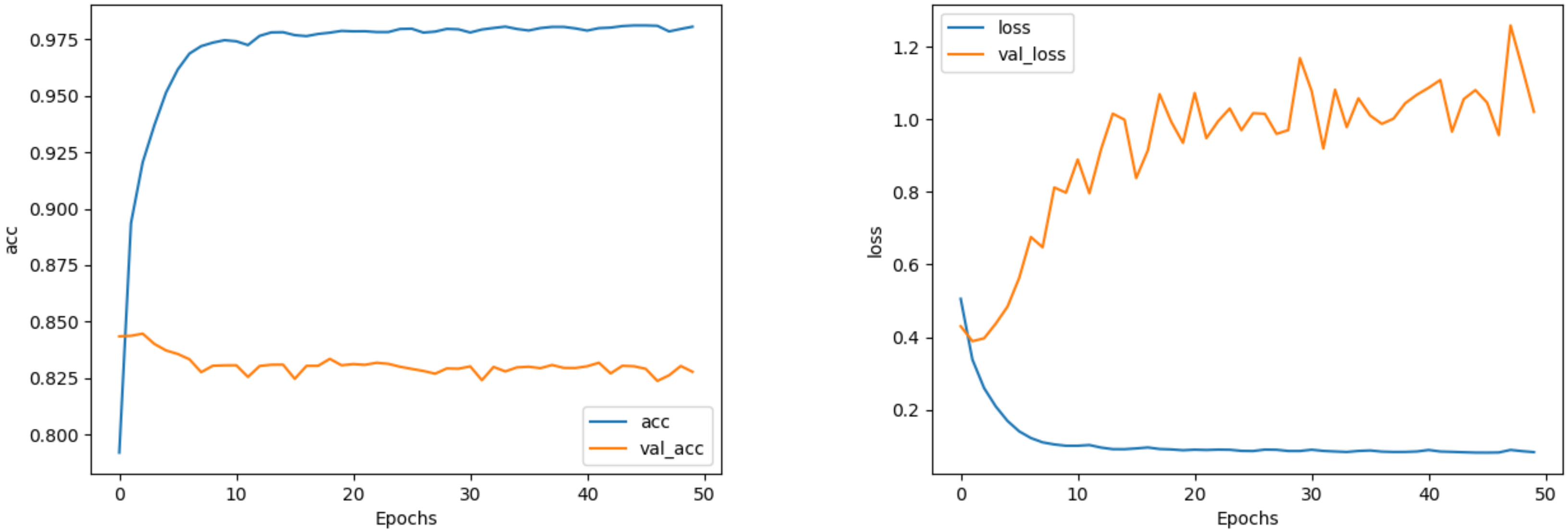
```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 120, 16)           160000
_________________________________________________________________
bidirectional_1 (Bidirection (None, 64)                9600
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 390
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 7
=================================================================
Total params: 169,997
Trainable params: 169,997
Non-trainable params: 0
```



IMDB with GRU : ~ 20s per epoch

```
# Model Definition with Conv1D
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.Conv1D(128, 5, activation='relu'),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(6, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
model.summary()
```

```
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 120, 16)           160000
_________________________________________________________________
conv1d (Conv1D)              (None, 116, 128)          10368
_________________________________________________________________
global_average_pooling1d (Gl (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 6)                 774
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 7
=================================================================
Total params: 171,149
Trainable params: 171,149
Non-trainable params: 0
_________________________________________________________________
```



IMDB with CNN : ~ 6s per epoch

Copyright Notice

In the town of Athy one Jeremy Lanigan
Battered away til he hadnt a pound.
His father died and made him a man again
Left him a farm and ten acres of ground.

He gave a grand party for friends and relations
Who didnt forget him when come to the wall,
And if youll but listen Ill make your eyes glisten
Of the rows and the ructions of Lanigan's Ball.

Myself to be sure got free invitation,
For all the nice girls and boys I might ask,
And just in a minute both friends and relations
Were dancing round merry as bees round a cask.

