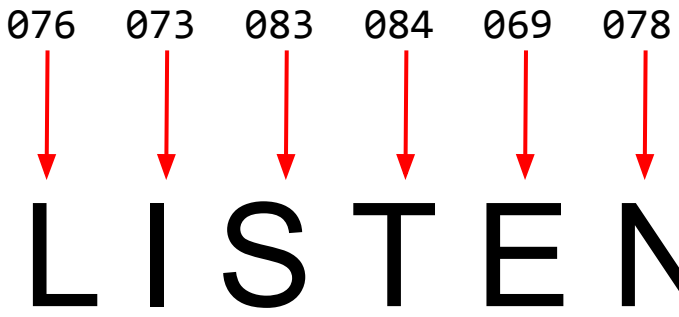
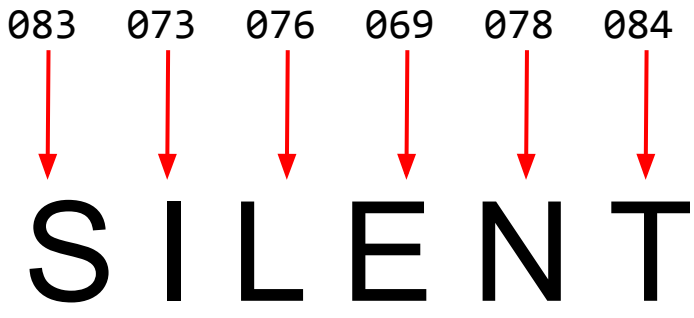
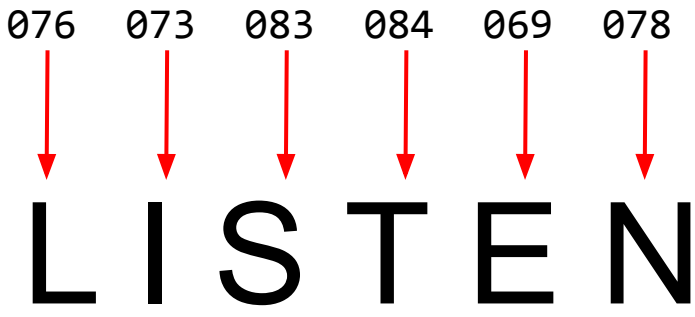
Copyright Notice

L I S T E N


076 073 083 084 069 078
L I S T E N

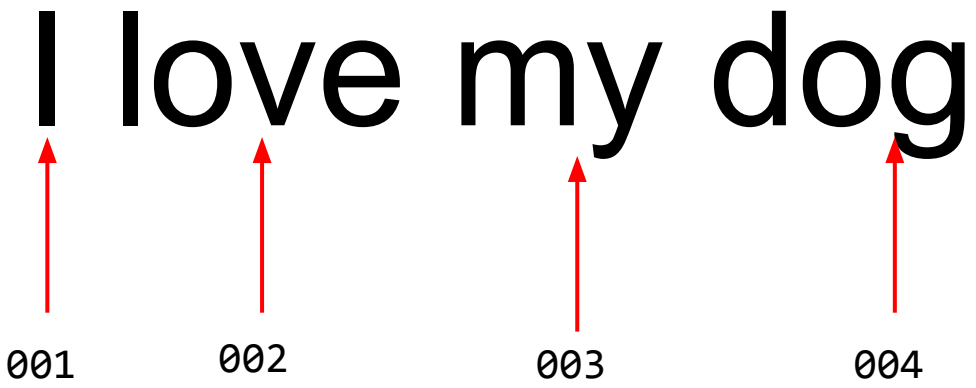

083 073 076 069 078 084
SILENT


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

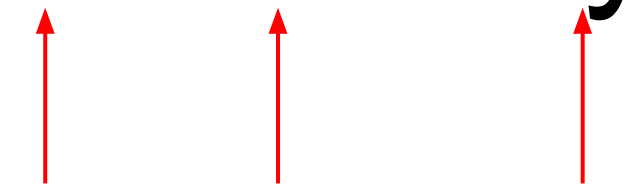
001 002 003 004

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

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

vectorize_layer = tf.keras.layers.TextVectorization()

vectorize_layer.adapt(sentences)

vocabulary = vectorize_layer.get_vocabulary(include_special_tokens=False)

print(vocabulary)
```

```
import tensorflow as tf

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

vectorize_layer = tf.keras.layers.TextVectorization()

vectorize_layer.adapt(sentences)

vocabulary = vectorize_layer.get_vocabulary(include_special_tokens=False)

print(vocabulary)
```

```
import tensorflow as tf

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

vectorize_layer = tf.keras.layers.TextVectorization()

vectorize_layer.adapt(sentences)

vocabulary = vectorize_layer.get_vocabulary(include_special_tokens=False)

print(vocabulary)
```

```
import tensorflow as tf

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

vectorize_layer = tf.keras.layers.TextVectorization()

vectorize_layer.adapt(sentences)

vocabulary = vectorize_layer.get_vocabulary(include_special_tokens=False)

print(vocabulary)
```

```
import tensorflow as tf

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

vectorize_layer = tf.keras.layers.TextVectorization()

vectorize_layer.adapt(sentences)

vocabulary = vectorize_layer.get_vocabulary(include_special_tokens=False)

print(vocabulary)
```

```
import tensorflow as tf

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

vectorize_layer = tf.keras.layers.TextVectorization()

vectorize_layer.adapt(sentences)

vocabulary = vectorize_layer.get_vocabulary(include_special_tokens=False)

print(vocabulary)
```

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

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

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

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

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

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

```
import tensorflow as tf

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

vectorize_layer = tf.keras.layers.TextVectorization()

vectorize_layer.adapt(sentences)

vocabulary = vectorize_layer.get_vocabulary(include_special_tokens=False)

print(vocabulary)
```

```
import tensorflow as tf

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

vectorize_layer = tf.keras.layers.TextVectorization()

vectorize_layer.adapt(sentences)

vocabulary = vectorize_layer.get_vocabulary()

print(vocabulary)
```

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

```
import tensorflow as tf

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

vectorize_layer = tf.keras.layers.TextVectorization()

vectorize_layer.adapt(sentences)

vocabulary = vectorize_layer.get_vocabulary()

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

```
import tensorflow as tf

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

vectorize_layer = tf.keras.layers.TextVectorization()

vectorize_layer.adapt(sentences)

vocabulary = vectorize_layer.get_vocabulary()

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

```
import tensorflow as tf

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

vectorize_layer = tf.keras.layers.TextVectorization()

vectorize_layer.adapt(sentences)

vocabulary = vectorize_layer.get_vocabulary()

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

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

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

```
import tensorflow as tf

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

vectorize_layer = tf.keras.layers.TextVectorization()

vectorize_layer.adapt(sentences)

vocabulary = vectorize_layer.get_vocabulary()

sequence = vectorize_layer('I love my dog')

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

print(sequence)
```

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

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

```
print(sequence)
```

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

```
import tensorflow as tf

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

vectorize_layer = tf.keras.layers.TextVectorization()

vectorize_layer.adapt(sentences)

vocabulary = vectorize_layer.get_vocabulary()

sequences = vectorize_layer(sentences)

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

print(sequences)
```

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

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

print(sequences)

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

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

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

print(sequences)

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

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

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

print(sequences)

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

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

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

print(sequences)

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

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

```
print(sequences)
```

OUTPUT:

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

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

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

sequences = vectorize_layer(sentences)

print(sequences)

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

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

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

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

sequences = sentences_dataset.map(vectorize_layer)
```

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

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

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

sequences = sentences_dataset.map(vectorize_layer)
```

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

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

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

sequences = sentences_dataset.map(vectorize_layer)

print(sequences)

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

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

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

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

sequences = sentences_dataset.map(vectorize_layer)

print(sequences)
```

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

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

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

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

sequences = sentences_dataset.map(vectorize_layer)

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

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

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

sequences = sentences_dataset.map(vectorize_layer)



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

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

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

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

sequences = sentences_dataset.map(vectorize_layer)



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

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

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

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

sequences = sentences_dataset.map(vectorize_layer)

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

print(sequences_pre)
```

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

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

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

sequences = sentences_dataset.map(vectorize_layer)

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

print(sequences_pre)
```

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

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

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

sequences = sentences_dataset.map(vectorize_layer)

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

print(sequences_pre)
```

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

```
import tensorflow as tf

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

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

vectorize_layer.adapt(sentences)

vocabulary = vectorize_layer.get_vocabulary()

ragged_sequences = vectorize_layer(sentences)

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

print(ragged_sequences)
```

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

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

```
print(ragged_sequences)
```

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

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

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

```
print(ragged_sequences)
```

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

```
import tensorflow as tf

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

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

vectorize_layer.adapt(sentences)

vocabulary = vectorize_layer.get_vocabulary()

ragged_sequences = vectorize_layer(sentences)

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

print(pre_padded_sequences)
```

```
print(pre_padded_sequences)

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

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

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

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

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

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

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

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

test_seq = vectorize_layer(test_data)

print(test_seq)
```

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

test_seq = vectorize_layer(test_data)

print(test_seq)


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

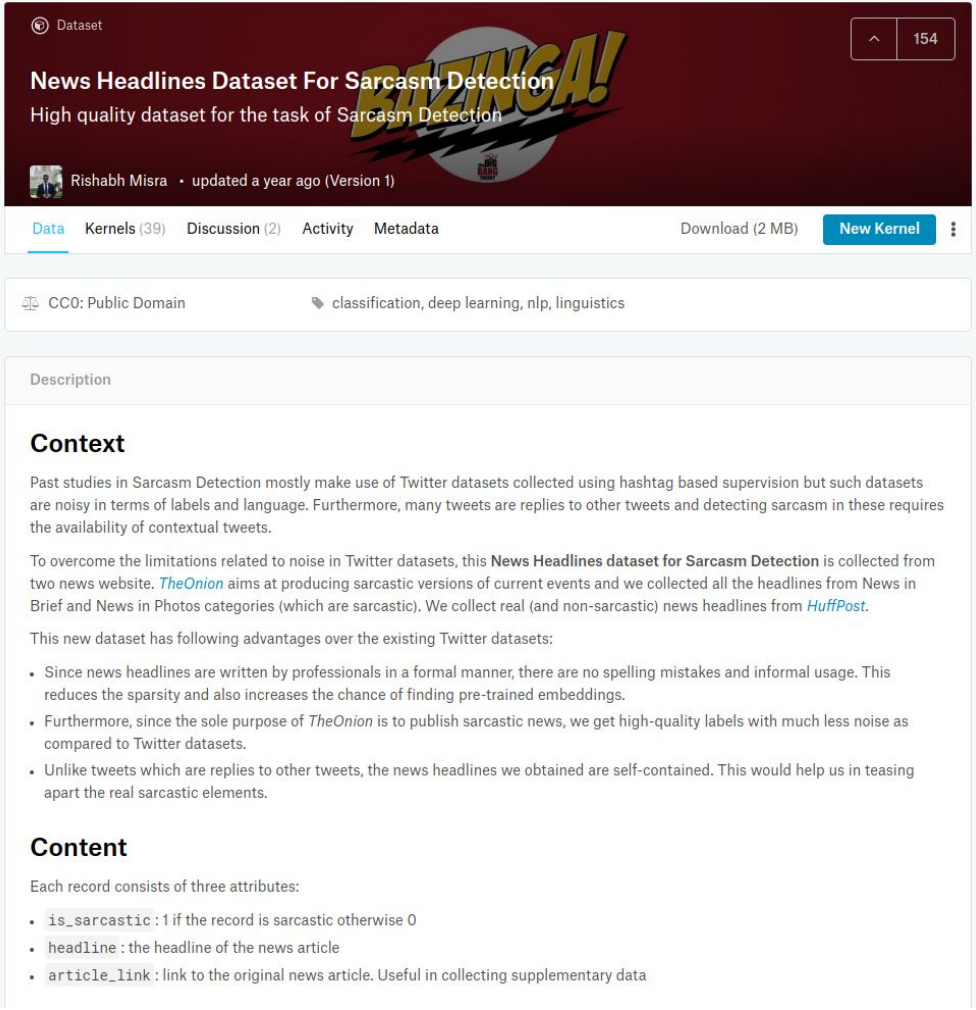
```
test_data = [
    'i really love my dog',
    'my dog loves my manatee'
]

test_seq = vectorize_layer(test_data)

print(test_seq)
```

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

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

Sarcasm in News Headlines Dataset by Rishabh Misra

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



is_sarcastic: 1 if the record is sarcastic otherwise 0

headline: the headline of the news article

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

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

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

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

```
[

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

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

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

]
```

```
import json

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

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

```
import json

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

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

```
import json

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

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

```
import json

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

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

```
import json

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

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

```
import tensorflow as tf

vectorize_layer = tf.keras.layers.TextVectorization()

vectorize_layer.adapt(sentences)

vocabulary = vectorize_layer.get_vocabulary()

post_padded_sequences = vectorize_layer(sentences)

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

```
padded sequence: [140 825 2 813 1100 2048 571 5057 199 139 39 46 2
13050 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

```
import tensorflow as tf

vectorize_layer = tf.keras.layers.TextVectorization()

vectorize_layer.adapt(sentences)

vocabulary = vectorize_layer.get_vocabulary()

post_padded_sequences = vectorize_layer(sentences)

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

```
padded sequence: [140 825 2 813 1100 2048 571 5057 199 139 39 46 2
13050 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

```
import tensorflow as tf

vectorize_layer = tf.keras.layers.TextVectorization()

vectorize_layer.adapt(sentences)

vocabulary = vectorize_layer.get_vocabulary()

post_padded_sequences = vectorize_layer(sentences)

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

```
padded sequence: [140 825 2 813 1100 2048 571 5057 199 139 39 46 2
13050 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

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_datasets as tfds

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

```
import tensorflow_datasets as tfds

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

single_example = list(imdb['train'].take(1))[0]

print(single_example[0])
```

```
tf.Tensor(b"This was an absolutely terrible movie. Don't be lured in by Christopher Walken or Michael Ironside. Both are great actors, but this must simply be their worst role in history. Even their great acting could not redeem this movie's ridiculous storyline. This movie is an early nineties US propaganda piece. The most pathetic scenes were those when the Columbian rebels were making their cases for revolutions. Maria Conchita Alonso appeared phony, and her pseudo-love affair with Walken was nothing but a pathetic emotional plug in a movie that was devoid of any real meaning. I am disappointed that there are movies like this, ruining actor's like Christopher Walken's good name. I could barely sit through it.", shape=(), dtype=string)
```

```
import tensorflow_datasets as tfds

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

single_example = list(imdb['train'].take(1))[0]

print(single_example[0])
```

```
tf.Tensor(b"This was an absolutely terrible movie. Don't be lured in by Christopher Walken or Michael Ironside. Both are great actors, but this must simply be their worst role in history. Even their great acting could not redeem this movie's ridiculous storyline. This movie is an early nineties US propaganda piece. The most pathetic scenes were those when the Columbian rebels were making their cases for revolutions. Maria Conchita Alonso appeared phony, and her pseudo-love affair with Walken was nothing but a pathetic emotional plug in a movie that was devoid of any real meaning. I am disappointed that there are movies like this, ruining actor's like Christopher Walken's good name. I could barely sit through it.", shape=(), dtype=string)
```

```
import tensorflow_datasets as tfds

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

single_example = list(imdb['train'].take(1))[0]

print(single_example[1])
```

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

```
import tensorflow_datasets as tfds

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

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

train_reviews = train_dataset.map(lambda review, label: review)
train_labels = train_dataset.map(lambda review, label: label)

test_reviews = test_dataset.map(lambda review, label: review)
test_labels = test_dataset.map(lambda review, label: label)
```

```
import tensorflow_datasets as tfds

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

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

train_reviews = train_dataset.map(lambda review, label: review)
train_labels = train_dataset.map(lambda review, label: label)

test_reviews = test_dataset.map(lambda review, label: review)
test_labels = test_dataset.map(lambda review, label: label)
```

```
import tensorflow_datasets as tfds

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

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

train_reviews = train_dataset.map(lambda review, label: review)
train_labels = train_dataset.map(lambda review, label: label)

test_reviews = test_dataset.map(lambda review, label: review)
test_labels = test_dataset.map(lambda review, label: label)
```

```
vectorize_layer = tf.keras.layers.TextVectorization(max_tokens=10000)

vectorize_layer.adapt(train_reviews)

def padding_func(sequences):
    sequences = sequences.ragged_batch(batch_size=sequences.cardinality())
    sequences = sequences.get_single_element()

    padded_sequences = tf.keras.utils.pad_sequences(sequences.numpy(), maxlen=120,
                                                    truncating='post', padding='pre')

    padded_sequences = tf.data.Dataset.from_tensor_slices(padded_sequences)

    return padded_sequences

train_sequences = train_reviews.map(lambda text: vectorize_layer(text)).apply(padding_func)
test_sequences = test_reviews.map(lambda text: vectorize_layer(text)).apply(padding_func)
```

```
vectorize_layer = tf.keras.layers.TextVectorization(max_tokens=10000)

vectorize_layer.adapt(train_reviews)

def padding_func(sequences):
    sequences = sequences.ragged_batch(batch_size=sequences.cardinality())
    sequences = sequences.get_single_element()

    padded_sequences = tf.keras.utils.pad_sequences(sequences.numpy(), maxlen=120,
                                                    truncating='post', padding='pre')

    padded_sequences = tf.data.Dataset.from_tensor_slices(padded_sequences)

    return padded_sequences

train_sequences = train_reviews.map(lambda text: vectorize_layer(text)).apply(padding_func)
test_sequences = test_reviews.map(lambda text: vectorize_layer(text)).apply(padding_func)
```

```
vectorize_layer = tf.keras.layers.TextVectorization(max_tokens=10000)

vectorize_layer.adapt(train_reviews)

def padding_func(sequences):
    sequences = sequences.ragged_batch(batch_size=sequences.cardinality())
    sequences = sequences.get_single_element()

    padded_sequences = tf.keras.utils.pad_sequences(sequences.numpy(), maxlen=120,
                                                    truncating='post', padding='pre')

    padded_sequences = tf.data.Dataset.from_tensor_slices(padded_sequences)

    return padded_sequences

train_sequences = train_reviews.map(lambda text: vectorize_layer(text)).apply(padding_func)
test_sequences = test_reviews.map(lambda text: vectorize_layer(text)).apply(padding_func)
```

```
vectorize_layer = tf.keras.layers.TextVectorization(max_tokens=10000)

vectorize_layer.adapt(train_reviews)

def padding_func(sequences):
    sequences = sequences.ragged_batch(batch_size=sequences.cardinality())
    sequences = sequences.get_single_element()

    padded_sequences = tf.keras.utils.pad_sequences(sequences.numpy(), maxlen=120,
                                                    truncating='post', padding='pre')

    padded_sequences = tf.data.Dataset.from_tensor_slices(padded_sequences)

    return padded_sequences

train_sequences = train_reviews.map(lambda text: vectorize_layer(text)).apply(padding_func)
test_sequences = test_reviews.map(lambda text: vectorize_layer(text)).apply(padding_func)
```

