

Copyright Notice

These slides are distributed under the Creative Commons License.

[DeepLearning.AI](#) makes these slides available for educational purposes. You may not use or distribute these slides for commercial purposes. You may make copies of these slides and use or distribute them for educational purposes as long as you cite [DeepLearning.AI](#) as the source of the slides.

For the rest of the details of the license, see <https://creativecommons.org/licenses/by-sa/2.0/legalcode>

audio	"fashion_mnist"	text
	"horses_or_humans"	
"nsynth"	"image_label_folder"	"cnn_dailymail"
	"imagenet2012"	"glue"
image	"imagenet2012_corrupted"	"imdb_reviews"
	"kmnist"	"lm1b"
"abstract_reasoning"	"lsun"	"multi_nli"
"caltech101"	"mnist"	"squad"
"cats_vs_dogs"	"omniglot"	"wikipedia"
"celeb_a"	"open_images_v4"	"xnli"
"celeb_a_hq"	"oxford_iiit_pet"	
"cifar10"	"quickdraw_bitmap"	translate
"cifar100"	"rock_paper_scissors"	
"cifar10_corrupted"	"shapes3d"	"flores"
"coco2014"	"smallnorb"	"para_crawl"
"colorectal_histology"	"sun397"	"ted_hrlr_translate"
"cycle_gan"	"svhn_cropped"	"ted_multi_translate"
"diabetic_retinopathy..."	"tf_flowers"	"wmt15_translate"
"dsprites"		"wmt16_translate"
"dtd"	structured	"wmt17_translate"
"emnist"		"wmt18_translate"
	"higgs"	"wmt19_translate"
	"iris"	
	"titanic"	

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

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

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

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

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

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

```
import tensorflow_datasets as tfds
```

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

```
import numpy as np
```

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

```
training_sentences = []
```

```
training_labels = []
```

```
testing_sentences = []
```

```
testing_labels = []
```

```
for s,l in train_data:
```

```
    training_sentences.append(str(s.numpy()))
```

```
    training_labels.append(l.numpy())
```

```
for s,l in test_data:
```

```
    testing_sentences.append(str(s.numpy()))
```

```
    testing_labels.append(l.numpy())
```



```
training_sentences = []  
training_labels = []  
  
testing_sentences = []  
testing_labels = []
```

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

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

```
training_sentences = []
```

```
training_labels = []
```

```
testing_sentences = []
```

```
testing_labels = []
```

```
for s, l in train_data:
```

```
    training_sentences.append(str(s.numpy()))
```

```
    training_labels.append(l.numpy())
```

```
for s, l in test_data:
```

```
    testing_sentences.append(str(s.numpy()))
```

```
    testing_labels.append(l.numpy())
```

```
training_sentences = []
```

```
training_labels = []
```

```
testing_sentences = []
```

```
testing_labels = []
```

```
for s,l in train_data:
```

```
    training_sentences.append(str(s.numpy()))
```

```
    training_labels.append(l.numpy())
```

```
for s,l in test_data:
```

```
    testing_sentences.append(str(s.numpy()))
```

```
    testing_labels.append(l.numpy())
```

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

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

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

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

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

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

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

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

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

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

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

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



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

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

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

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

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

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

```
tokenizer = Tokenizer(num_words = vocab_size, oov_token=oov_tok)
```

```
tokenizer.fit_on_texts(training_sentences)
```

```
word_index = tokenizer.word_index
```

```
sequences = tokenizer.texts_to_sequences(training_sentences)
```

```
padded = pad_sequences(sequences,maxlen=max_length, truncating=trunc_type)
```

```
testing_sequences = tokenizer.texts_to_sequences(testing_sentences)
```

```
testing_padded = pad_sequences(testing_sequences,maxlen=max_length)
```

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

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

```
tokenizer = Tokenizer(num_words = vocab_size, oov_token=oov_tok)
```

```
tokenizer.fit_on_texts(training_sentences)
```

```
word_index = tokenizer.word_index
```

```
sequences = tokenizer.texts_to_sequences(training_sentences)
```

```
padded = pad_sequences(sequences,maxlen=max_length, truncating=trunc_type)
```

```
testing_sequences = tokenizer.texts_to_sequences(testing_sentences)
```

```
testing_padded = pad_sequences(testing_sequences,maxlen=max_length)
```