Judy ODaly, that nice little milliner,
She tipped me a wink for to give her a call,
And I soon arrived with Peggy McGilligan
Just in time for Lanigans Ball.

```
tokenizer = Tokenizer()

data="In the town of Athy one Jeremy Lanigan \n Battered away ... ..."
corpus = data.lower().split("\n")

tokenizer.fit_on_texts(corpus)
total_words = len(tokenizer.word_index) + 1
```

```
tokenizer = Tokenizer()

data="In the town of Athy one Jeremy Lanigan \n Battered away ... ..."
corpus = data.lower().split("\n")

tokenizer.fit_on_texts(corpus)
total_words = len(tokenizer.word_index) + 1
```

```
tokenizer = Tokenizer()

data="In the town of Athy one Jeremy Lanigan \n Battered away ... ..."
corpus = data.lower().split("\n")

tokenizer.fit_on_texts(corpus)
total_words = len(tokenizer.word_index) + 1
```

```
tokenizer = Tokenizer()

data="In the town of Athy one Jeremy Lanigan \n Battered away ... ..."
corpus = data.lower().split("\n")

tokenizer.fit_on_texts(corpus)
total_words = len(tokenizer.word_index) + 1
```

```
tokenizer = Tokenizer()

data="In the town of Athy one Jeremy Lanigan \n Battered away ... ..."
corpus = data.lower().split("\n")

tokenizer.fit_on_texts(corpus)
total_words = len(tokenizer.word_index) + 1
```

```
input_sequences = []
for line in corpus:
    token_list = tokenizer.texts_to_sequences([line])[0]
    for i in range(1, len(token_list)):
        n_gram_sequence = token_list[:i+1]
        input_sequences.append(n_gram_sequence)
```

```
input_sequences = []
for line in corpus:
    token_list = tokenizer.texts_to_sequences([line])[0]
    for i in range(1, len(token_list)):
        n_gram_sequence = token_list[:i+1]
        input_sequences.append(n_gram_sequence)
```

```
input_sequences = []
for line in corpus:
    token_list = tokenizer.texts_to_sequences([line])[0]
    for i in range(1, len(token_list)):
        n_gram_sequence = token_list[:i+1]
        input_sequences.append(n_gram_sequence)
```

In the town of Athy one Jeremy Lanigan

↓

[4 2 66 8 67 68 69 70]

```
input_sequences = []
for line in corpus:
    token_list = tokenizer.texts_to_sequences([line])[0]
    for i in range(1, len(token_list)):
        n_gram_sequence = token_list[:i+1]
        input_sequences.append(n_gram_sequence)
```

Line:

[4 2 66 8 67 68 69 70]

Input Sequences:

[4 2]

[4 2 66]

[4 2 66 8]

[4 2 66 8 67]

[4 2 66 8 67 68]

[4 2 66 8 67 68 69]

[4 2 66 8 67 68 69 70]

```
max_sequence_len = max([len(x) for x in input_sequences])
```

```
input_sequences =
    np.array(pad_sequences(input_sequences, maxlen=max_sequence_len, padding='pre'))
```

Line:

[4 2 66 8 67 68 69 70]

Padded Input Sequences:

[0 0 0 0 0 0 0 0 0 0 4 2]

[0 0 0 0 0 0 0 0 0 4 2 66]

[0 0 0 0 0 0 0 0 4 2 66 8]

[0 0 0 0 0 0 0 4 2 66 8 67]

[0 0 0 0 0 0 4 2 66 8 67 68]

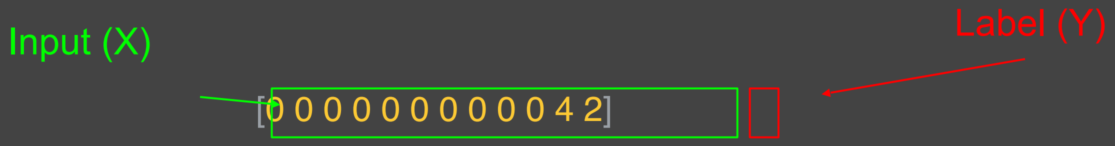
[0 0 0 0 0 4 2 66 8 67 68 69]

[0 0 0 0 4 2 66 8 67 68 69 70]