```
vectorize_layer = tf.keras.layers.TextVectorization(max_tokens=10000)

vectorize_layer.adapt(train_reviews)

def padding_func(sequences):
    sequences = sequences.ragged_batch(batch_size=sequences.cardinality())
    sequences = sequences.get_single_element()

    padded_sequences = tf.keras.utils.pad_sequences(sequences.numpy(), maxlen=120,
                                                    truncating='post', padding='pre')

    padded_sequences = tf.data.Dataset.from_tensor_slices(padded_sequences)

    return padded_sequences

train_sequences = train_reviews.map(lambda text: vectorize_layer(text)).apply(padding_func)
test_sequences = test_reviews.map(lambda text: vectorize_layer(text)).apply(padding_func)
```

```
vectorize_layer = tf.keras.layers.TextVectorization(max_tokens=10000)

vectorize_layer.adapt(train_reviews)

def padding_func(sequences):
    sequences = sequences.ragged_batch(batch_size=sequences.cardinality())
    sequences = sequences.get_single_element()

    padded_sequences = tf.keras.utils.pad_sequences(sequences.numpy(), maxlen=120,
                                                    truncating='post', padding='pre')

    padded_sequences = tf.data.Dataset.from_tensor_slices(padded_sequences)

    return padded_sequences

train_sequences = train_reviews.map(lambda text: vectorize_layer(text)).apply(padding_func)
test_sequences = test_reviews.map(lambda text: vectorize_layer(text)).apply(padding_func)
```

```
vectorize_layer = tf.keras.layers.TextVectorization(max_tokens=10000)

vectorize_layer.adapt(train_reviews)

def padding_func(sequences):
    sequences = sequences.ragged_batch(batch_size=sequences.cardinality())
    sequences = sequences.get_single_element()

    padded_sequences = tf.keras.utils.pad_sequences(sequences.numpy(), maxlen=120,
                                                    truncating='post', padding='pre')

    padded_sequences = tf.data.Dataset.from_tensor_slices(padded_sequences)

    return padded_sequences

train_sequences = train_reviews.map(lambda text: vectorize_layer(text)).apply(padding_func)
test_sequences = test_reviews.map(lambda text: vectorize_layer(text)).apply(padding_func)
```

```
vectorize_layer = tf.keras.layers.TextVectorization(max_tokens=10000)

vectorize_layer.adapt(train_reviews)

def padding_func(sequences):
    sequences = sequences.ragged_batch(batch_size=sequences.cardinality())
    sequences = sequences.get_single_element()

    padded_sequences = tf.keras.utils.pad_sequences(sequences.numpy(), maxlen=120,
                                                    truncating='post', padding='pre')

    padded_sequences = tf.data.Dataset.from_tensor_slices(padded_sequences)

    return padded_sequences

train_sequences = train_reviews.map(lambda text: vectorize_layer(text)).apply(padding_func)
test_sequences = test_reviews.map(lambda text: vectorize_layer(text)).apply(padding_func)
```

```
vectorize_layer = tf.keras.layers.TextVectorization(max_tokens=10000)

vectorize_layer.adapt(train_reviews)

def padding_func(sequences):
    sequences = sequences.ragged_batch(batch_size=sequences.cardinality())
    sequences = sequences.get_single_element()

    padded_sequences = tf.keras.utils.pad_sequences(sequences.numpy(), maxlen=120,
                                                    truncating='post', padding='pre')

    padded_sequences = tf.data.Dataset.from_tensor_slices(padded_sequences)

    return padded_sequences

train_sequences = train_reviews.map(lambda text: vectorize_layer(text)).apply(padding_func)
test_sequences = test_reviews.map(lambda text: vectorize_layer(text)).apply(padding_func)
```

```
vectorize_layer = tf.keras.layers.TextVectorization(max_tokens=10000)

vectorize_layer.adapt(train_reviews)

def padding_func(sequences):
    sequences = sequences.ragged_batch(batch_size=sequences.cardinality())
    sequences = sequences.get_single_element()

    padded_sequences = tf.keras.utils.pad_sequences(sequences.numpy(), maxlen=120,
                                                    truncating='post', padding='pre')

    padded_sequences = tf.data.Dataset.from_tensor_slices(padded_sequences)

    return padded_sequences

train_sequences = train_reviews.map(lambda text: vectorize_layer(text)).apply(padding_func)
test_sequences = test_reviews.map(lambda text: vectorize_layer(text)).apply(padding_func)
```

```
train_dataset_vectorized = tf.data.Dataset.zip(train_sequences, train_labels)
test_dataset_vectorized = tf.data.Dataset.zip(test_sequences, test_labels)

SHUFFLE_BUFFER_SIZE = 1000
PREFETCH_BUFFER_SIZE = tf.data.AUTOTUNE
BATCH_SIZE = 32

train_dataset_final = (train_dataset_vectorized
                       .cache()
                       .shuffle(SHUFFLE_BUFFER_SIZE)
                       .prefetch(PREFETCH_BUFFER_SIZE)
                       .batch(BATCH_SIZE)
                       )

test_dataset_final = (test_dataset_vectorized
                      .cache()
                      .prefetch(PREFETCH_BUFFER_SIZE)
                      .batch(BATCH_SIZE)
                      )
```

```
train_dataset_vectorized = tf.data.Dataset.zip(train_sequences, train_labels)
test_dataset_vectorized = tf.data.Dataset.zip(test_sequences, test_labels)

SHUFFLE_BUFFER_SIZE = 1000
PREFETCH_BUFFER_SIZE = tf.data.AUTOTUNE
BATCH_SIZE = 32

train_dataset_final = (train_dataset_vectorized
                       .cache()
                       .shuffle(SHUFFLE_BUFFER_SIZE)
                       .prefetch(PREFETCH_BUFFER_SIZE)
                       .batch(BATCH_SIZE)
                       )

test_dataset_final = (test_dataset_vectorized
                      .cache()
                      .prefetch(PREFETCH_BUFFER_SIZE)
                      .batch(BATCH_SIZE)
                      )
```

```
train_dataset_vectorized = tf.data.Dataset.zip(train_sequences, train_labels)
test_dataset_vectorized = tf.data.Dataset.zip(test_sequences, test_labels)

SHUFFLE_BUFFER_SIZE = 1000
PREFETCH_BUFFER_SIZE = tf.data.AUTOTUNE
BATCH_SIZE = 32

train_dataset_final = (train_dataset_vectorized
                       .cache()
                       .shuffle(SHUFFLE_BUFFER_SIZE)
                       .prefetch(PREFETCH_BUFFER_SIZE)
                       .batch(BATCH_SIZE)
                       )

test_dataset_final = (test_dataset_vectorized
                      .cache()
                      .prefetch(PREFETCH_BUFFER_SIZE)
                      .batch(BATCH_SIZE)
                      )
```

```
model = tf.keras.Sequential([
    tf.keras.Input(shape=(120,)),
    tf.keras.layers.Embedding(vocab_size, embedding_dim),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(6, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
model = tf.keras.Sequential([
    tf.keras.Input(shape=(120,)),
    tf.keras.layers.Embedding(vocab_size, embedding_dim),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(6, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
model = tf.keras.Sequential([
    tf.keras.Input(shape=(120,)),
    tf.keras.layers.Embedding(vocab_size, embedding_dim),
    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.Input(shape=(120,)),
    tf.keras.layers.Embedding(vocab_size, embedding_dim),
    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(train_dataset_final,
          epochs=num_epochs,
          validation_data=test_dataset_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
```

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

```
(10000, 16)
```

```
import io

out_v = io.open('vecs.tsv', 'w', encoding='utf-8')
out_m = io.open('meta.tsv', 'w', encoding='utf-8')

vocabulary = vectorize_layer.get_vocabulary()

for word_num in range(1, len(vocabulary)):
    word_name = vocabulary[word_num]
    word_embedding = embedding_weights[word_num]
    out_m.write(word_name + "\n")
    out_v.write('\t'.join([str(x) for x in word_embedding]) + "\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')

vocabulary = vectorize_layer.get_vocabulary()

for word_num in range(1, len(vocabulary)):
    word_name = vocabulary[word_num]
    word_embedding = embedding_weights[word_num]
    out_m.write(word_name + "\n")
    out_v.write('\t'.join([str(x) for x in word_embedding]) + "\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')

vocabulary = vectorize_layer.get_vocabulary()

for word_num in range(1, len(vocabulary)):
    word_name = vocabulary[word_num]
    word_embedding = embedding_weights[word_num]
    out_m.write(word_name + "\n")
    out_v.write('\t'.join([str(x) for x in word_embedding]) + "\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')

vocabulary = vectorize_layer.get_vocabulary()

for word_num in range(1, len(vocabulary)):
    word_name = vocabulary[word_num]
    word_embedding = embedding_weights[word_num]
    out_m.write(word_name + "\n")
    out_v.write('\t'.join([str(x) for x in word_embedding]) + "\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')

vocabulary = vectorize_layer.get_vocabulary()

for word_num in range(1, len(vocabulary)):
    word_name = vocabulary[word_num]
    word_embedding = embedding_weights[word_num]
    out_m.write(word_name + "\n")
    out_v.write('\t'.join([str(x) for x in word_embedding]) + "\n")

out_v.close()
out_m.close()
```

https://projector.tensorflow.org

Embedding Projector

DATA

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

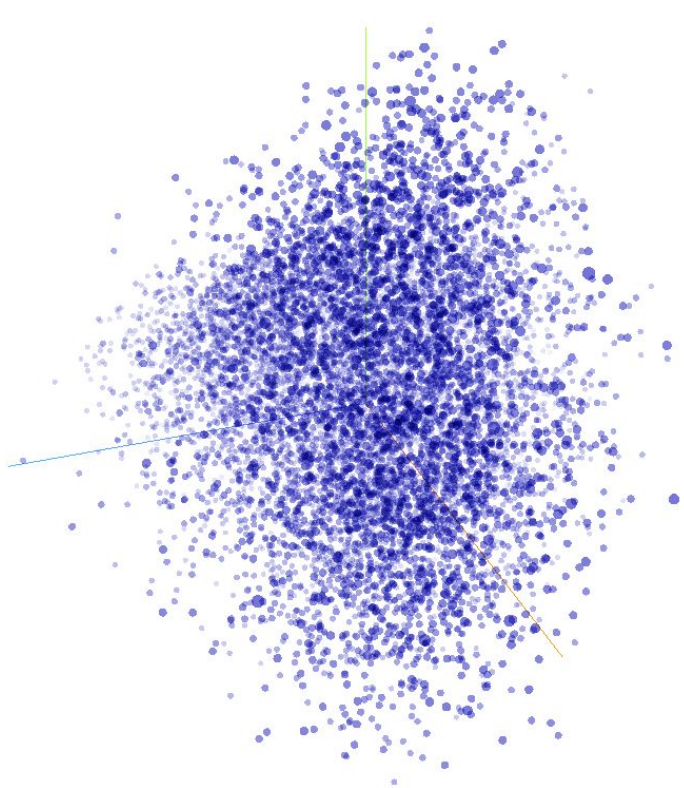
T-SNE | PCA | CUSTOM

X: Component #1
Y: Component #2
Z: Component #3

PCA is approximate.

Total variance described: 8.5%.

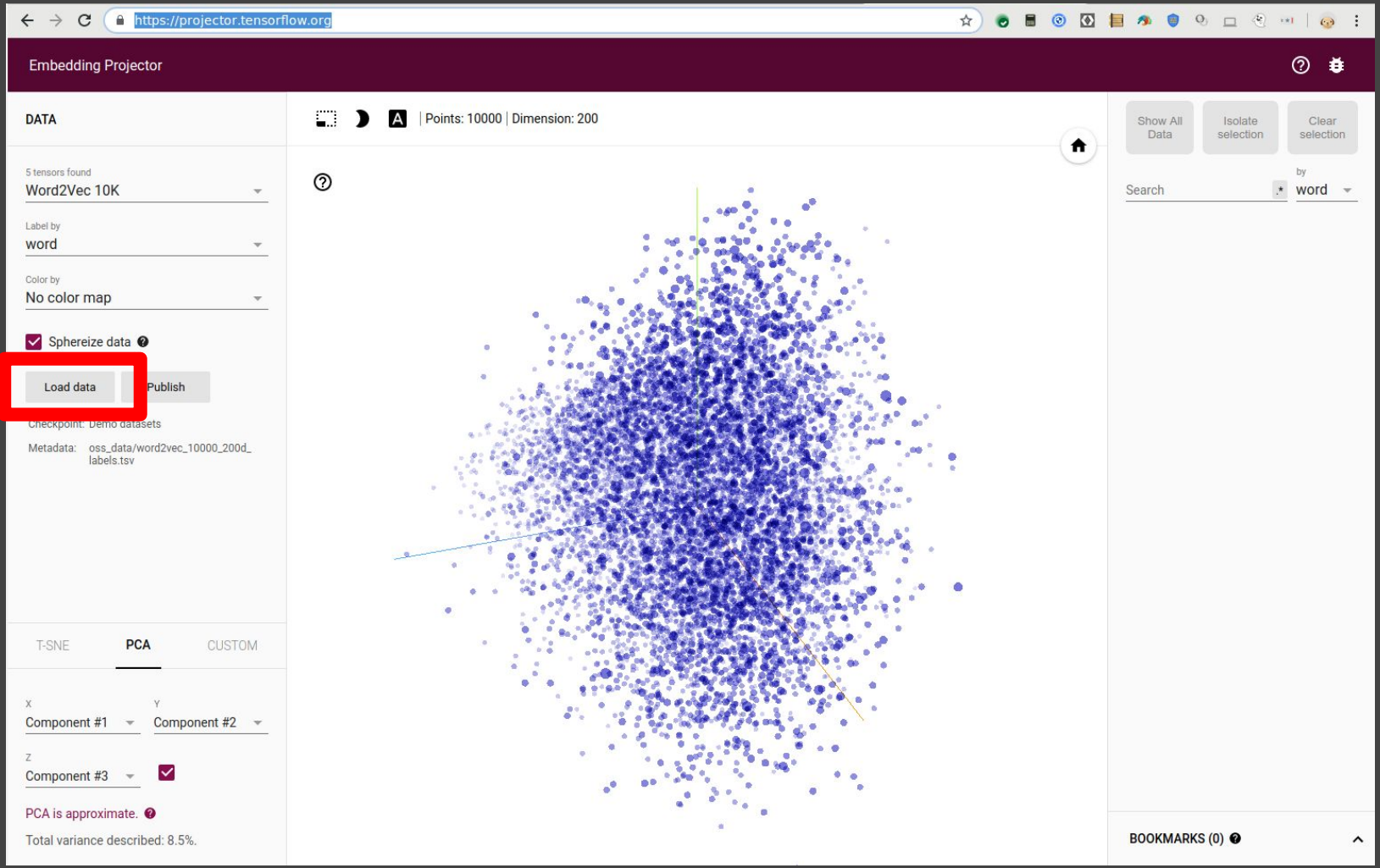
Points: 10000 | Dimension: 200



Show All Data | Isolate selection | Clear selection

Search | by word

BOOKMARKS (0)


https://projector.tensorflow.org
Embedding Projector
DATA
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
T-SNE
PCA
CUSTOM
X
Component #1
Y
Component #2
Z
Component #3
PCA is approximate.
Total variance described: 8.5%.
Points: 10000 | Dimension: 200
Show All Data
Isolate selection
Clear selection
Search
by
word
BOOKMARKS (0)

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
```

Choose file

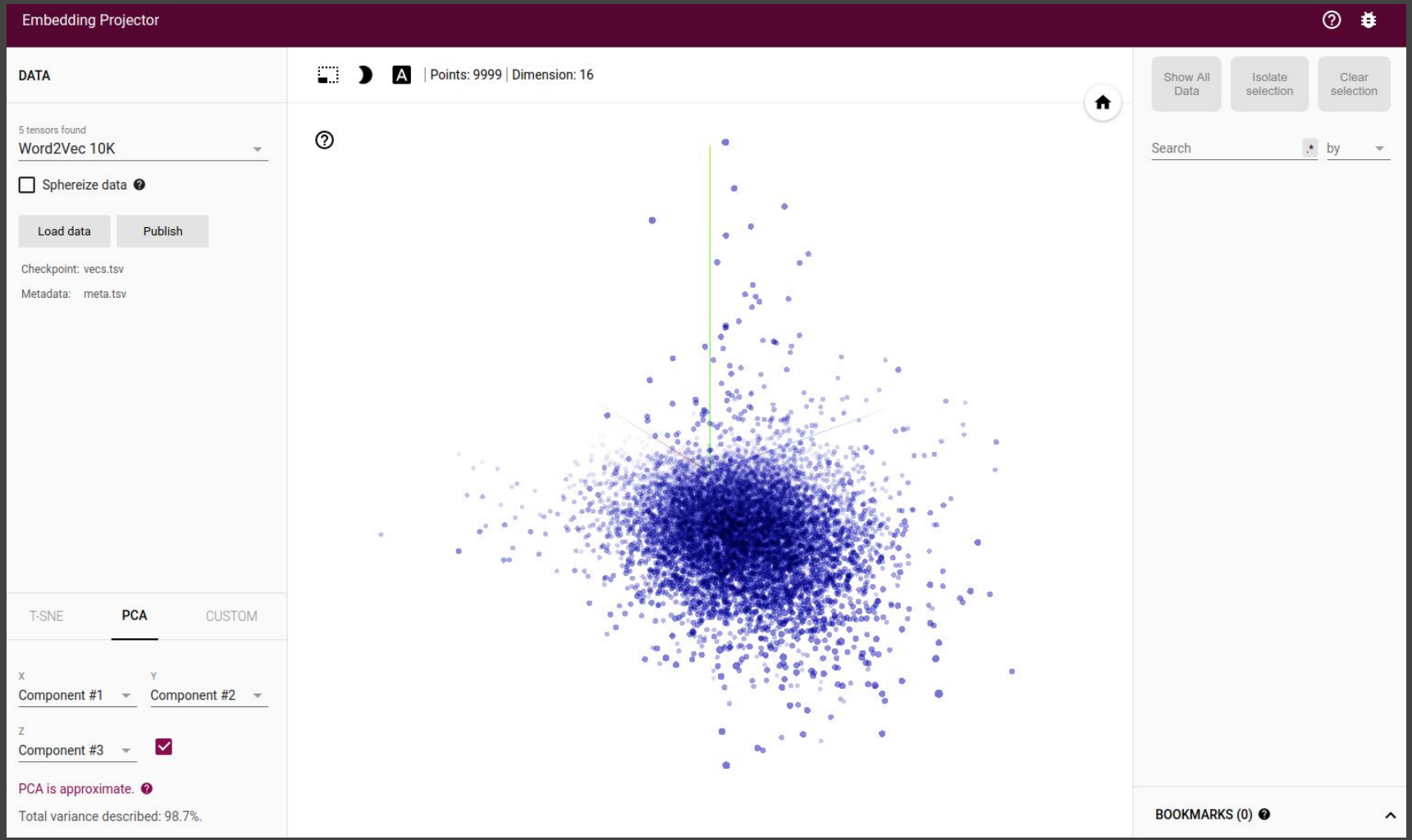
Step 2 (optional): **Load a TSV file of metadata.**