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

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

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

```
sequences = tokenizer.texts_to_sequences(training_sentences)
```

```
padded = pad_sequences(sequences,maxlen=max_length, truncating=trunc_type)
```

```
testing_sequences = tokenizer.texts_to_sequences(testing_sentences)
```

```
testing_padded = pad_sequences(testing_sequences,maxlen=max_length)
```

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

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

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

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

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

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

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

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

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

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



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

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

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

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

Layer (type)	Output Shape	Param #
=====		
embedding_11 (Embedding)	(None, 120, 16)	160000
<hr/>		
global_average_pooling1d_3 ((None, 16)	0	
<hr/>		
dense_16 (Dense)	(None, 6)	102
<hr/>		
dense_17 (Dense)	(None, 1)	7
=====		

Total params: 160,109
Trainable params: 160,109
Non-trainable params: 0

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

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

Epoch 8/10

25000/25000 [=====] -

6s 256us/sample - loss: 5.2086e-04 - acc: 1.0000 - val_loss: 0.7252 - val_acc: 0.8270

Epoch 9/10

25000/25000 [=====] -

6s 222us/sample - loss: 3.0199e-04 - acc: 1.0000 - val_loss: 0.7628 - val_acc: 0.8269

Epoch 10/10

25000/25000 [=====] -

6s 224us/sample - loss: 1.7872e-04 - acc: 1.0000 - val_loss: 0.7997 - val_acc: 0.8259


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

```
(10000, 16)
```

Hello : 1
World : 2
How : 3
Are : 4
You : 5

reverse_word_index = tokenizer.index_word

1 : Hello
2 : World
3 : How
4 : Are
5 : You

```
import io
```

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

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

```
for word_num in range(1, vocab_size):
```

```
    word = reverse_word_index[word_num]
```

```
    embeddings = weights[word_num]
```

```
    out_m.write(word + "\n")
```

```
    out_v.write('\t'.join([str(x) for x in embeddings]) + "\n")
```

```
out_v.close()
```

```
out_m.close()
```

```
import io
```

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

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

```
for word_num in range(1, vocab_size):
```

```
    word = reverse_word_index[word_num]
```

```
    embeddings = weights[word_num]
```

```
    out_m.write(word + "\n")
```

```
    out_v.write('\t'.join([str(x) for x in embeddings]) + "\n")
```

```
out_v.close()
```

```
out_m.close()
```

```
import io
```

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

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

```
for word_num in range(1, vocab_size):
```

```
    word = reverse_word_index[word_num]
```

```
    embeddings = weights[word_num]
```

```
    out_m.write(word + "\n")
```

```
    out_v.write('\t'.join([str(x) for x in embeddings]) + "\n")
```

```
out_v.close()
```

```
out_m.close()
```

```
try:
```

```
    from google.colab import files
```

```
except ImportError:
```

```
    pass
```

```
else:
```

```
    files.download('vecs.tsv')
```

```
    files.download('meta.tsv')
```

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

?

Home

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.

DATA

5 tensors found

Word2Vec 10K

☐ Sphereize data ?

Load data

Publish

Checkpoint: vecs.tsv

Metadata: meta.tsv

T-SNE

PCA

CUSTOM

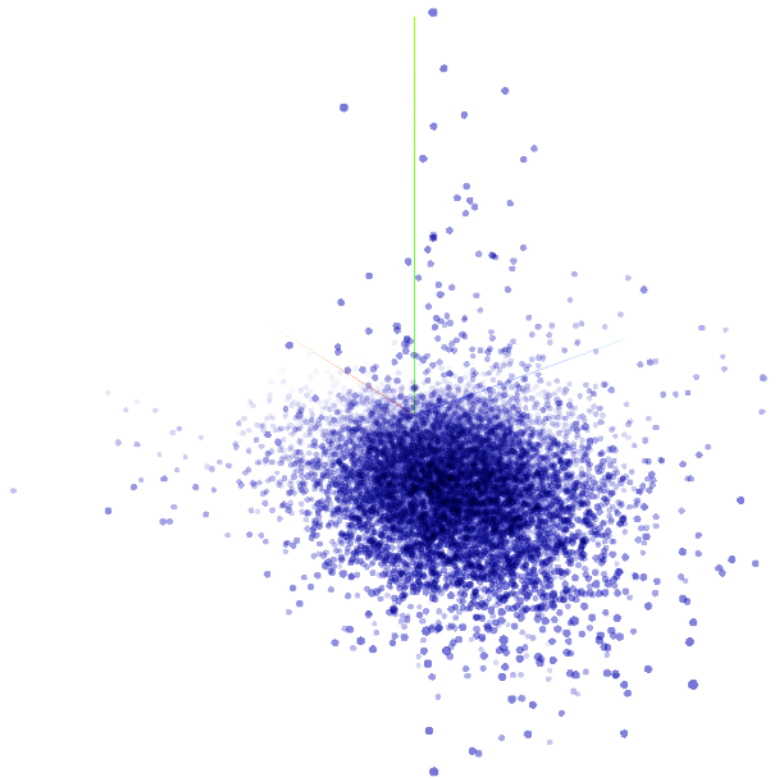
X
Component #1Y
Component #2Z
Component #3