Padded Input Sequences:

```
[0 0 0 0 0 0 0 0 0 0 4 2]

[0 0 0 0 0 0 0 0 0 4 2 66]

[0 0 0 0 0 0 0 0 4 2 66 8]

[0 0 0 0 0 0 0 4 2 66 8 67]

[0 0 0 0 0 0 4 2 66 8 67 68]

[0 0 0 0 0 4 2 66 8 67 68 69]

[0 0 0 0 4 2 66 8 67 68 69 70]
```

Padded Input Sequences:

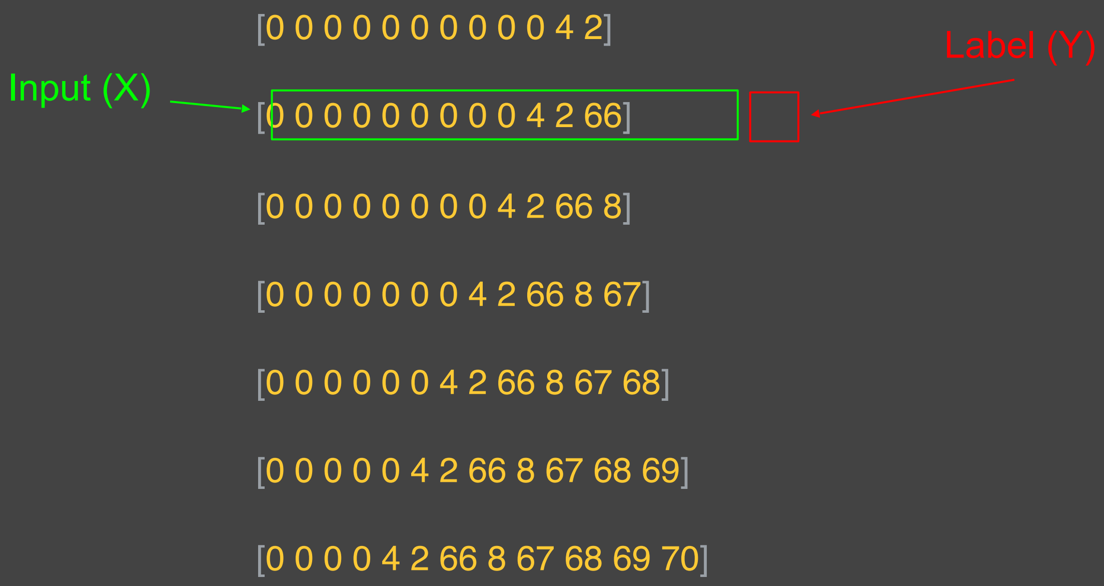


Input (X)

Label (Y)

```
[0 0 0 0 0 0 0 0 0 0 4 2]

[0 0 0 0 0 0 0 0 0 4 2 66]

[0 0 0 0 0 0 0 0 4 2 66 8]

[0 0 0 0 0 0 0 4 2 66 8 67]

[0 0 0 0 0 0 4 2 66 8 67 68]

[0 0 0 0 0 4 2 66 8 67 68 69]

[0 0 0 0 4 2 66 8 67 68 69 70]
```

Padded Input Sequences:



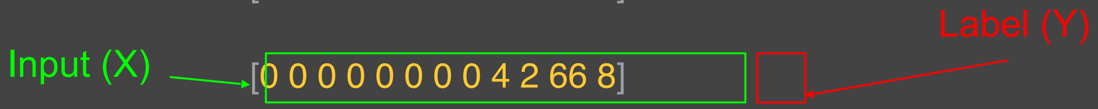
```
[0 0 0 0 0 0 0 0 0 0 4 2]
[0 0 0 0 0 0 0 0 0 4 2 66]
[0 0 0 0 0 0 0 0 4 2 66 8]
[0 0 0 0 0 0 0 4 2 66 8 67]
[0 0 0 0 0 0 4 2 66 8 67 68]
[0 0 0 0 0 4 2 66 8 67 68 69]
[0 0 0 0 4 2 66 8 67 68 69 70]
```

Input (X)