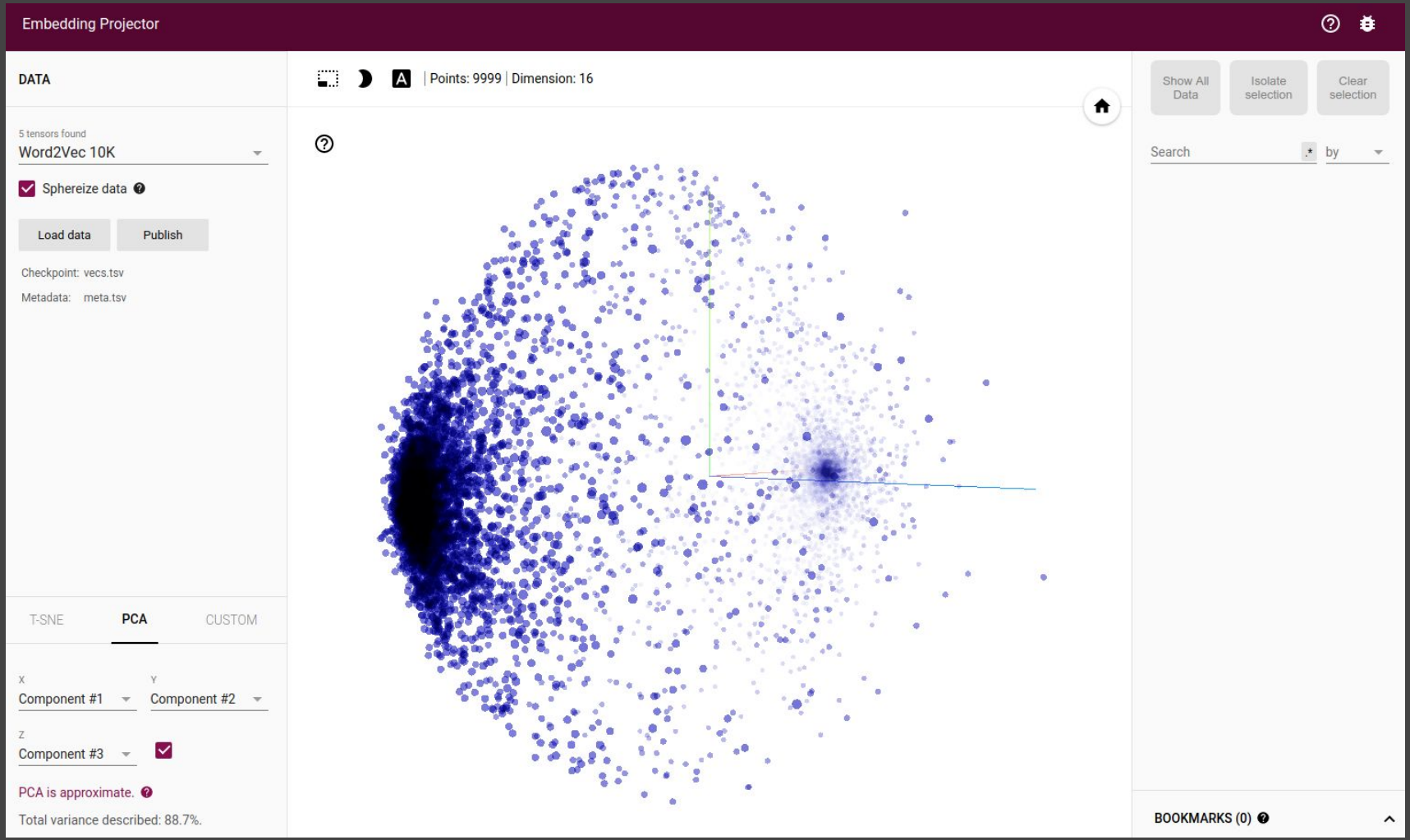
Example of 3 data points and 2 columns.
Note: If there is more than one column, the first row will be parsed as column labels.

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

Choose file

Click outside to dismiss.


Embedding Projector
DATA
5 tensors found
Word2Vec 10K
Sphereize data
Load data
Publish
Checkpoint: vecs.tsv
Metadata: meta.tsv
T-SNE
PCA
CUSTOM
X
Component #1
Y
Component #2
Z
Component #3
PCA is approximate.
Total variance described: 98.7%.
Points: 9999 | Dimension: 16
Show All Data
Isolate selection
Clear selection
Search
by
BOOKMARKS (0)


Embedding Projector
DATA
5 tensors found
Word2Vec 10K
Sphereize data
Load data
Publish
Checkpoint: vecs.tsv
Metadata: meta.tsv
T-SNE
PCA
CUSTOM
X
Component #1
Y
Component #2
Z
Component #3
PCA is approximate.
Total variance described: 88.7%.
| Points: 9999 | Dimension: 16
Show All Data
Isolate selection
Clear selection
Search
.*
by
BOOKMARKS (0)

```
TRAINING_SIZE = 20000

VOCAB_SIZE = 10000

MAX_LENGTH = 32

EMBEDDING_DIM = 16
```

```
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:]
```

```
vectorize_layer = tf.keras.layers.TextVectorization(
    max_tokens=VOCAB_SIZE,
    output_sequence_length=MAX_LENGTH)

vectorize_layer.adapt(train_sentences)

train_sequences = vectorize_layer(train_sentences)
test_sequences = vectorize_layer(test_sentences)

train_dataset_vectorized = tf.data.Dataset.from_tensor_slices(
    (train_sequences, train_labels))
test_dataset_vectorized = tf.data.Dataset.from_tensor_slices(
    (test_sequences, test_labels))
```

```
vectorize_layer = tf.keras.layers.TextVectorization(
    max_tokens=VOCAB_SIZE,
    output_sequence_length=MAX_LENGTH)

vectorize_layer.adapt(train_sentences)

train_sequences = vectorize_layer(train_sentences)
test_sequences = vectorize_layer(test_sentences)

train_dataset_vectorized = tf.data.Dataset.from_tensor_slices(
    (train_sequences, train_labels))
test_dataset_vectorized = tf.data.Dataset.from_tensor_slices(
    (test_sequences, test_labels))
```

```
vectorize_layer = tf.keras.layers.TextVectorization(
    max_tokens=VOCAB_SIZE,
    output_sequence_length=MAX_LENGTH)

vectorize_layer.adapt(train_sentences)

train_sequences = vectorize_layer(train_sentences)
test_sequences = vectorize_layer(test_sentences)

train_dataset_vectorized = tf.data.Dataset.from_tensor_slices(
    (train_sequences, train_labels))
test_dataset_vectorized = tf.data.Dataset.from_tensor_slices(
    (test_sequences, test_labels))
```

```
vectorize_layer = tf.keras.layers.TextVectorization(
    max_tokens=VOCAB_SIZE,
    output_sequence_length=MAX_LENGTH)

vectorize_layer.adapt(train_sentences)

train_sequences = vectorize_layer(train_sentences)
test_sequences = vectorize_layer(test_sentences)

train_dataset_vectorized = tf.data.Dataset.from_tensor_slices(
    (train_sequences, train_labels))
test_dataset_vectorized = tf.data.Dataset.from_tensor_slices(
    (test_sequences, test_labels))
```

```
SHUFFLE_BUFFER_SIZE = 1000
PREFETCH_BUFFER_SIZE = tf.data.AUTOTUNE
BATCH_SIZE = 32

train_dataset_final = (train_dataset_vectorized
                       .cache()
                       .shuffle(SHUFFLE_BUFFER_SIZE)
                       .prefetch(PREFETCH_BUFFER_SIZE)
                       .batch(BATCH_SIZE)
                       )

test_dataset_final = (test_dataset_vectorized
                      .cache()
                      .prefetch(PREFETCH_BUFFER_SIZE)
                      .batch(BATCH_SIZE)
                      )
```

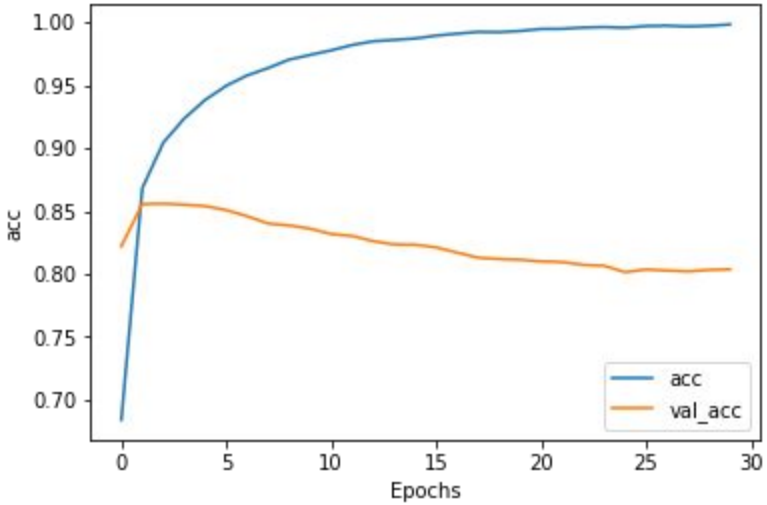
```
model = tf.keras.Sequential([
    tf.keras.Input(shape=(MAX_LENGTH,)),
    tf.keras.layers.Embedding(VOCAB_SIZE, EMBEDDING_DIM),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(6, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

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

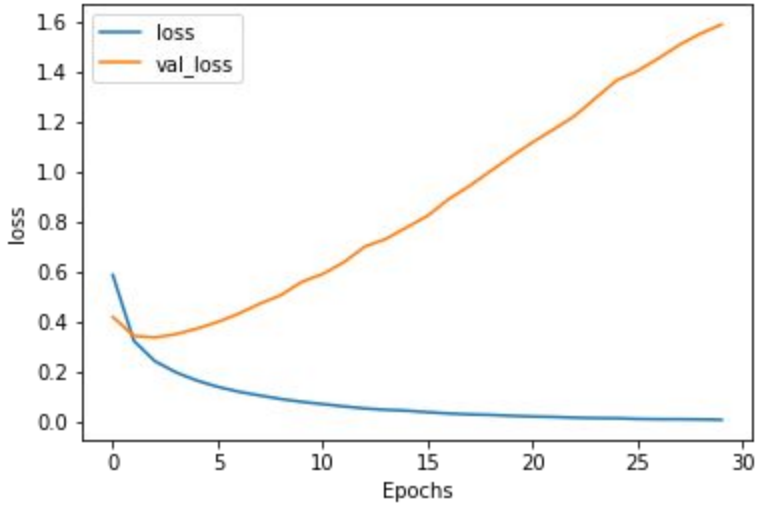
```
model.summary()
```

```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_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
```

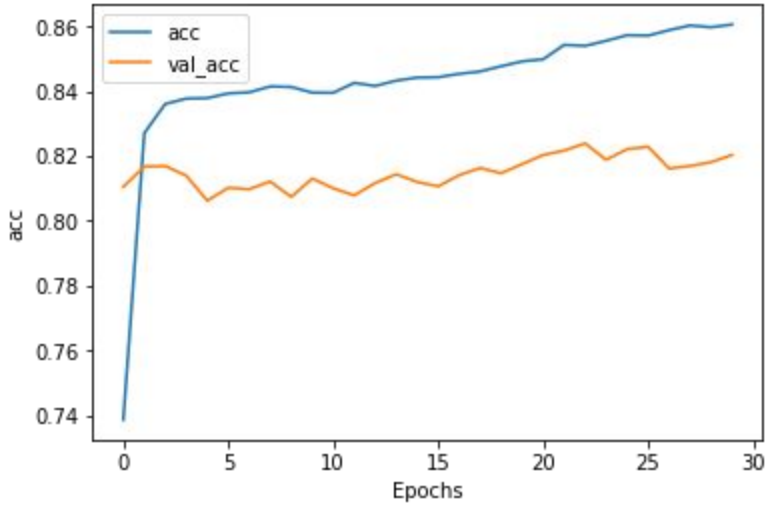
```
num_epochs = 30

history = model.fit(train_dataset_final, epochs=num_epochs,
                    validation_data=test_dataset_final, verbose=2)
```

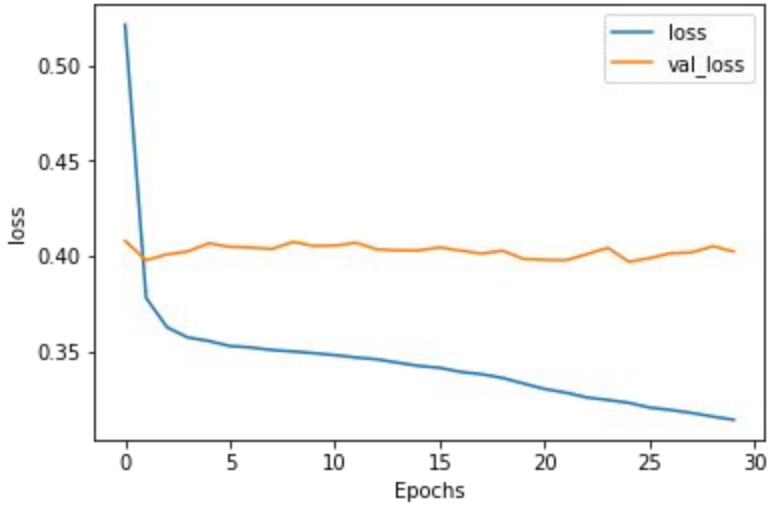
```
import matplotlib.pyplot as plt


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


acc
val_acc
Epochs
acc

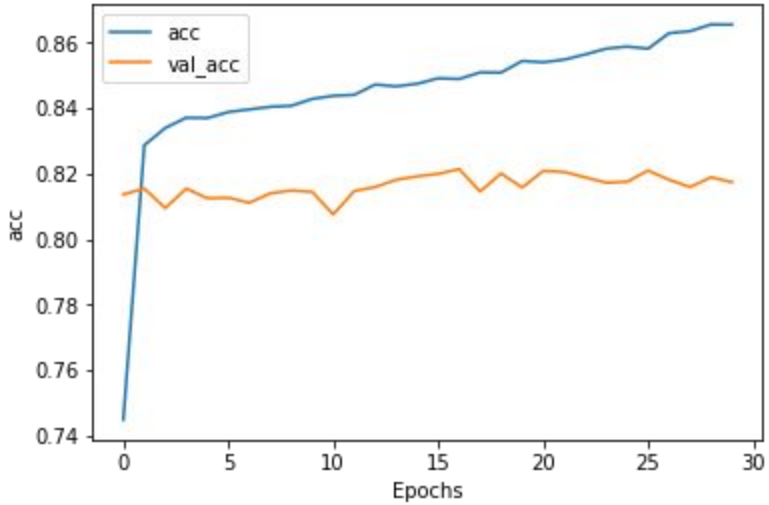

loss
val_loss
Epochs
loss

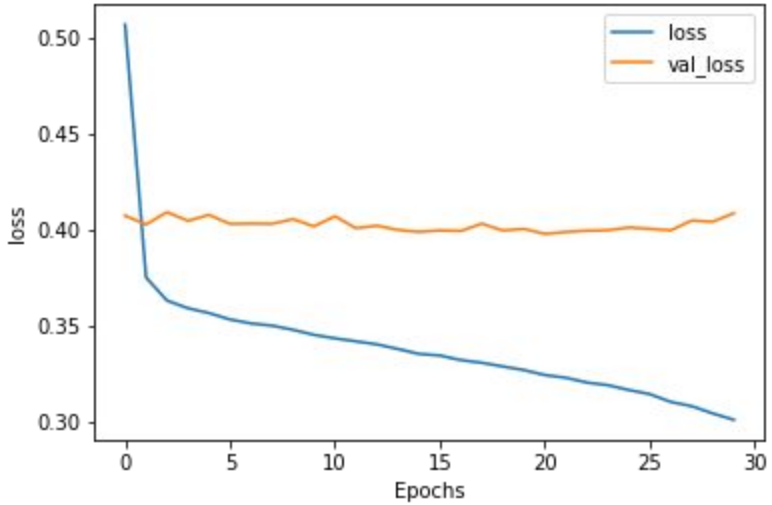
```
TRAINING_SIZE = 20000

VOCAB_SIZE = 1000 (was 10000)

MAX_LENGTH = 16   (was 32)

EMBEDDING_DIM = 16
```


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

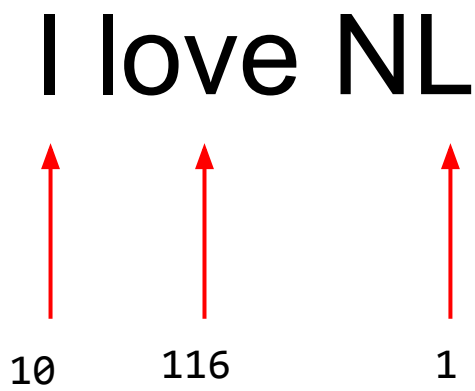

loss
val_loss
0.50
0.45
0.40
0.35
loss
0
5
10
15
20
25
30
Epochs

```
TRAINING_SIZE = 20000

VOCAB_SIZE = 1000  (was 10000)

MAX_LENGTH = 16    (was 32)

EMBEDDING_DIM = 32 (was 16)
```