PCA is approximate. ?

Total variance described: 98.7%.



Points: 9999 | Dimension: 16

Show All
DataIsolate
selectionClear
selection

Search



by



BOOKMARKS (0) ?



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%.



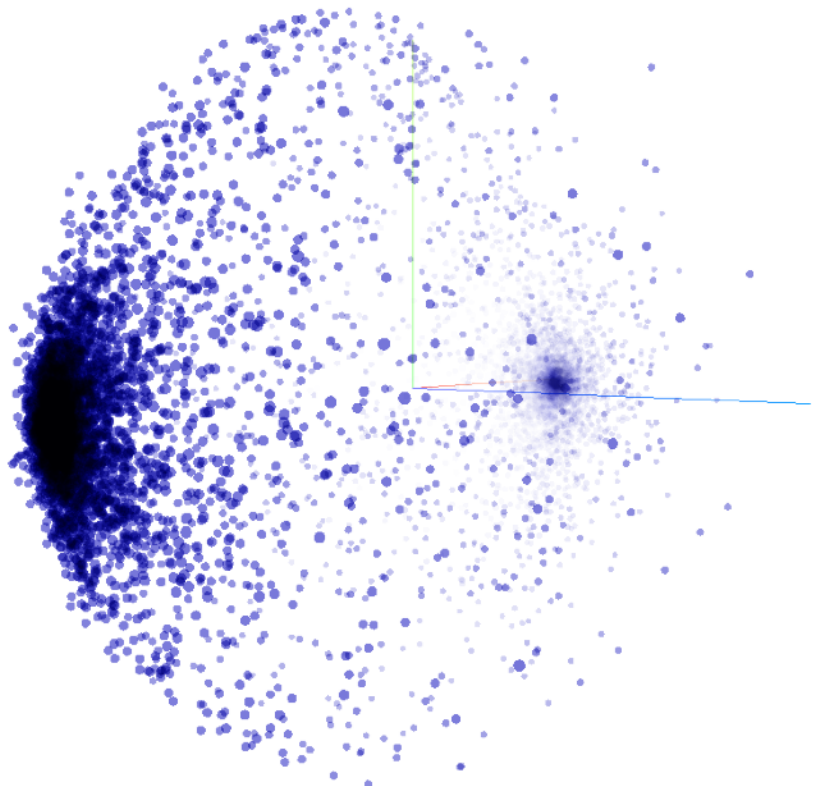
Points: 9999 | Dimension: 16

Show All
DataIsolate
selectionClear
selection

Search



by



BOOKMARKS (0) ?



```
import json
```

```
import tensorflow as tf
```

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

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

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

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

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

```
sentences = []  
labels = []
```