Label (Y)

Padded Input Sequences:

```
[0 0 0 0 0 0 0 0 0 0 4 2]

[0 0 0 0 0 0 0 0 0 4 2 66]
```



```
[0 0 0 0 0 0 0 0 4 2 66 8]
```

```
[0 0 0 0 0 0 0 4 2 66 8 67]

[0 0 0 0 0 0 4 2 66 8 67 68]

[0 0 0 0 0 4 2 66 8 67 68 69]

[0 0 0 0 4 2 66 8 67 68 69 70]
```

```
xs = input_sequences[:,:-1]
labels = input_sequences[:,-1]
```

```
ys = tf.keras.utils.to_categorical(labels, num_classes=total_words)
```

```
Sentence: [0 0 0 0 4 2 66 8 67 68 69 70]

       X:[0 0 0 0 4 2 66 8 67 68 69]

   Label:[ 70 ]

       Y: [ 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.  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.  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.  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.  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.  0.  0.
            0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
```

```
Sentence: [0 0 0 0 4 2 66 8 67 68 69 70]

       X:[0 0 0 0 4 2 66 8 67 68 69]

   Label:[ 70 ]

      Y: [ 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.  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.  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.  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.  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.  0.  0.
           0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
```

```
model = Sequential()
model.add(Embedding(total_words, 64, input_length=max_sequence_len - 1))
model.add((LSTM(20)))
model.add(Dense(total_words, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(xs, ys, epochs=500, verbose=1)
```

```
model = Sequential()
model.add(Embedding(total_words, 64, input_length=max_sequence_len - 1))
model.add((LSTM(20)))
model.add(Dense(total_words, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(xs, ys, epochs=500, verbose=1)
```