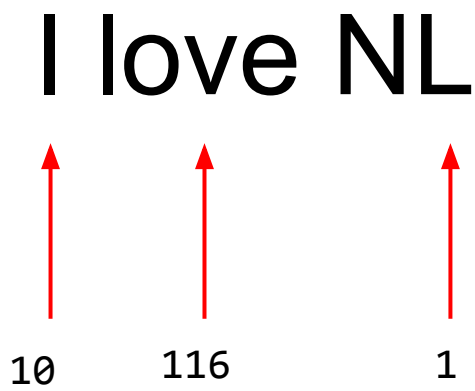

acc
val_acc
0.86
0.84
0.82
0.80
0.78
0.76
0.74
acc
0
5
10
15
20
25
30
Epochs


loss
val_loss
0.50
0.45
0.40
0.35
0.30
loss
0
5
10
15
20
25
30
Epochs

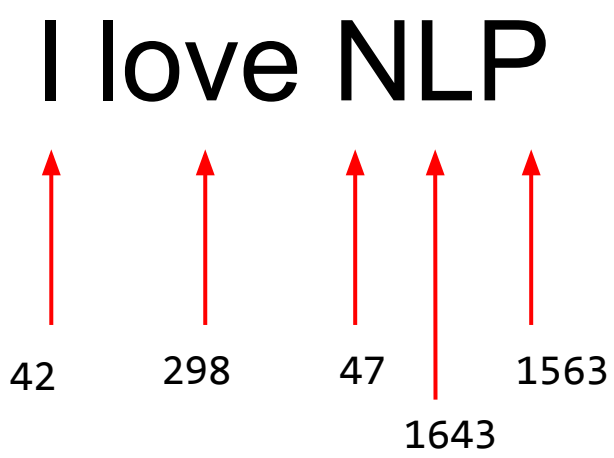
Word tokenization



Word tokenization



Subword tokenization



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

275 lines (209 sloc) | 9.65 KB

Raw Blame

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

275 lines (209 sloc) | 9.65 KB

Raw Blame

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 %}

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

275 lines (209 sloc) | 9.65 KB

Raw Blame

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

```
import keras_nlp

imdb = tfds.load("imdb_reviews", as_supervised=True, data_dir='./data', download=False)

train_reviews = imdb['train'].map(lambda review, label: review)
train_labels = imdb['train'].map(lambda review, label: label)

keras_nlp.tokenizers.compute_word_piece_vocabulary(
    train_reviews,
    vocabulary_size=8000,
    reserved_tokens=["[PAD]", "[UNK]"],
    vocabulary_output_file='imdb_vocab_subwords.txt'
)

subword_tokenizer = keras_nlp.tokenizers.WordPieceTokenizer(
    vocabulary='./imdb_vocab_subwords.txt'
)
```

```
import keras_nlp

imdb = tfds.load("imdb_reviews", as_supervised=True, data_dir='./data', download=False)

train_reviews = imdb['train'].map(lambda review, label: review)
train_labels = imdb['train'].map(lambda review, label: label)

keras_nlp.tokenizers.compute_word_piece_vocabulary(
    train_reviews,
    vocabulary_size=8000,
    reserved_tokens=["[PAD]", "[UNK]"],
    vocabulary_output_file='imdb_vocab_subwords.txt'
)

subword_tokenizer = keras_nlp.tokenizers.WordPieceTokenizer(
    vocabulary='./imdb_vocab_subwords.txt'
)
```

```
import keras_nlp

imdb = tfds.load("imdb_reviews", as_supervised=True, data_dir='./data', download=False)

train_reviews = imdb['train'].map(lambda review, label: review)
train_labels = imdb['train'].map(lambda review, label: label)

keras_nlp.tokenizers.compute_word_piece_vocabulary(
    train_reviews,
    vocabulary_size=8000,
    reserved_tokens=["[PAD]", "[UNK]"],
    vocabulary_output_file='imdb_vocab_subwords.txt'
)

subword_tokenizer = keras_nlp.tokenizers.WordPieceTokenizer(
    vocabulary='./imdb_vocab_subwords.txt'
)
```

```
import keras_nlp

imdb = tfds.load("imdb_reviews", as_supervised=True, data_dir='./data', download=False)

train_reviews = imdb['train'].map(lambda review, label: review)
train_labels = imdb['train'].map(lambda review, label: label)

keras_nlp.tokenizers.compute_word_piece_vocabulary(
    train_reviews,
    vocabulary_size=8000,
    reserved_tokens=["[PAD]", "[UNK]"],
    vocabulary_output_file='imdb_vocab_subwords.txt'
)

subword_tokenizer = keras_nlp.tokenizers.WordPieceTokenizer(
    vocabulary='./imdb_vocab_subwords.txt'
)
```

```
import keras_nlp

imdb = tfds.load("imdb_reviews", as_supervised=True, data_dir='./data', download=False)

train_reviews = imdb['train'].map(lambda review, label: review)
train_labels = imdb['train'].map(lambda review, label: label)

keras_nlp.tokenizers.compute_word_piece_vocabulary(
    train_reviews,
    vocabulary_size=8000,
    reserved_tokens=["[PAD]", "[UNK]"],
    vocabulary_output_file='imdb_vocab_subwords.txt'
)

subword_tokenizer = keras_nlp.tokenizers.WordPieceTokenizer(
    vocabulary='./imdb_vocab_subwords.txt'
)
```

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

tokenized_string = subword_tokenizer.tokenize(sample_string)
print('Tokenized string is {}'.format(tokenized_string))

original_string = subword_tokenizer.detokenize(tokenized_string).numpy().decode("utf-8")
print('The original string: {}'.format(original_string))
```

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

tokenized_string = subword_tokenizer.tokenize(sample_string)
print('Tokenized string is {}'.format(tokenized_string))

original_string = subword_tokenizer.detokenize(tokenized_string).numpy().decode("utf-8")
print('The original string: {}'.format(original_string))
```

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

tokenized_string = subword_tokenizer.tokenize(sample_string)
print('Tokenized string is {}'.format(tokenized_string))

original_string = subword_tokenizer.detokenize(tokenized_string).numpy().decode("utf-8")
print('The original string: {}'.format(original_string))
```

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

tokenized_string = subword_tokenizer.tokenize(sample_string)
print('Tokenized string is {}'.format(tokenized_string))

original_string = subword_tokenizer.detokenize(tokenized_string).numpy().decode("utf-8")
print('The original string: {}'.format(original_string))
```

```
Tokenized string is [  53 2235  543 1827 3024   13  198 1659  174  167 2220  238]
The original string: TensorFlow , from basics to mastery
```

```
for i in range(len(tokenized_string)):
    subword = subword_tokenizer.detokenize(tokenized_string[i:i+1]).numpy().decode("utf-8")
    print(subword)
```

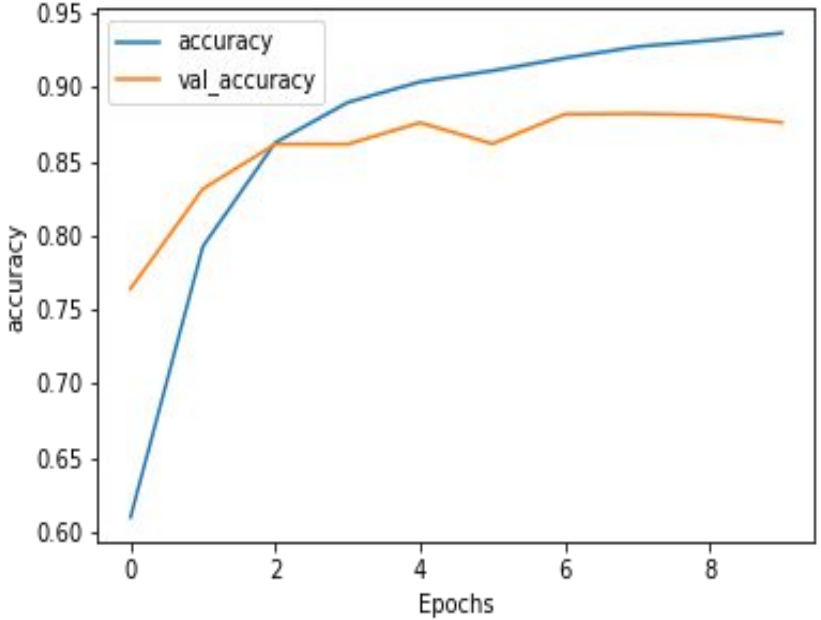
```
for i in range(len(tokenized_string)):
    subword = subword_tokenizer.detokenize(tokenized_string[i:i+1]).numpy().decode("utf-8")
    print(subword)
```

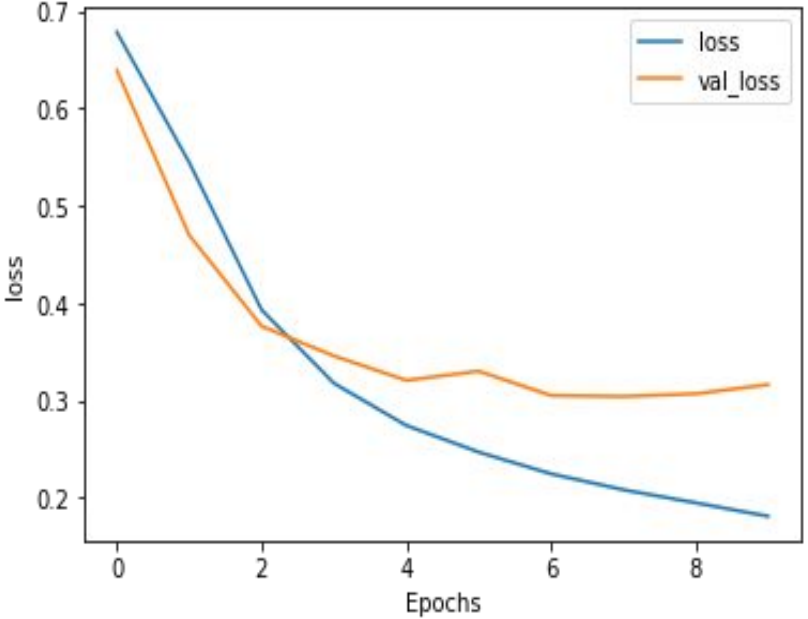
```
T
##ens
##or
##F
##low
,
from
basic
##s
to
master
##y
```

```
embedding_dim = 64
model = tf.keras.Sequential([
  tf.keras.Input(shape=(MAX_LENGTH,)),
  tf.keras.layers.Embedding(subword_tokenizer.vocabulary_size(), EMBEDDING_DIM),
  tf.keras.layers.GlobalAveragePooling1D(),
  tf.keras.layers.Dense(6, activation='relu'),
  tf.keras.layers.Dense(1, activation='sigmoid')
])

model.summary()
```

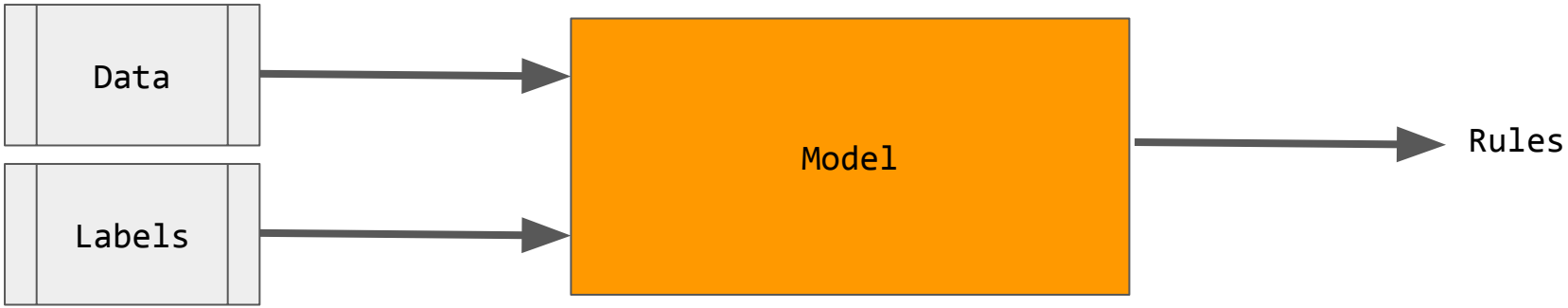
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 120, 64)           488,460
_________________________________________________________________
global_average_pooling1d_1 ( (None, 64)                0
_________________________________________________________________
dense_4 (Dense)              (None, 6)                 390
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 7
=================================================================
Total params: 489,037
Trainable params: 489,037
Non-trainable params: 0
```

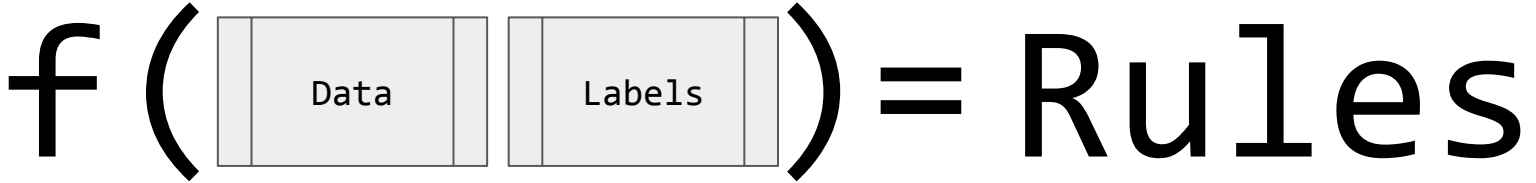
```
num_epochs = 10

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

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

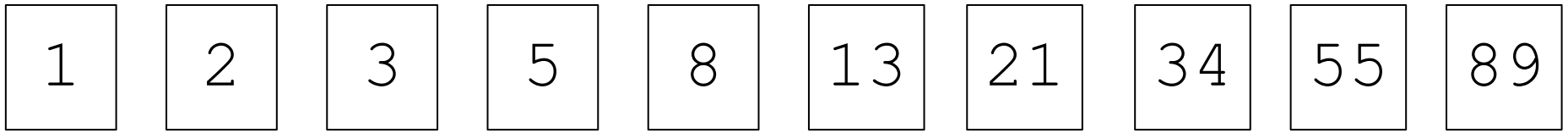

accuracy
val_accuracy
accuracy
Epochs
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0
2
4
6
8