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

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



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

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

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

```
tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_tok)
```

```
tokenizer.fit_on_texts(training_sentences)
```

```
word_index = tokenizer.word_index
```

```
training_sequences = tokenizer.texts_to_sequences(training_sentences)
```

```
training_padded = pad_sequences(training_sequences, maxlen=max_length,  
                                padding=padding_type, truncating=trunc_type)
```

```
testing_sequences = tokenizer.texts_to_sequences(testing_sentences)
```

```
testing_padded = pad_sequences(testing_sequences, maxlen=max_length,  
                               padding=padding_type, truncating=trunc_type)
```

```
tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_tok)
```

```
tokenizer.fit_on_texts(training_sentences)
```

```
word_index = tokenizer.word_index
```

```
training_sequences = tokenizer.texts_to_sequences(training_sentences)
```

```
training_padded = pad_sequences(training_sequences, maxlen=max_length,  
                                padding=padding_type, truncating=trunc_type)
```

```
testing_sequences = tokenizer.texts_to_sequences(testing_sentences)
```

```
testing_padded = pad_sequences(testing_sequences, maxlen=max_length,  
                               padding=padding_type, truncating=trunc_type)
```

```
tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_tok)
```

```
tokenizer.fit_on_texts(training_sentences)
```

```
word_index = tokenizer.word_index
```

```
training_sequences = tokenizer.texts_to_sequences(training_sentences)
```

```
training_padded = pad_sequences(training_sequences, maxlen=max_length,  
                                padding=padding_type, truncating=trunc_type)
```

```
testing_sequences = tokenizer.texts_to_sequences(testing_sentences)
```

```
testing_padded = pad_sequences(testing_sequences, maxlen=max_length,  
                               padding=padding_type, truncating=trunc_type)
```

```
tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_tok)
```

```
tokenizer.fit_on_texts(training_sentences)
```

```
word_index = tokenizer.word_index
```

```
training_sequences = tokenizer.texts_to_sequences(training_sentences)
```

```
training_padded = pad_sequences(training_sequences, maxlen=max_length,  
                                padding=padding_type, truncating=trunc_type)
```

```
testing_sequences = tokenizer.texts_to_sequences(testing_sentences)
```

```
testing_padded = pad_sequences(testing_sequences, maxlen=max_length,  
                               padding=padding_type, truncating=trunc_type)
```

```
tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_tok)
```

```
tokenizer.fit_on_texts(training_sentences)
```

```
word_index = tokenizer.word_index
```

```
training_sequences = tokenizer.texts_to_sequences(training_sentences)
```

```
training_padded = pad_sequences(training_sequences, maxlen=max_length,  
                                padding=padding_type, truncating=trunc_type)
```

```
testing_sequences = tokenizer.texts_to_sequences(testing_sentences)
```

```
testing_padded = pad_sequences(testing_sequences, maxlen=max_length,  
                               padding=padding_type, truncating=trunc_type)
```



```
tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_tok)
```

```
tokenizer.fit_on_texts(training_sentences)
```

```
word_index = tokenizer.word_index
```

```
training_sequences = tokenizer.texts_to_sequences(training_sentences)
```

```
training_padded = pad_sequences(training_sequences, maxlen=max_length,  
                                padding=padding_type, truncating=trunc_type)
```

```
testing_sequences = tokenizer.texts_to_sequences(testing_sentences)
```

```
testing_padded = pad_sequences(testing_sequences, maxlen=max_length,  
                               padding=padding_type, truncating=trunc_type)
```

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

model.summary()

Layer (type)	Output Shape	Param #
embedding_2 (Embedding)	(None, 32, 16)	160000
global_average_pooling1d_2 (GlobalAveragePooling1D)	(None, 16)	0
dense_4 (Dense)	(None, 24)	408
dense_5 (Dense)	(None, 1)	25

Total params: 160,433
Trainable params: 160,433
Non-trainable params: 0

```
num_epochs = 30
```

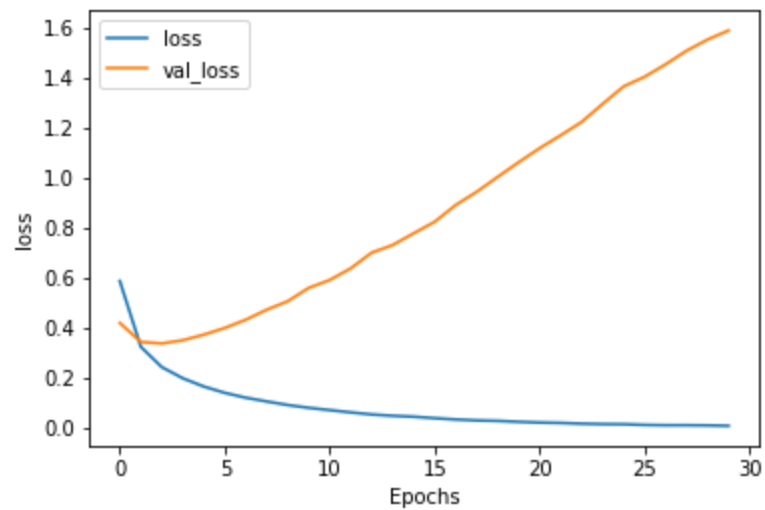
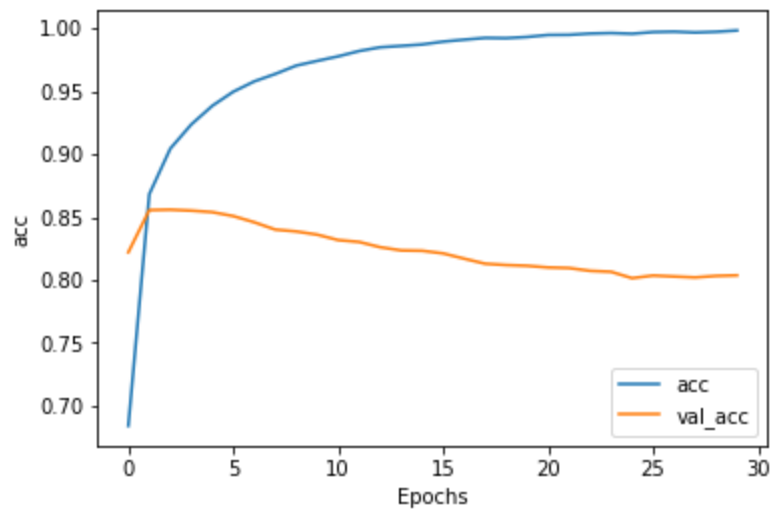
```
history = model.fit(training_padded, training_labels, epochs=num_epochs,  
                    validation_data=(testing_padded, testing_labels), verbose=2)
```

```
import matplotlib.pyplot as plt
```