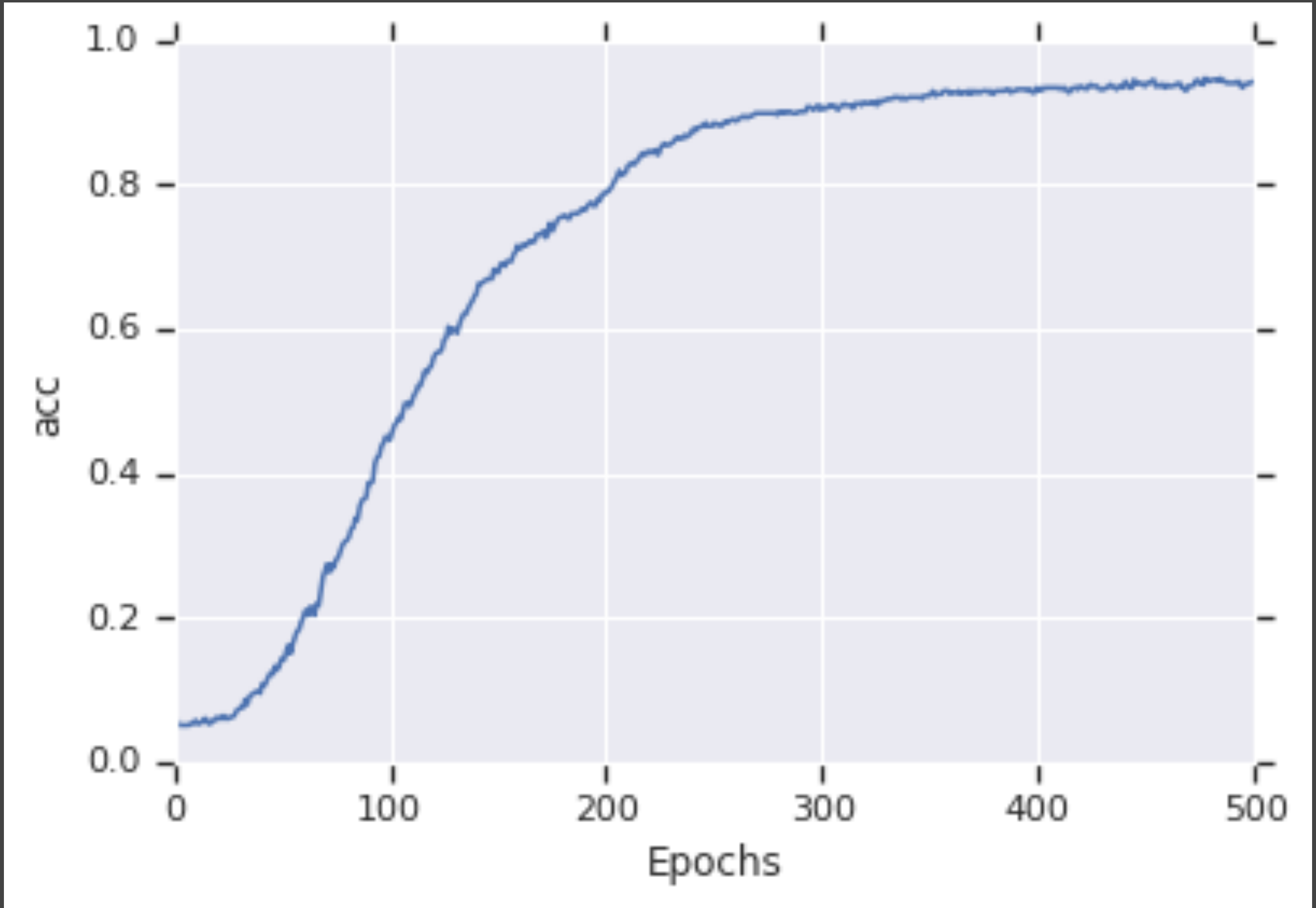
```
model = Sequential()
model.add(Embedding(total_words, 64, input_length=max_sequence_len - 1))
model.add((LSTM(20)))
model.add(Dense(total_words, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(xs, ys, epochs=500, verbose=1)
```

```
model = Sequential()
model.add(Embedding(total_words, 64, input_length=max_sequence_len - 1))
model.add((LSTM(20)))
model.add(Dense(total_words, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(xs, ys, epochs=500, verbose=1)
```

```
model = Sequential()
model.add(Embedding(total_words, 64, input_length=max_sequence_len - 1))
model.add((LSTM(20)))
model.add(Dense(total_words, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(xs, ys, epochs=500, verbose=1)
```

```
model = Sequential()
model.add(Embedding(total_words, 64, input_length=max_sequence_len - 1))
model.add((LSTM(20)))
model.add(Dense(total_words, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(xs, ys, epochs=500, verbose=1)
```

```
model = Sequential()
model.add(Embedding(total_words, 64, input_length=max_sequence_len - 1))
model.add((LSTM(20)))
model.add(Dense(total_words, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(xs, ys, epochs=500, verbose=1)
```


1.0
0.8
0.6
0.4
0.2
0.0
acc
0
100
200
300
400
500
Epochs

Laurence went to dublin round the plenty as red wall me for wall wall
Laurence went to dublin odaly of the nice of lanigans ball ball ball hall
Laurence went to dublin he hadnt a minute both relations hall new relations youd