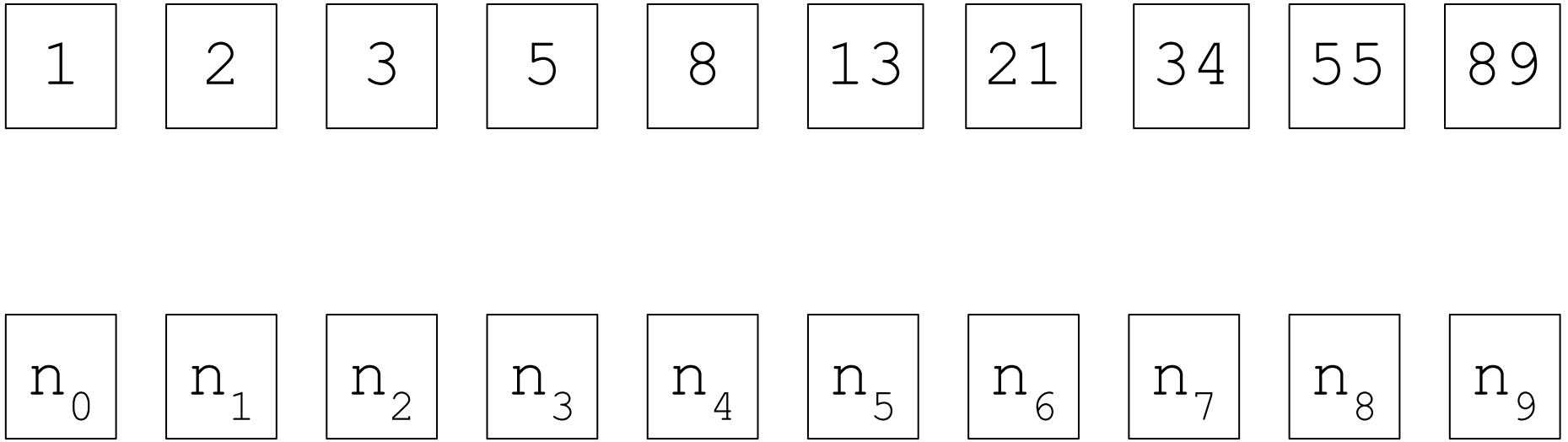

loss
val_loss
loss
Epochs
0.7
0.6
0.5
0.4
0.3
0.2
0
2
4
6
8

Copyright Notice


Data
Labels
Model
Rules

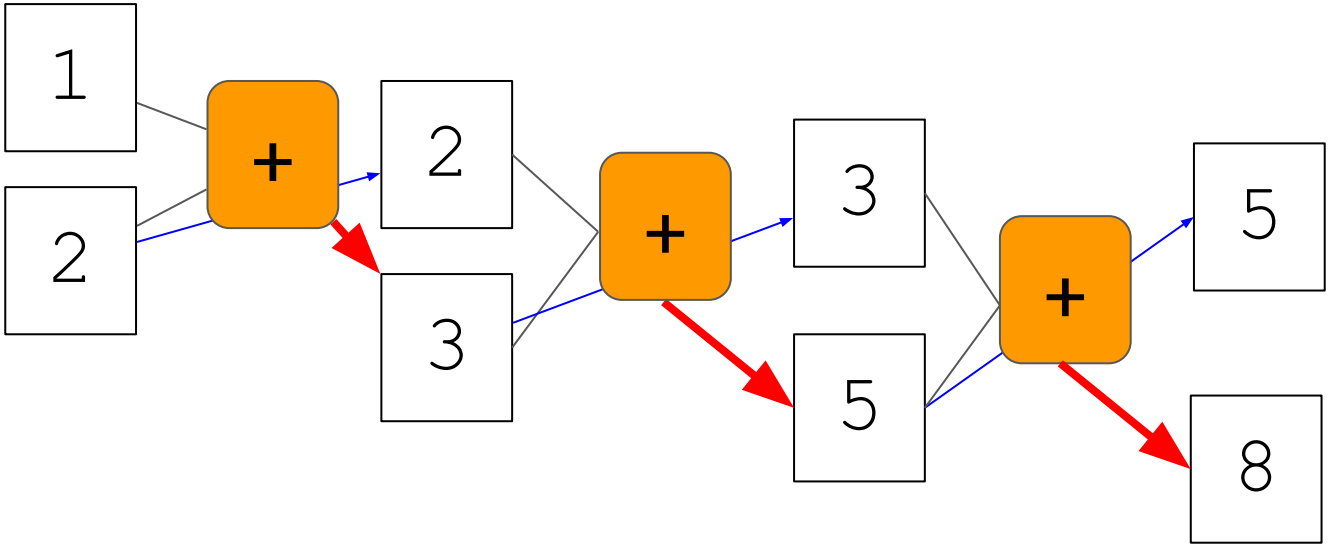

f(
Data
Labels
)=Rules

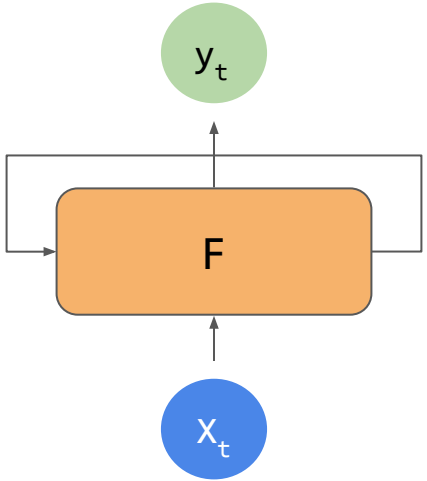

1
2
3
5
8
13
21
34
55
89

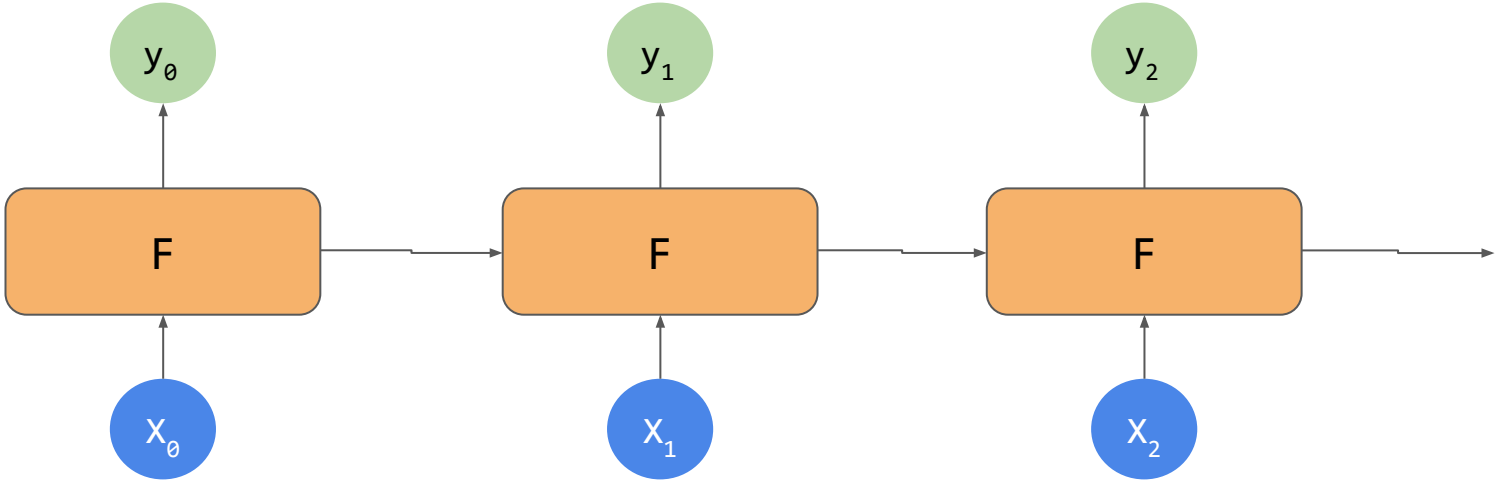

1 2 3 5 8 13 21 34 55 89
n_0 n_1 n_2 n_3 n_4 n_5 n_6 n_7 n_8 n_9

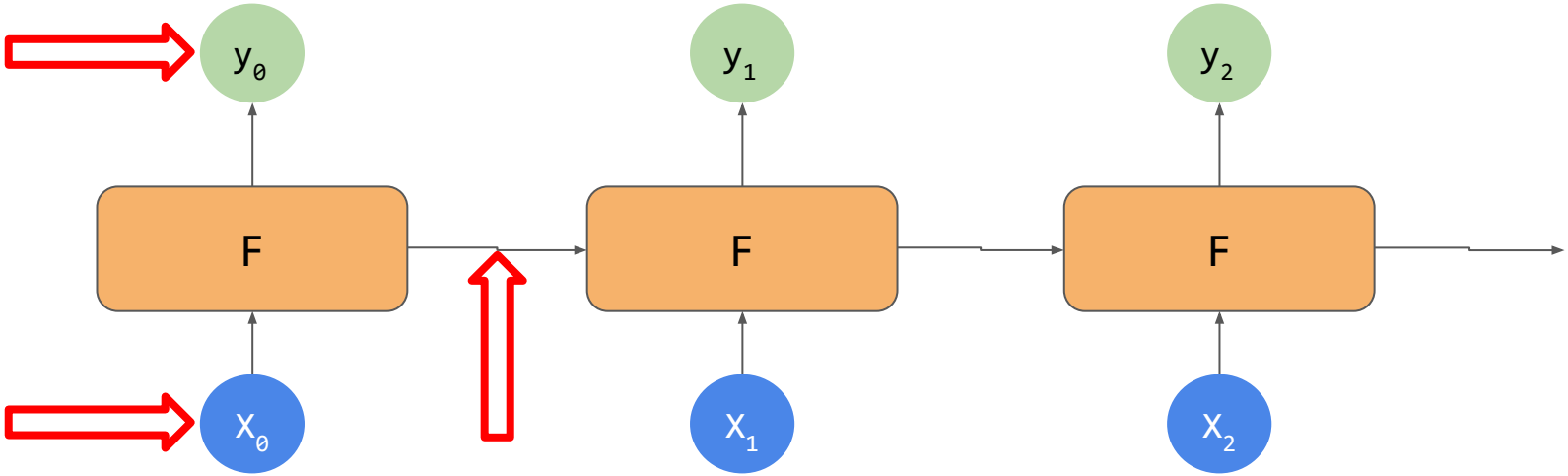
1	2	3	5	8	13	21	34	55	89
n_0	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9

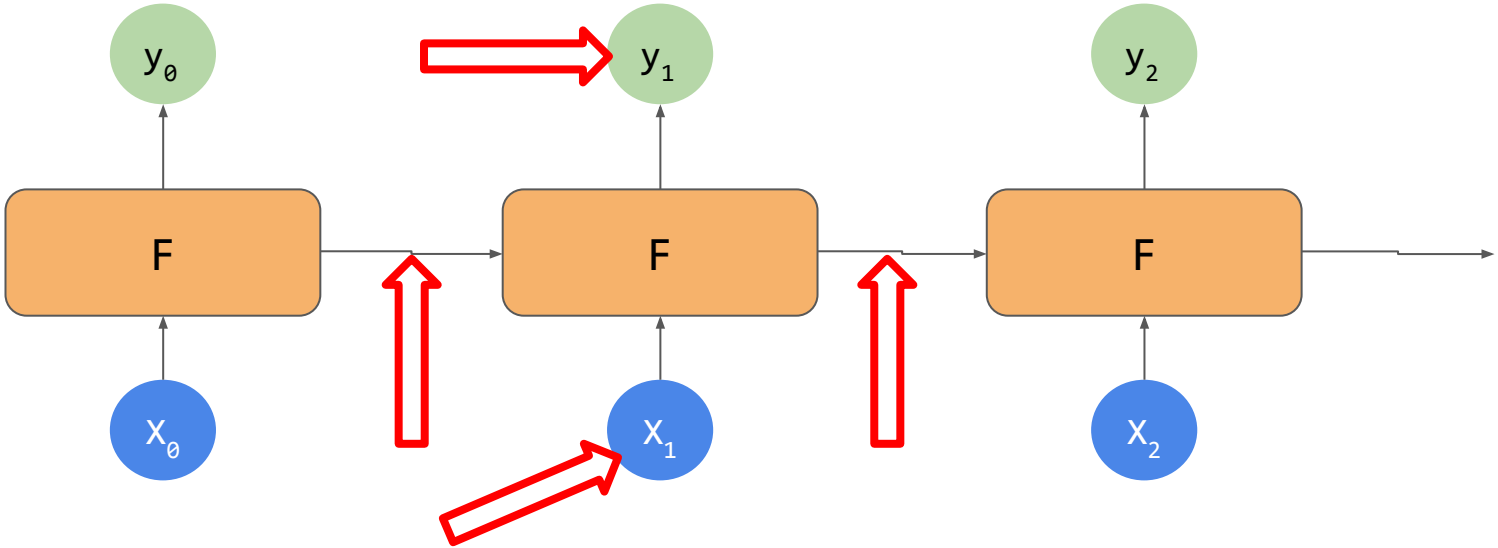
$$n_x = n_{x-1} + n_{x-2}$$

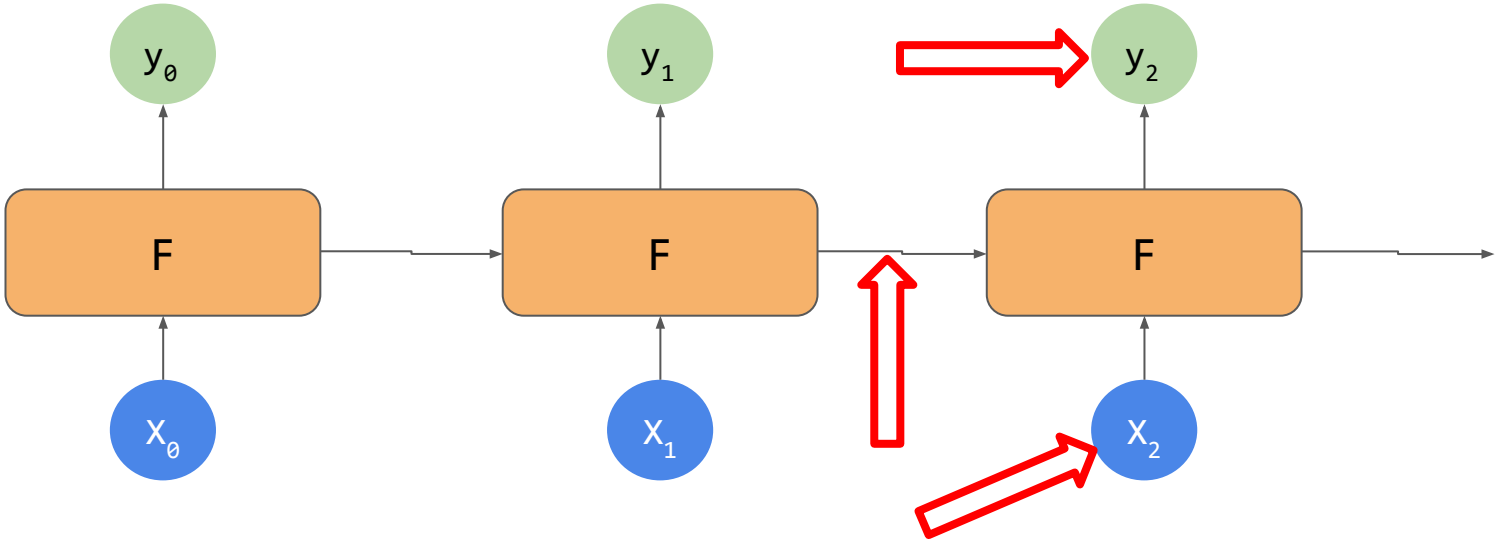

1
2
+
2
3
+
3
5
+
5
8

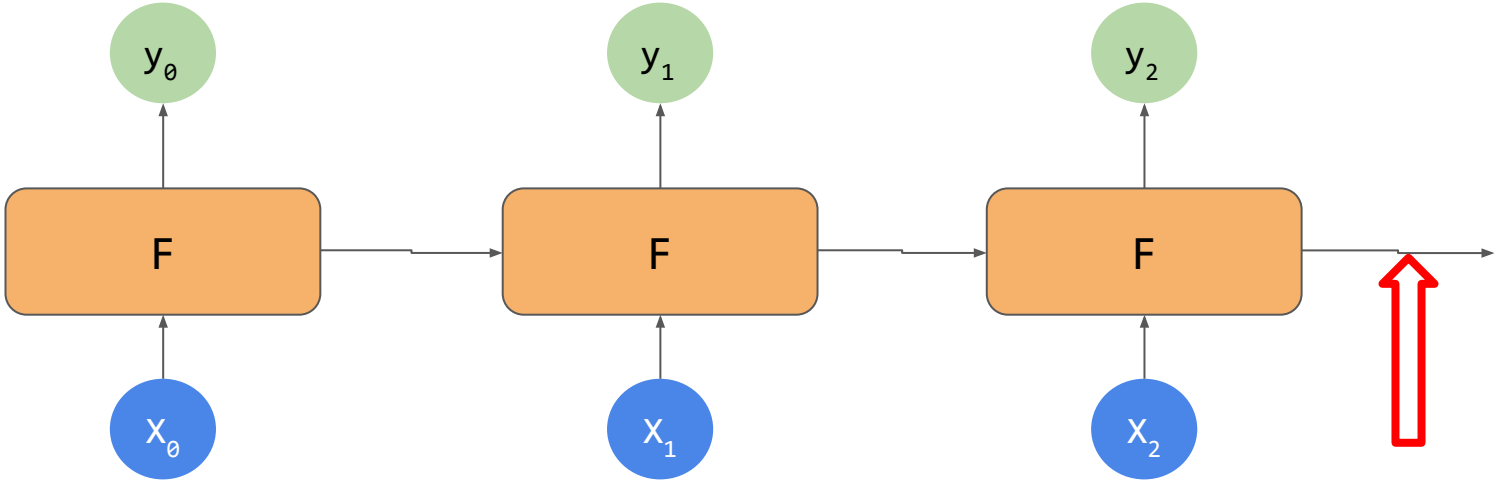

y_t
F
x_t


y0
y1
y2
F
F
F
x0
x1
x2


y0
y1
y2
F
F
F
x0
x1
x2

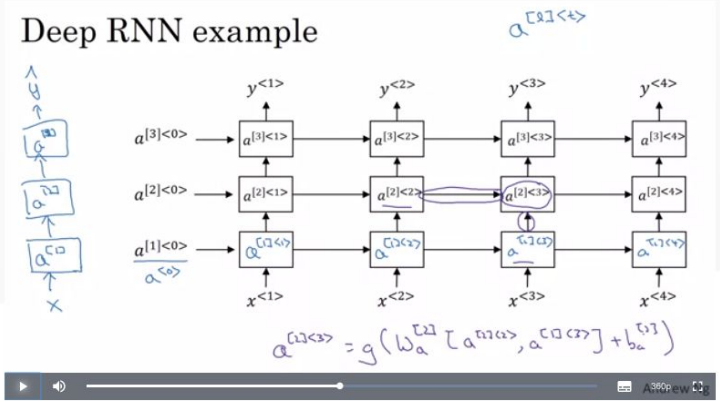

y0
y1
y2
F
F
F
x0
x1
x2


y0
y1
y2
F
F
F
x0
x1
x2


y0
y1
y2
F
F
F
x0
x1
x2

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

Deep RNNs



From the course by deeplearning.ai

Sequence Models

13393 ratings

Try the Course for Free

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

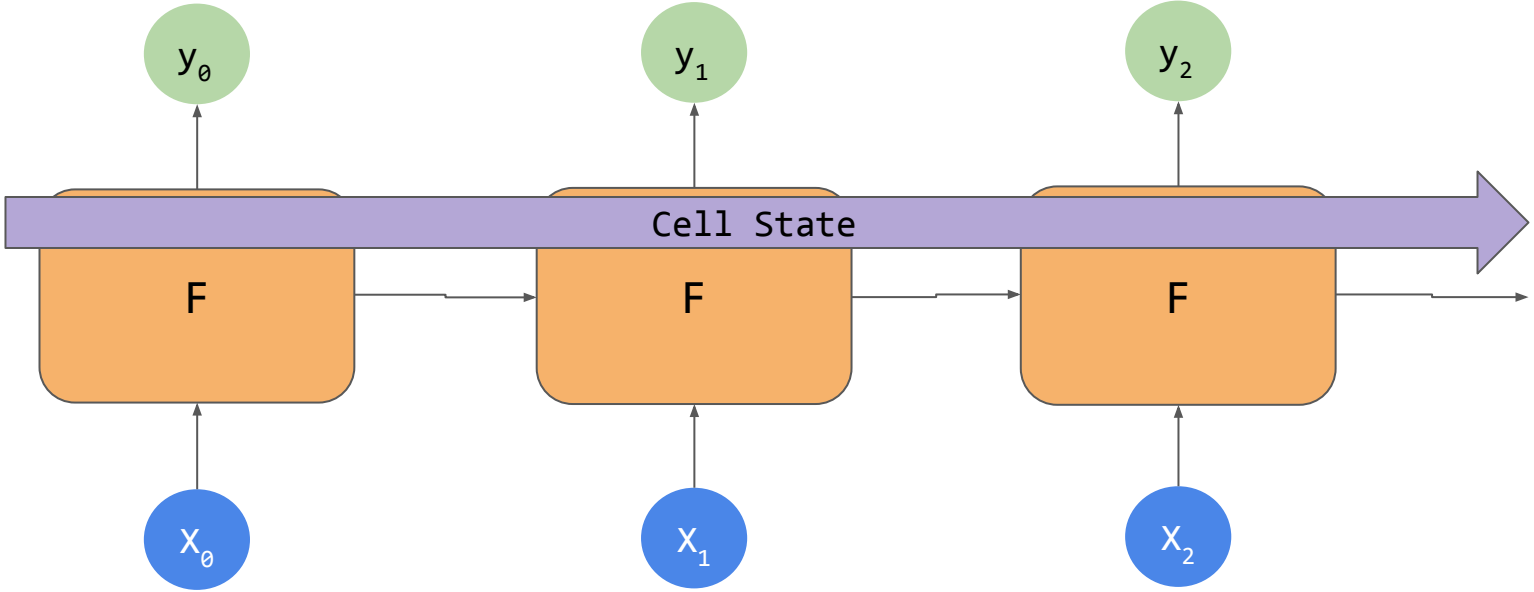
I lived in Ireland, so at school they made me learn how to speak <...>

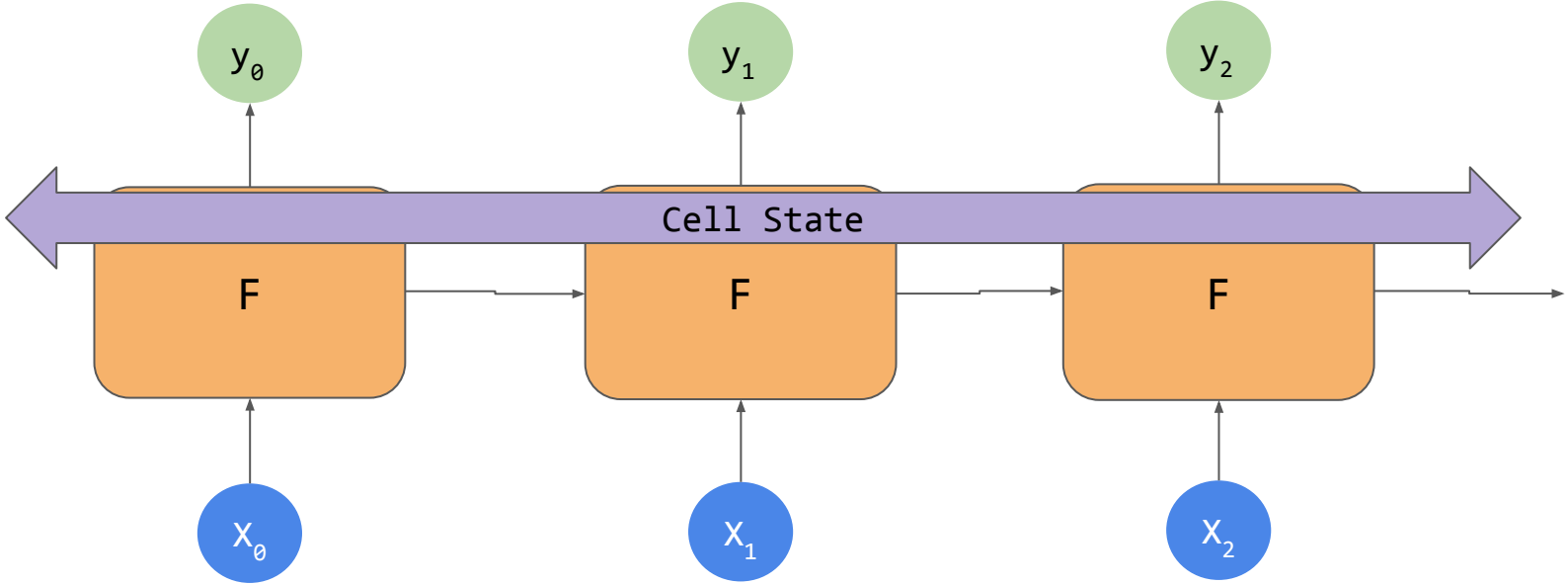
I lived in Ireland, so at school they made me learn how to speak <...>