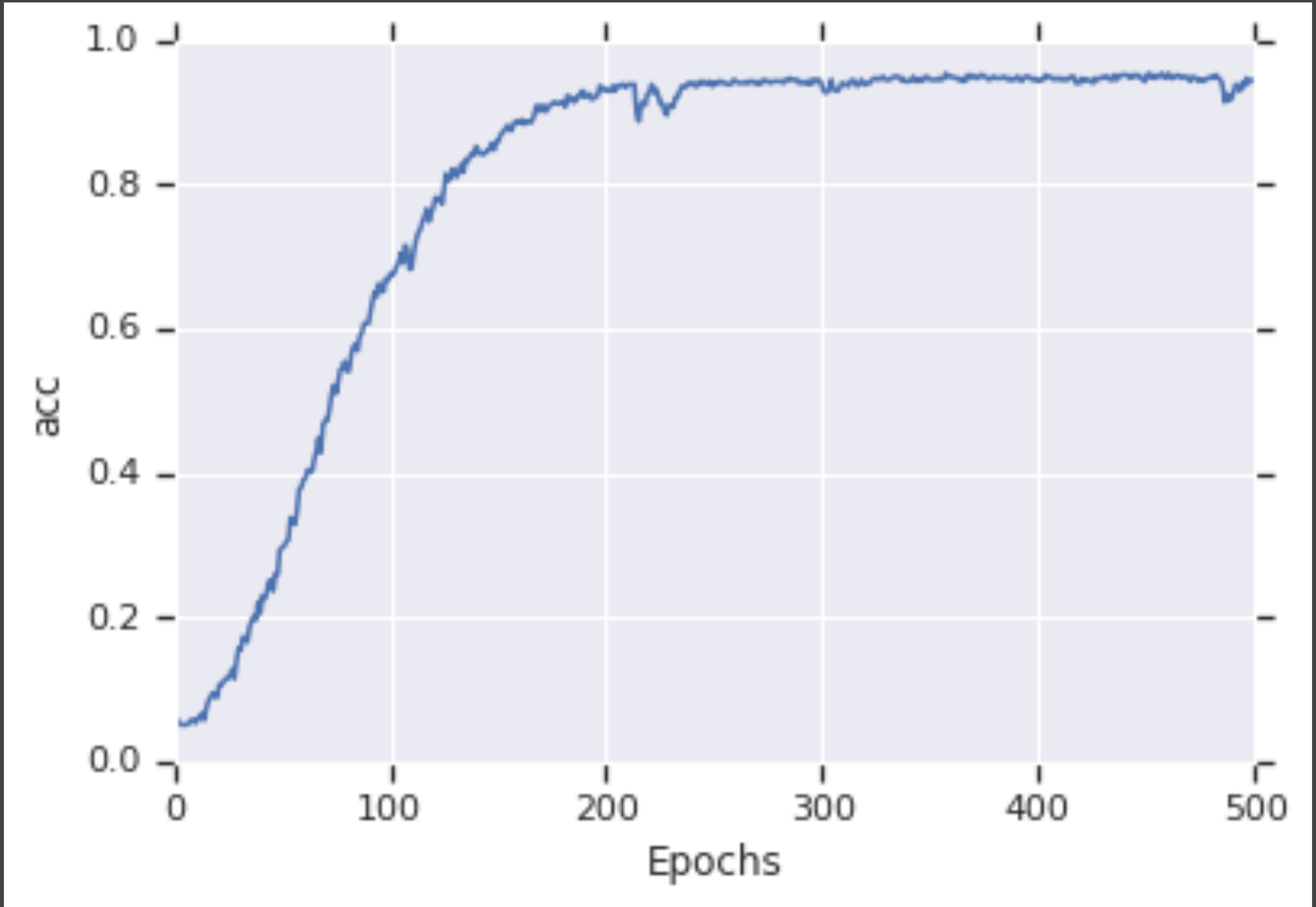
Laurence went to dublin round the plenty as red wall me for wall wall
Laurence went to dublin odaly of the nice of lanigans ball ball ball hall
Laurence went to dublin he hadnt a minute both relations hall new relations youd

Laurence went to dublin round the plenty as red wall me for wall wall
Laurence went to dublin odaly of the nice of lanigans ball ball ball hall
Laurence went to dublin he hadnt a minute both relations hall new relations youd

Laurence went to dublin round the plenty as red wall me for wall wall
Laurence went to dublin odaly of the nice of lanigans ball ball ball hall
Laurence went to dublin he hadnt a minute both relations hall new relations youd

```
model = Sequential()
model.add(Embedding(total_words, 64, input_length=max_sequence_len - 1))
model.add(Bidirectional(LSTM(20)))
model.add(Dense(total_words, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(xs, ys, epochs=500, verbose=1)
```

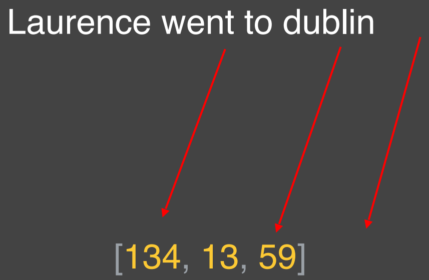
```
model = Sequential()
model.add(Embedding(total_words, 64, input_length=max_sequence_len - 1))
model.add(Bidirectional(LSTM(20)))
model.add(Dense(total_words, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(xs, ys, epochs=500, verbose=1)
```


1.0
0.8
0.6
0.4
0.2
0.0
acc
0
100
200
300
400
500
Epochs

Laurence went to dublin think and wine for lanigans ball entangled in nonsense me
Laurence went to dublin his pipes bellows chanters and all all entangled all kinds
Laurence went to dublin how the room a whirligig ructions long at brooks fainted

Laurence went to dublin

```
token_list = tokenizer.texts_to_sequences([seed_text])[0]
```



Laurence went to dublin

[134, 13, 59]

```
token_list = pad_sequences([token_list], maxlen=max_sequence_len - 1, padding='pre')
```

```
[  0   0   0   0   0   0   0 134  13  59]
```

```
predicted = model.predict(token_list)
predicted = np.argmax(probabilities, axis= - 1)[0]
```

```
output_word = tokenizer.index_word[predicted]
seed_text += " " + output_word
seed_text = "Laurence went to dublin"
next_words = 10

for _ in range(next_words):
  token_list = tokenizer.texts_to_sequences([seed_text])[0]
  token_list = pad_sequences([token_list], maxlen=max_sequence_len - 1, padding='pre')
  predicted = model.predict_classes(token_list, verbose=0)
  output_word = output_word = tokenizer.index_word[predicted]
  seed_text += " " + output_word
print(seed_text)
```

Laurence went to dublin round a cask cask cask cask cask squeezed forget tea twas make eyes glisten mchugh mchugh lanigan lanigan glisten

```
!wget --no-check-certificate \
    https://storage.googleapis.com/laurencemoroney-blog.appspot.com/irish-lyrics-eof.txt \
    -O /tmp/irish-lyrics-eof.txt
```

```
data = open('/tmp/irish-lyrics-eof.txt').read()
```

```
model = Sequential()
model.add(Embedding(total_words, 100, input_length=max_sequence_len-1))
model.add(Bidirectional(LSTM(150)))
model.add(Dense(total_words, activation='softmax'))
adam = Adam(lr=0.01)
model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
history = model.fit(xs, ys, epochs=100, verbose=1)
```

```
model = Sequential()
model.add(Embedding(total_words, 100, input_length=max_sequence_len-1))
model.add(Bidirectional(LSTM(150)))
model.add(Dense(total_words, activation='softmax'))
adam = Adam(lr=0.01)
model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
history = model.fit(xs, ys, epochs=100, verbose=1)
```

```
model = Sequential()
model.add(Embedding(total_words, 100, input_length=max_sequence_len-1))
model.add(Bidirectional(LSTM(150)))
model.add(Dense(total_words, activation='softmax'))
adam = Adam(lr=0.01)
model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
history = model.fit(xs, ys, epochs=100, verbose=1)
```

```
model = Sequential()
model.add(Embedding(total_words, 100, input_length=max_sequence_len-1))
model.add(Bidirectional(LSTM(150)))
model.add(Dense(total_words, activation='softmax'))
adam = Adam(lr=0.01)
model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
history = model.fit(xs, ys, epochs=100, verbose=1)
```

```
model = Sequential()
model.add(Embedding(total_words, 100, input_length=max_sequence_len-1))
model.add(Bidirectional(LSTM(150)))
model.add(Dense(total_words, activation='softmax'))
adam = Adam(lr=0.01)
model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
history = model.fit(xs, ys, epochs=100, verbose=1)
```

Help Me Obi-Wan Kenobi, you're my only hope
my dear
and hope as i did fly with its flavours
along with all its joys
but sure i will build
love you still
gold it did join
do mans run away cross our country
are wedding i was down to
off holyhead wished meself
down among the pigs
played some hearty rigs
me embarrass
find me brother
me chamber she gave me
who storied be irishmen
to greet you
lovely molly
gone away from me home
home to leave the old tin cans
the foemans chain one was shining
sky above i think i love

https://www.tensorflow.org/tutorials/sequences/text_generation