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 <...>

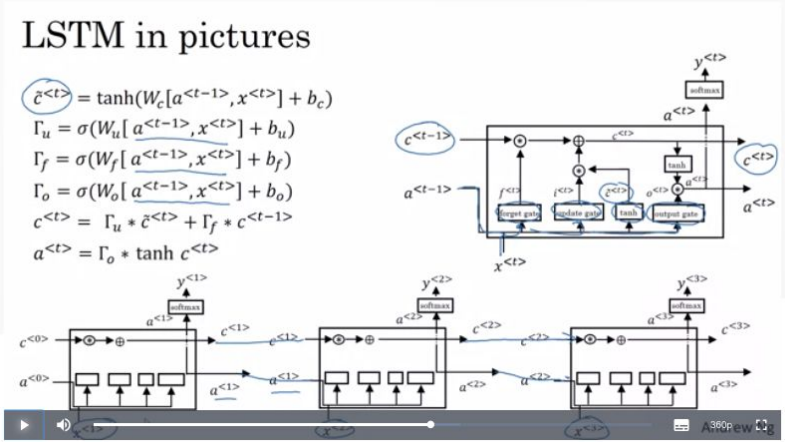
I lived in Ireland, so at school they made me learn how to speak Gaelic


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)



From the course by deeplearning.ai

Sequence Models

13393 ratings

Try the Course for Free

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

```
model = tf.keras.Sequential([
    tf.keras.Input(shape=(None,)),
    tf.keras.layers.Embedding(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.Input(shape=(None,)),
    tf.keras.layers.Embedding(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.Input(shape=(None,)),
    tf.keras.layers.Embedding(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
```

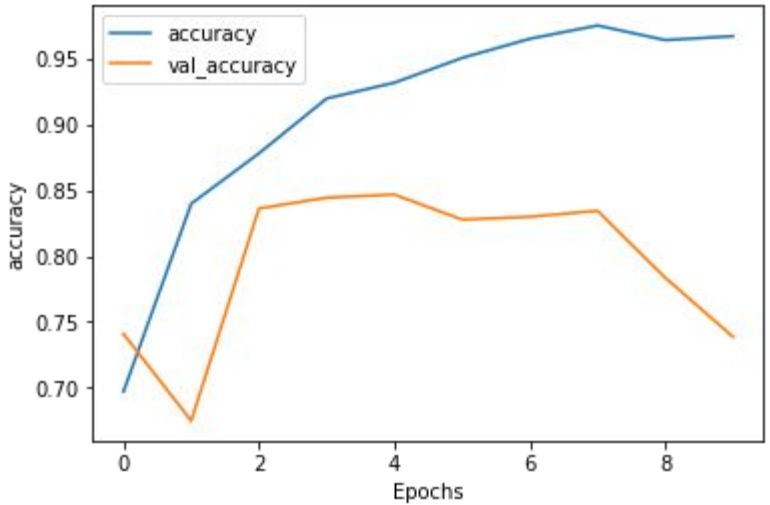
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, None, 64)          523840
_________________________________________________________________
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.Input(shape=(None,)),
    tf.keras.layers.Embedding(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')
])
```

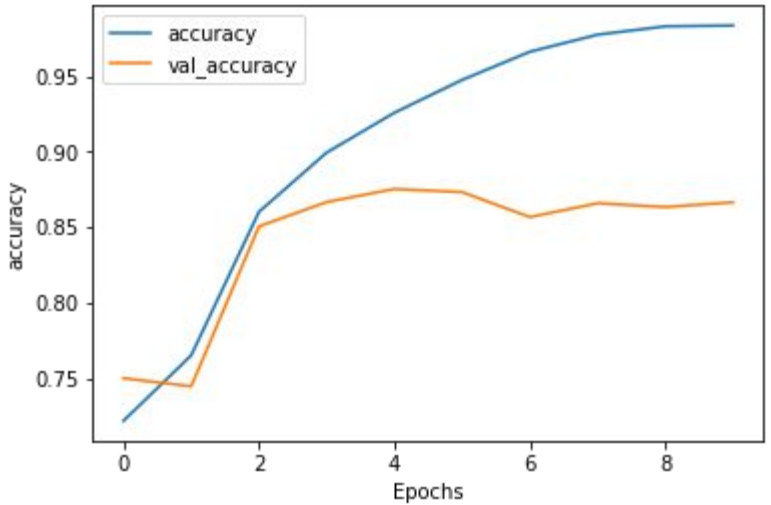
```
model = tf.keras.Sequential([
    tf.keras.Input(shape=(None,)),
    tf.keras.layers.Embedding(vocab_size, 64),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64, return_sequences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Dense(64, activation='relu'),
    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

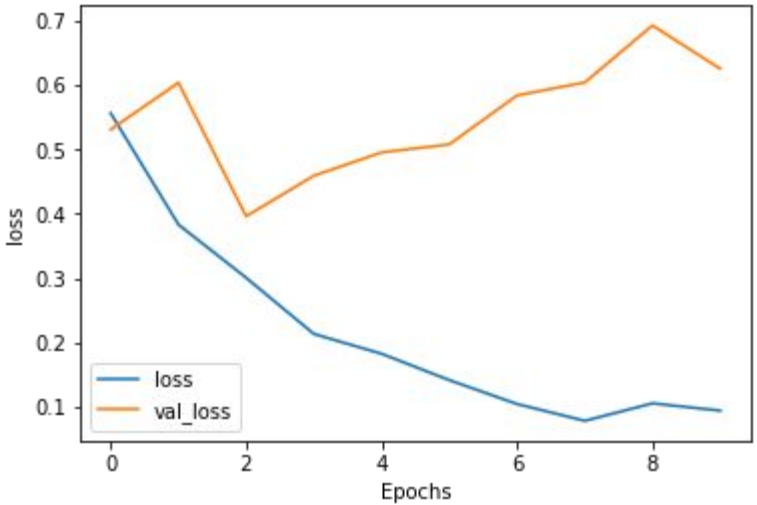


1 Layer LSTM

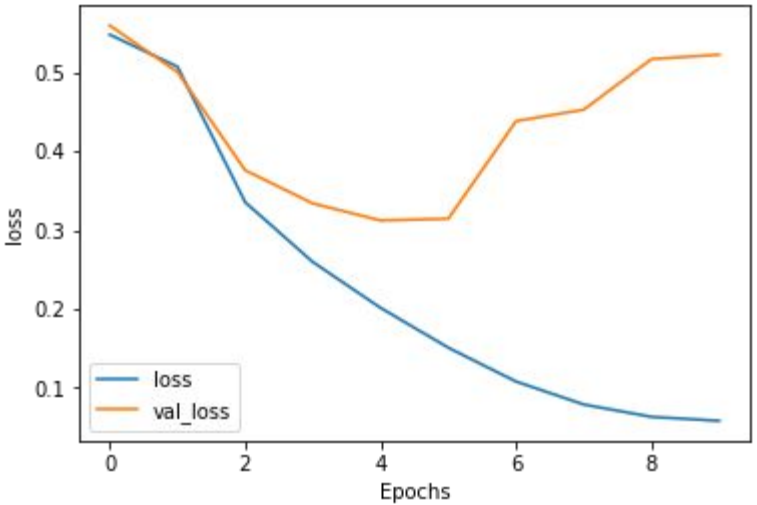


2 Layer LSTM

10 Epochs : Loss Measurement

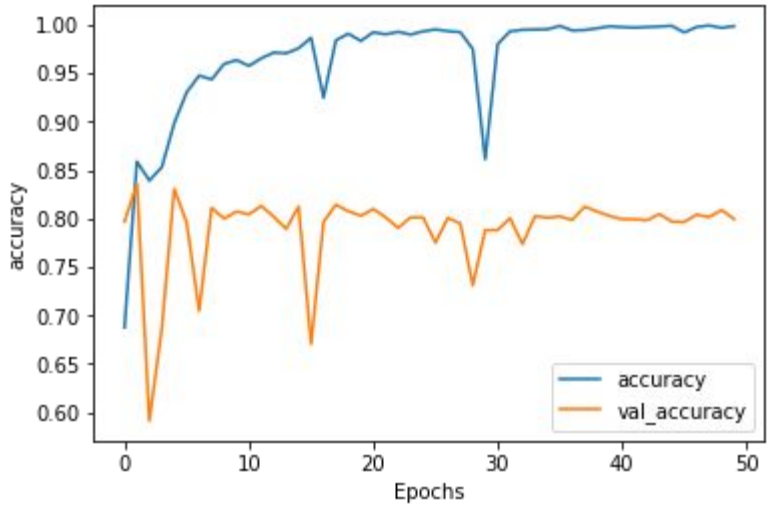


1 Layer LSTM

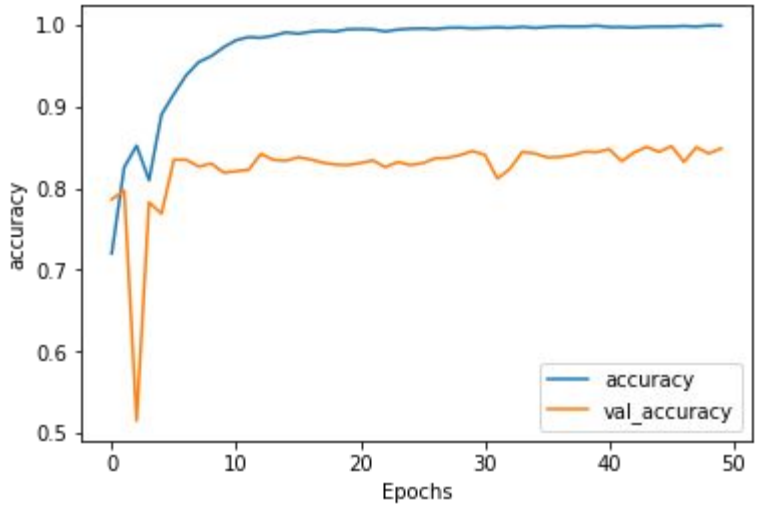


2 Layer LSTM

50 Epochs : Accuracy Measurement

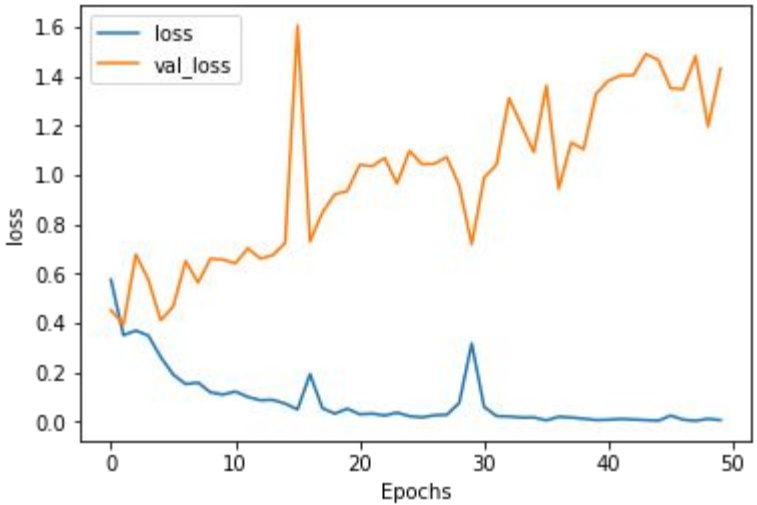


1 Layer LSTM

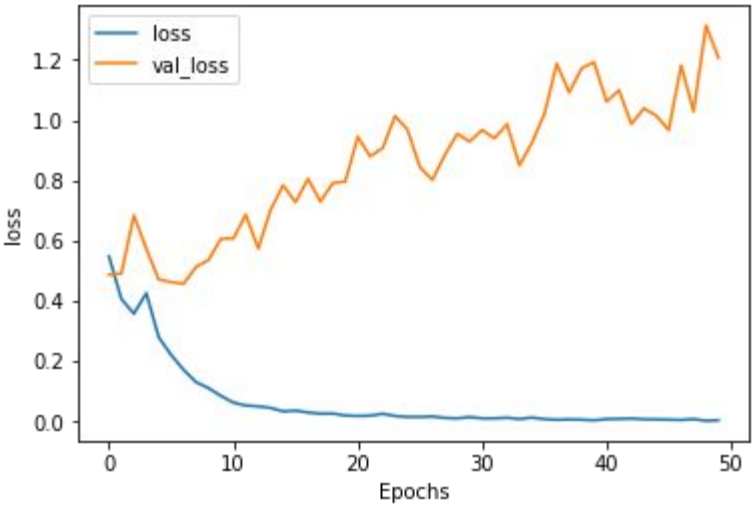


2 Layer LSTM

50 Epochs : Loss Measurement



1 Layer LSTM

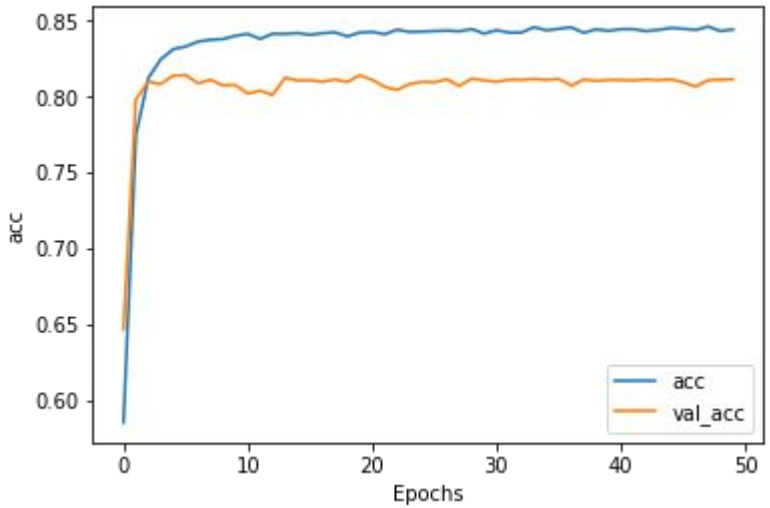


2 Layer LSTM

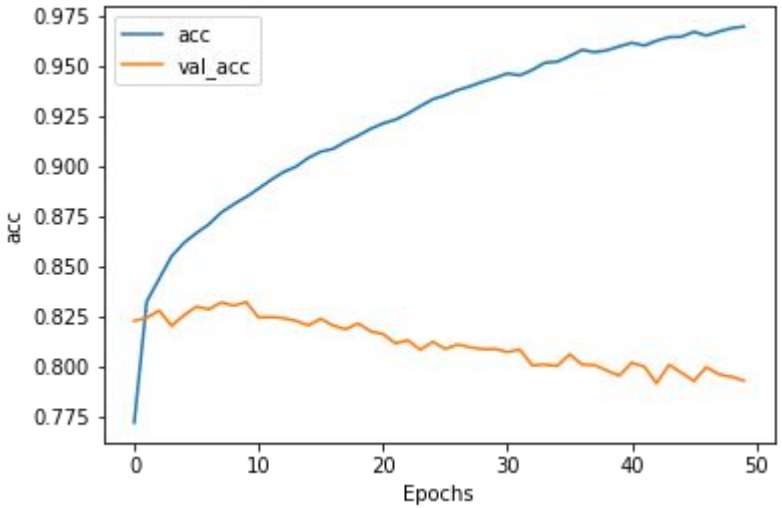
```
model = tf.keras.Sequential([
    tf.keras.Input(shape=(None,)),
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(24, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
model = tf.keras.Sequential([
    tf.keras.Input(shape=(None,)),
    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')
])
```

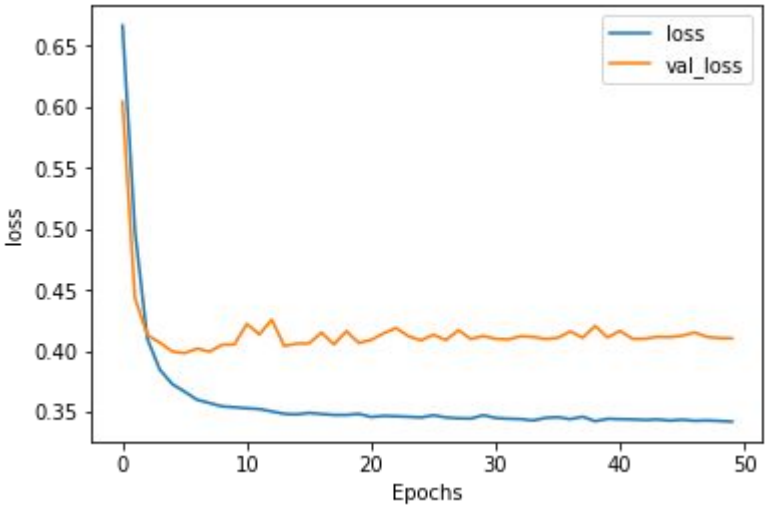
```
model = tf.keras.Sequential([
    tf.keras.Input(shape=(None,)),
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Dense(24, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```



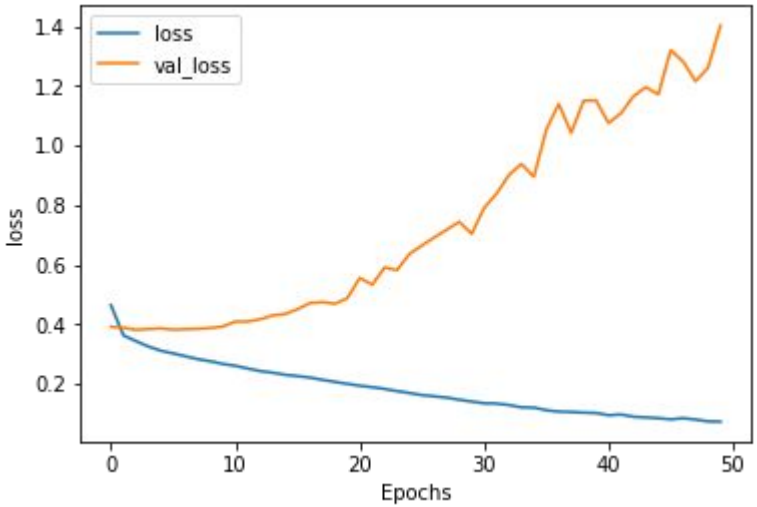
Without LSTM



With LSTM



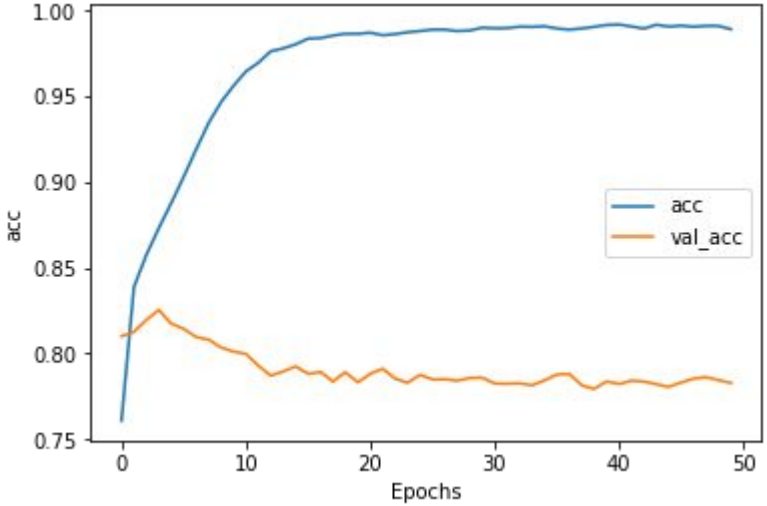
Without LSTM

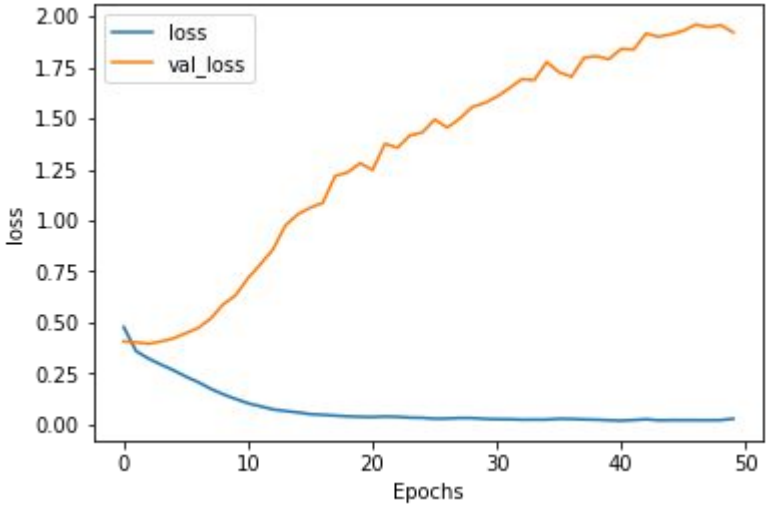


With LSTM

```
model = tf.keras.Sequential([
    tf.keras.Input(shape=(None,)),
    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(6, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
model = tf.keras.Sequential([
    tf.keras.Input(shape=(None,)),
    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(6, 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.Input(shape=(None,)),
    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(6, 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'),
```

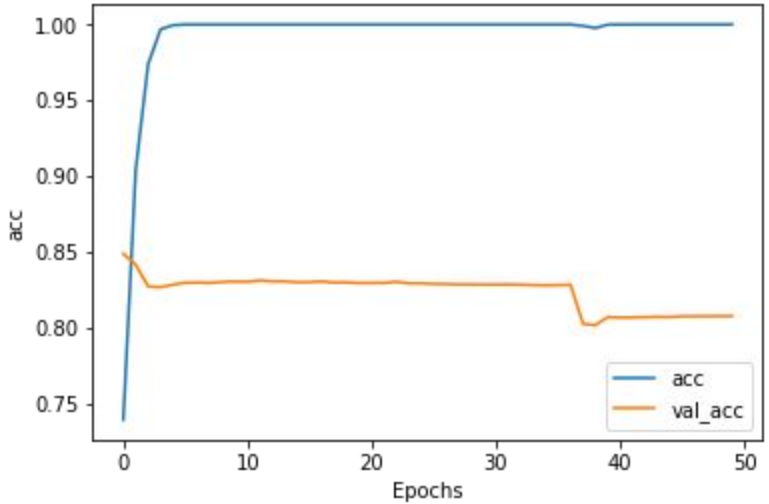
```
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 120, 16)           16000
_________________________________________________________________
conv1d (Conv1D)              (None, 116, 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
```

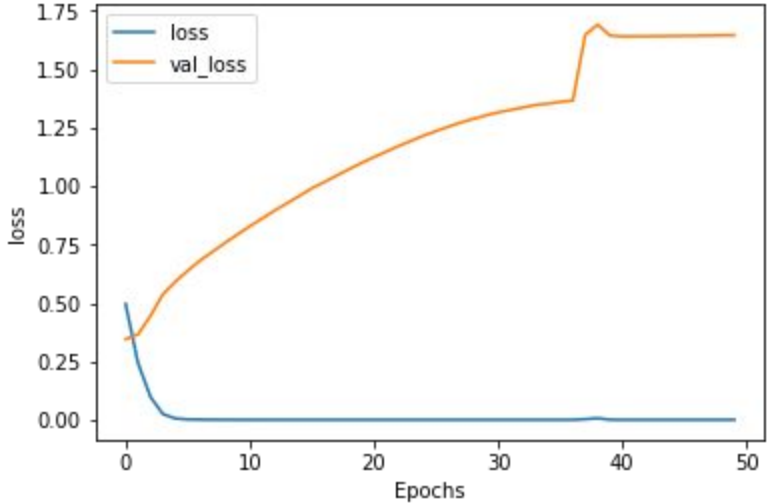
```
max_length = 120

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.Input(shape=(None,)),
    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
```

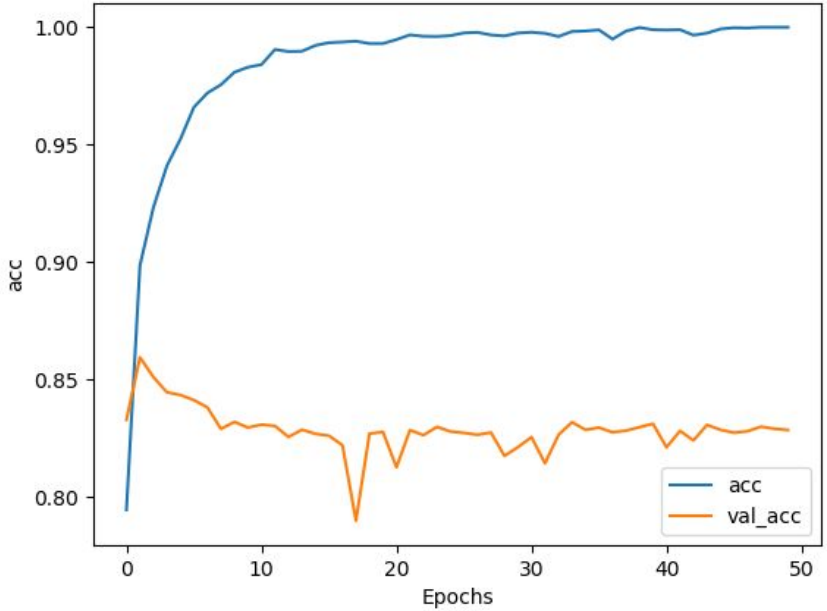


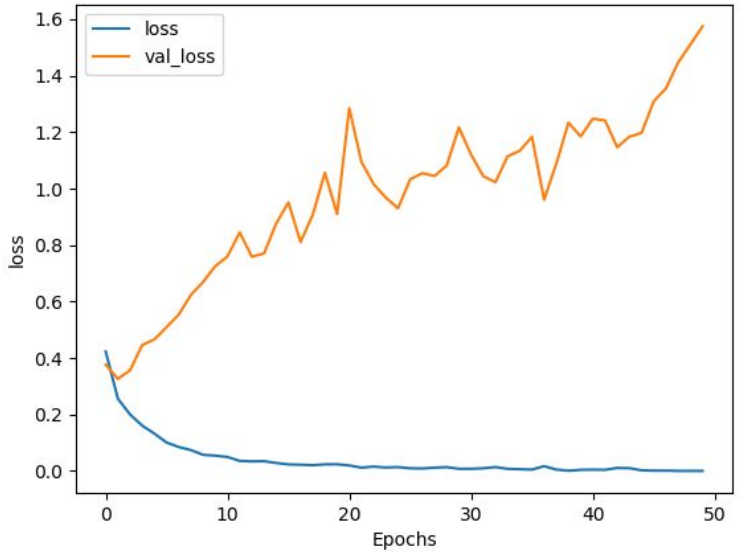


IMDB with Embedding-only : ~ 5s per epoch

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


model = tf.keras.Sequential([
    tf.keras.Input(shape=(None,)),
    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
```

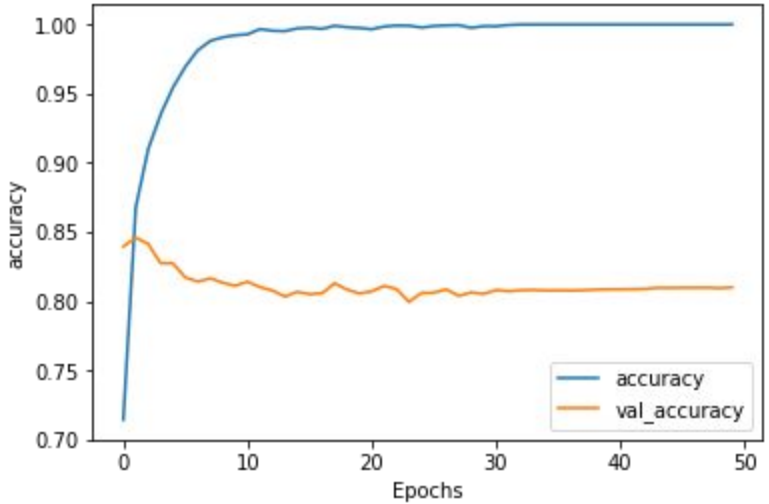


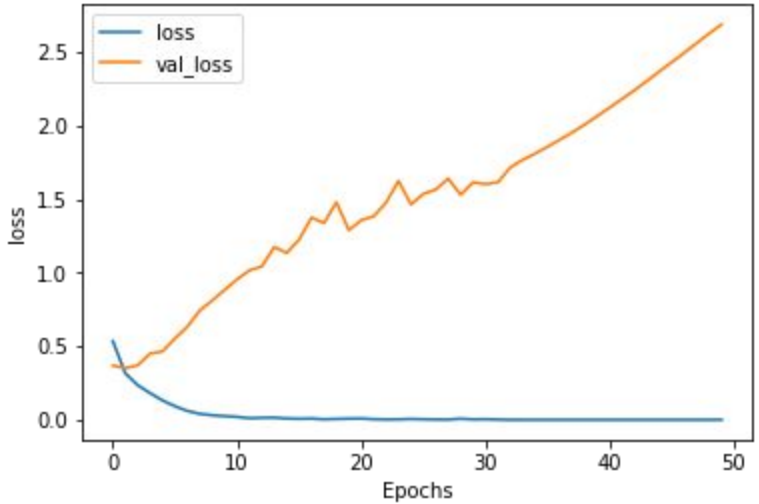


IMDB with LSTM ~43s per epoch

```
model = tf.keras.Sequential([
    tf.keras.Input(shape=(None,)),
    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
```

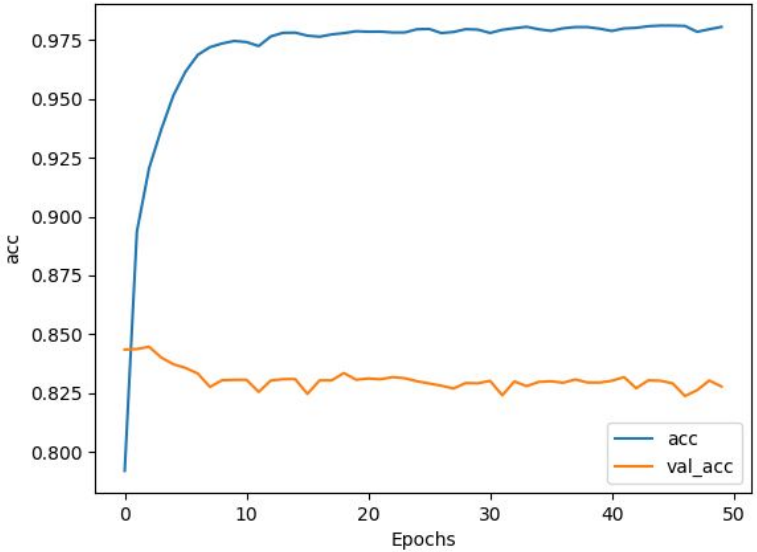


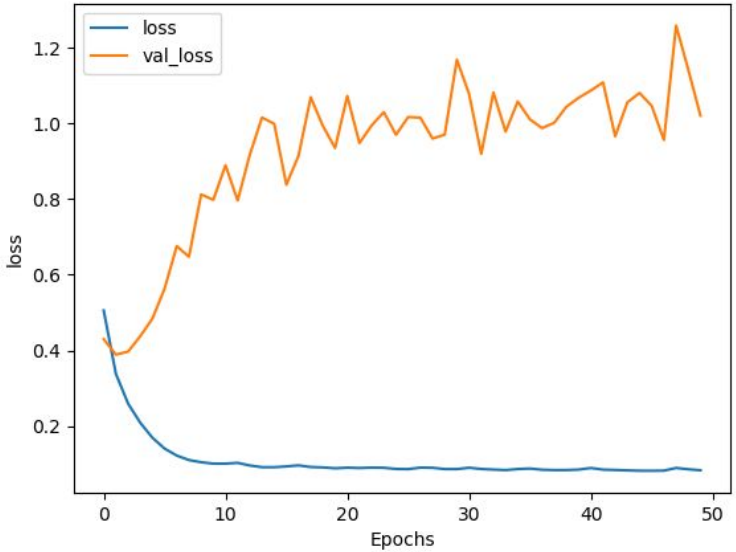


IMDB with GRU : ~ 20s per epoch

```
model = tf.keras.Sequential([
    tf.keras.Input(shape=(None,)),
    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.

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

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

vocabulary = vectorize_layer.get_vocabulary()
vocab_size = len(vocabulary)
```

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

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

vocabulary = vectorize_layer.get_vocabulary()
vocab_size = len(vocabulary)
```

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

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

vocabulary = vectorize_layer.get_vocabulary()
vocab_size = len(vocabulary)
```

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

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

vocabulary = vectorize_layer.get_vocabulary()
vocab_size = len(vocabulary)
```

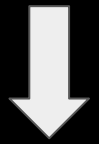
```
data = "In the town of Athy one Jeremy Lanigan \n Battered away ... ..."
corpus = data.lower().split("\n")

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

vocabulary = vectorize_layer.get_vocabulary()
vocab_size = len(vocabulary)
```

```
input_sequences = []
for line in corpus:
    sequence = vectorize_layer(line).numpy()
    for i in range(1, len(sequence)):
        n_gram_sequence = sequence[:i+1]
        input_sequences.append(n_gram_sequence)
```

```
input_sequences = []
for line in corpus:
    sequence = vectorize_layer(line).numpy()
    for i in range(1, len(sequence)):
        n_gram_sequence = sequence[:i+1]
        input_sequences.append(n_gram_sequence)
```

```
input_sequences = []
for line in corpus:
    sequence = vectorize_layer(line).numpy()
    for i in range(1, len(sequence)):
        n_gram_sequence = sequence[: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:
    sequence = vectorize_layer(line).numpy()
    for i in range(1, len(sequence))
        n_gram_sequence = sequence[: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(tf.keras.utils.pad_sequences(input_sequences,
                                                        maxlen=max_sequence_len,
                                                        padding='pre'))
```

Line:

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

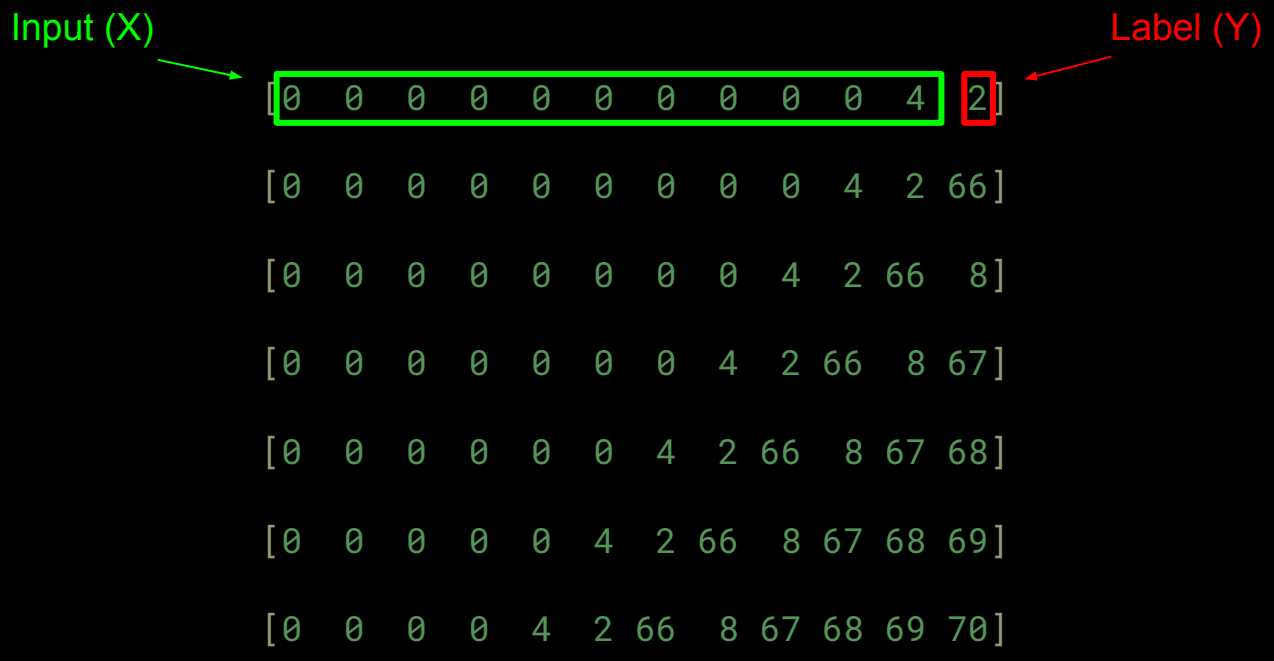
Padded Input Sequences:

```
[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]
```

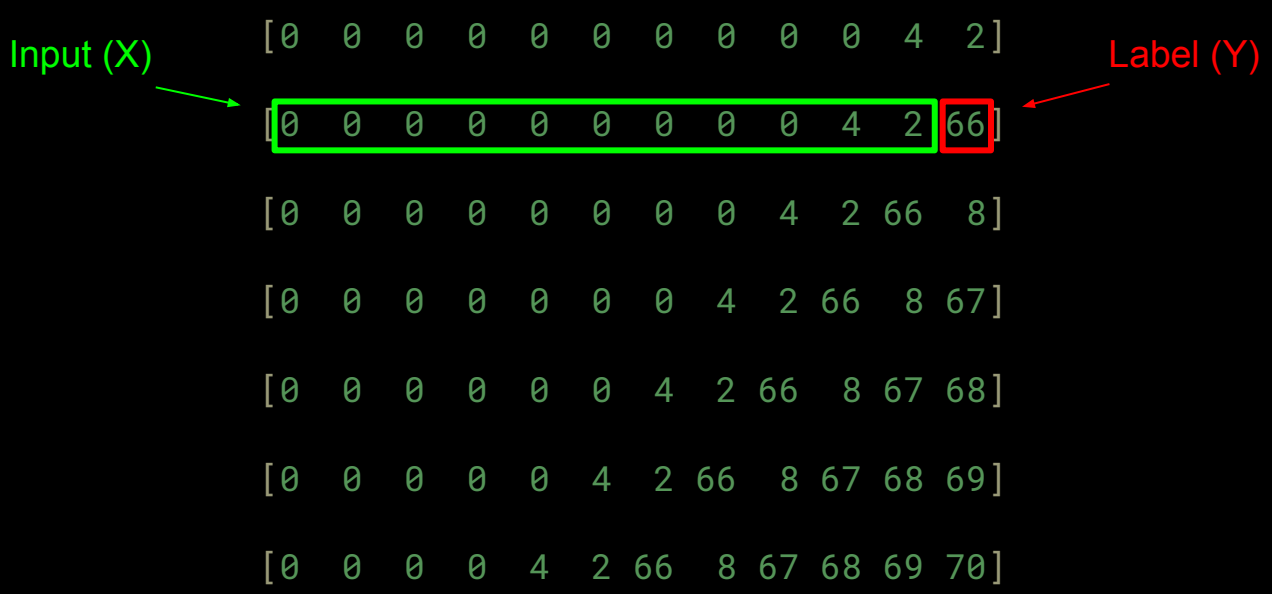
Padded Input Sequences:

```
[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]
```

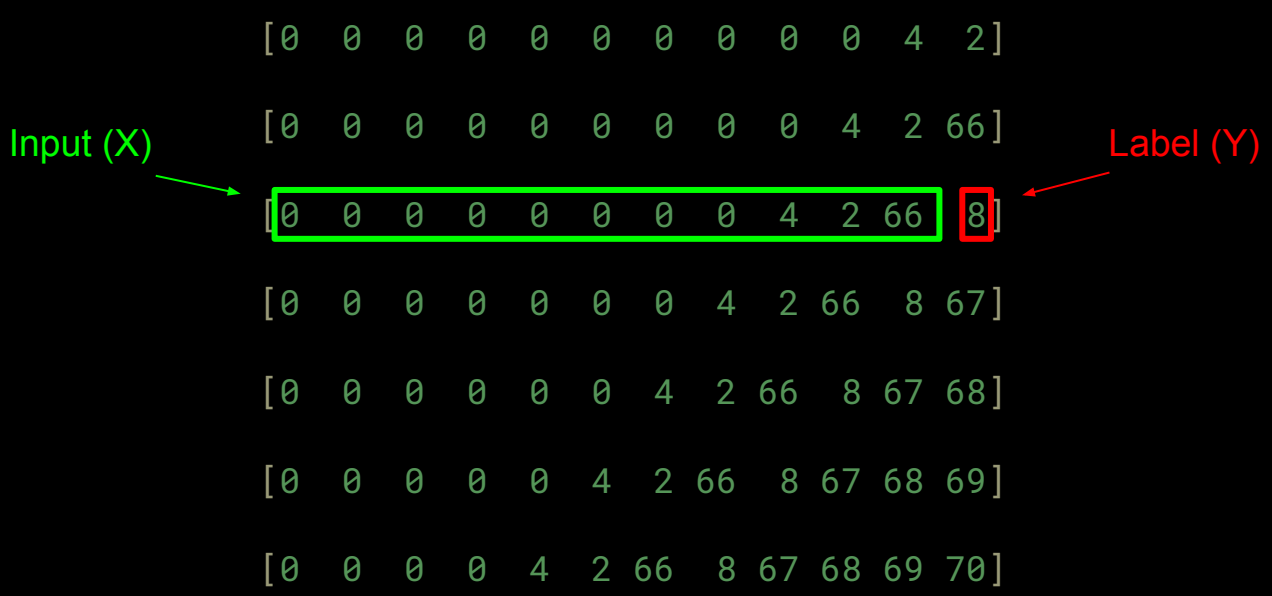
Padded Input Sequences:



Padded Input Sequences:



Padded Input Sequences:



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

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

```
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 = tf.keras.Sequential([
    tf.keras.Input(shape=(max_sequence_len-1,)),
    tf.keras.layers.Embedding(vocab_size, 64),
    tf.keras.layers.LSTM(20),
    tf.keras.layers.Dense(vocab_size, activation='softmax')
])

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

model.fit(xs, ys, epochs=500)
```

```
model = tf.keras.Sequential([
    tf.keras.Input(shape=(max_sequence_len-1,)),
    tf.keras.layers.Embedding(vocab_size, 64),
    tf.keras.layers.LSTM(20),
    tf.keras.layers.Dense(vocab_size, activation='softmax')
])

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

model.fit(xs, ys, epochs=500)
```

```
model = tf.keras.Sequential([
    tf.keras.Input(shape=(max_sequence_len-1,)),
    tf.keras.layers.Embedding(vocab_size, 64),
    tf.keras.layers.LSTM(20),
    tf.keras.layers.Dense(vocab_size, activation='softmax')
])

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

model.fit(xs, ys, epochs=500)
```

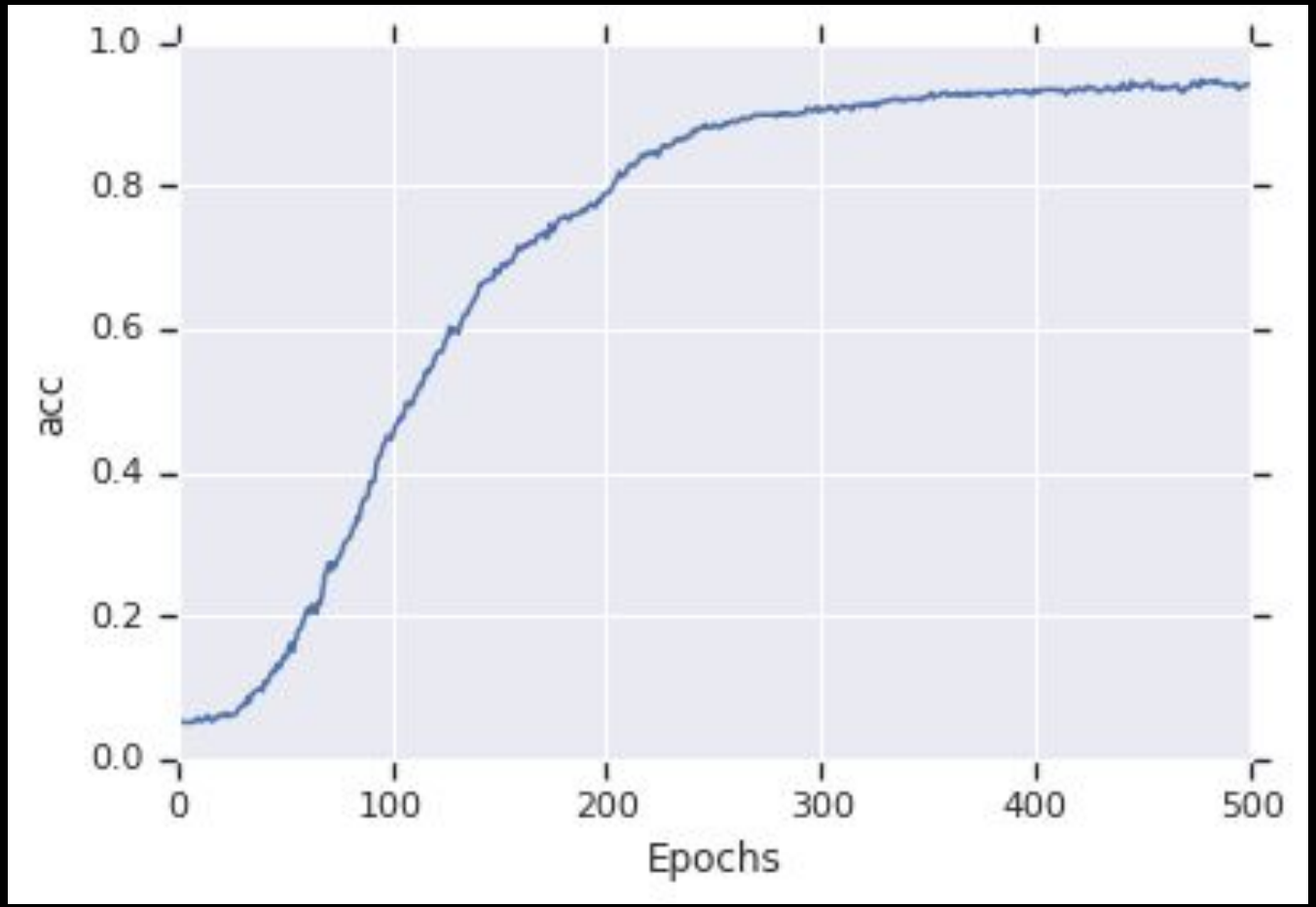
```
model = tf.keras.Sequential([
    tf.keras.Input(shape=(max_sequence_len-1,)),
    tf.keras.layers.Embedding(vocab_size, 64),
    tf.keras.layers.LSTM(20),
    tf.keras.layers.Dense(vocab_size, activation='softmax')
])

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

model.fit(xs, ys, epochs=500)
```

```
model = tf.keras.Sequential([
    tf.keras.Input(shape=(max_sequence_len-1,)),
    tf.keras.layers.Embedding(vocab_size, 64),
    tf.keras.layers.LSTM(20),
    tf.keras.layers.Dense(vocab_size, activation='softmax')
])

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

model.fit(xs, ys, epochs=500)
```

```
model = tf.keras.Sequential([
    tf.keras.Input(shape=(max_sequence_len-1,)),
    tf.keras.layers.Embedding(vocab_size, 64),
    tf.keras.layers.LSTM(20),
    tf.keras.layers.Dense(vocab_size, activation='softmax')
])

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

model.fit(xs, ys, epochs=500)
```

```
model = tf.keras.Sequential([
    tf.keras.Input(shape=(max_sequence_len-1,)),
    tf.keras.layers.Embedding(vocab_size, 64),
    tf.keras.layers.LSTM(20),
    tf.keras.layers.Dense(vocab_size, activation='softmax')
])

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

model.fit(xs, ys, epochs=500)
```

```
model = tf.keras.Sequential([
    tf.keras.Input(shape=(max_sequence_len-1,)),
    tf.keras.layers.Embedding(vocab_size, 64),
    tf.keras.layers.LSTM(20),
    tf.keras.layers.Dense(vocab_size, activation='softmax')
])

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

model.fit(xs, ys, epochs=500)
```


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

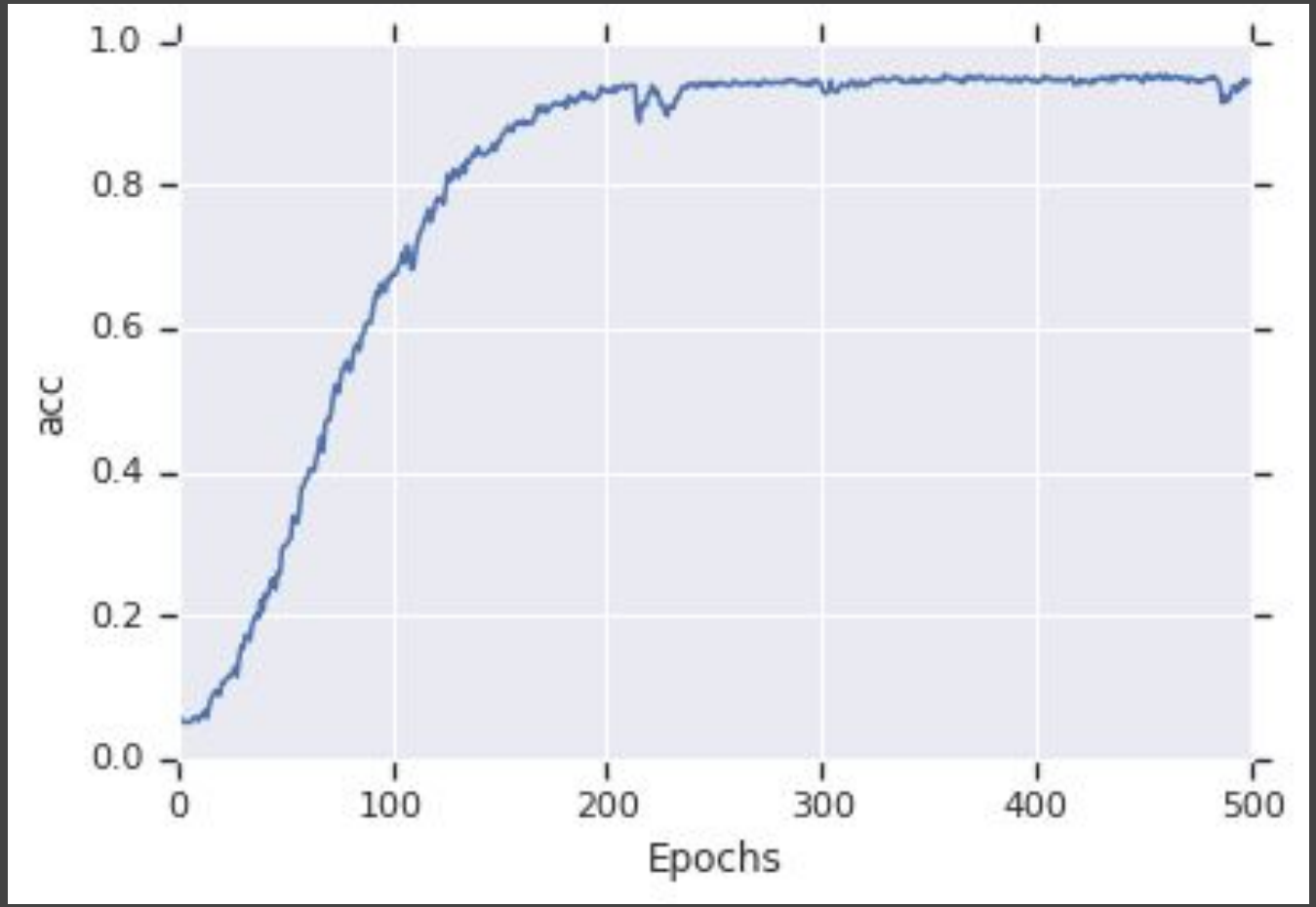
Laurence went to dublin round the plenty as red wall me for wall wall
Laurence went to dublin odaly of the nice of lanigans ball ball ball hall
Laurence went to dublin he hadnt a minute both relations hall new relations youd

```
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 = tf.keras.Sequential([
    tf.keras.Input(shape=(max_sequence_len-1,)),
    tf.keras.layers.Embedding(vocab_size, 64),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(20)),
    tf.keras.layers.Dense(vocab_size, activation='softmax')
])

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

model.fit(xs, ys, epochs=500)
```

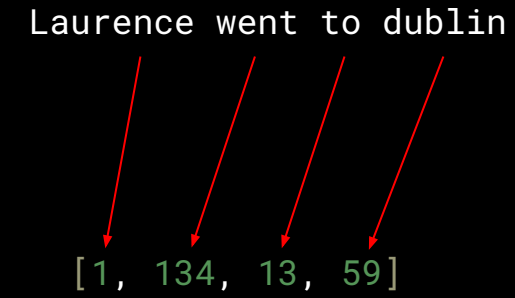
```
model = tf.keras.Sequential([
    tf.keras.Input(shape=(max_sequence_len-1,)),
    tf.keras.layers.Embedding(vocab_size, 64),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(20)),
    tf.keras.layers.Dense(vocab_size, activation='softmax')
])

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

model.fit(xs, ys, epochs=500)
```


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

```
seed_text = "Laurence went to Dublin"
```

```
sequence = vectorize_layer(seed_text)
```


Laurence went to dublin
[1, 134, 13, 59]

```
sequence = tf.keras.utils.pad_sequences(
    [sequence],
    maxlen=max_sequence_len-1,
    padding='pre')
```

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

```
probabilities = model.predict(sequence, verbose=0)

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

```
output_word = vocabulary[predicted]

seed_text += " " + output_word
```

```
seed_text = "Laurence went to Dublin"

next_words = 100

for _ in range(next_words):
    sequence = vectorize_layer(seed_text)
    sequence = tf.keras.utils.pad_sequences(
        [sequence],
        maxlen=max_sequence_len-1,
        padding='pre')
    probabilities = model.predict(sequence, verbose=0)
    predicted = np.argmax(probabilities, axis=-1)[0]
    output_word = vocabulary[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 glisten glisten glisten glisten
glisten glisten glisten glisten glisten glisten glisten
glisten glisten glisten glisten glisten glisten glisten
glisten glisten glisten glisten glisten glisten glisten
glisten glisten glisten glisten glisten glisten glisten
glisten glisten glisten glisten glisten glisten glisten
glisten glisten glisten glisten glisten glisten glisten
glisten glisten glisten glisten glisten glisten glisten
glisten glisten glisten glisten glisten glisten glisten
glisten glisten glisten glisten glisten glisten glisten
glisten glisten glisten glisten glisten glisten glisten
glisten glisten glisten glisten glisten glisten glisten

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

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

```
model = tf.keras.Sequential([
    tf.keras.Input(shape=(max_sequence_len-1,)),
    tf.keras.layers.Embedding(vocab_size, 100),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(150)),
    tf.keras.layers.Dense(vocab_size, activation='softmax')
])

adam = tf.keras.optimizers.Adam(learning_rate=0.01)
model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])

model.fit(xs, ys, epochs=100)
```

```
model = tf.keras.Sequential([
    tf.keras.Input(shape=(max_sequence_len-1,)),
    tf.keras.layers.Embedding(vocab_size, 100),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(150)),
    tf.keras.layers.Dense(vocab_size, activation='softmax')
])

adam = tf.keras.optimizers.Adam(learning_rate=0.01)
model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])

model.fit(xs, ys, epochs=100)
```

```
model = tf.keras.Sequential([
    tf.keras.Input(shape=(max_sequence_len-1,)),
    tf.keras.layers.Embedding(vocab_size, 100),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(150)),
    tf.keras.layers.Dense(vocab_size, activation='softmax')
])

adam = tf.keras.optimizers.Adam(learning_rate=0.01)
model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])

model.fit(xs, ys, epochs=100)
```

```
model = tf.keras.Sequential([
    tf.keras.Input(shape=(max_sequence_len-1,)),
    tf.keras.layers.Embedding(vocab_size, 100),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(150)),
    tf.keras.layers.Dense(vocab_size, activation='softmax')
])

adam = tf.keras.optimizers.Adam(learning_rate=0.01)
model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])

model.fit(xs, ys, epochs=100)
```

```
model = tf.keras.Sequential([
    tf.keras.Input(shape=(max_sequence_len-1,)),
    tf.keras.layers.Embedding(vocab_size, 100),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(150)),
    tf.keras.layers.Dense(vocab_size, activation='softmax')
])

adam = tf.keras.optimizers.Adam(learning_rate=0.01)
model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])

model.fit(xs, ys, epochs=100)
```

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