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

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

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



vocab_size = 1000 (was 10,000)

embedding_dim = 16

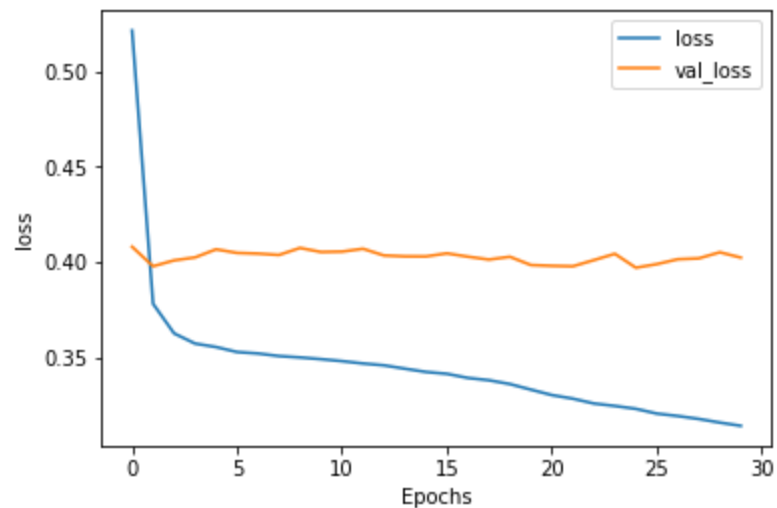
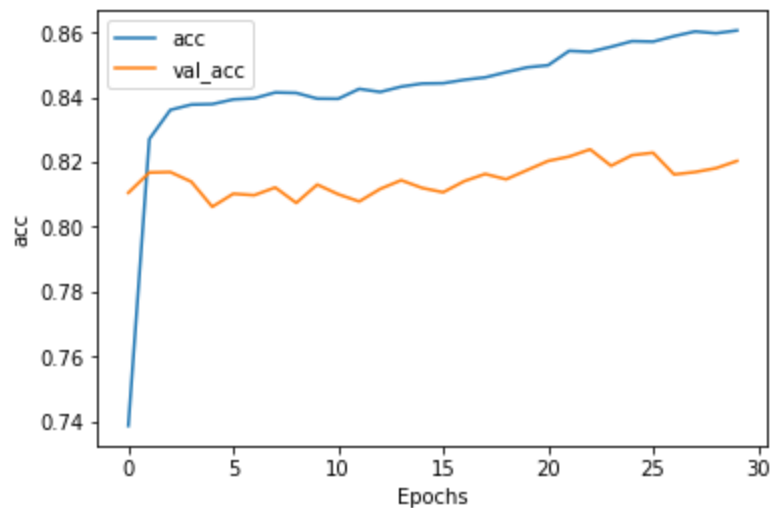
max_length = 16 (was 32)

trunc_type='post'

padding_type='post'

oov_tok = "<OOV>"

training_size = 20000



vocab_size = 1000 (was 10,000)

embedding_dim = 32 (was 16)

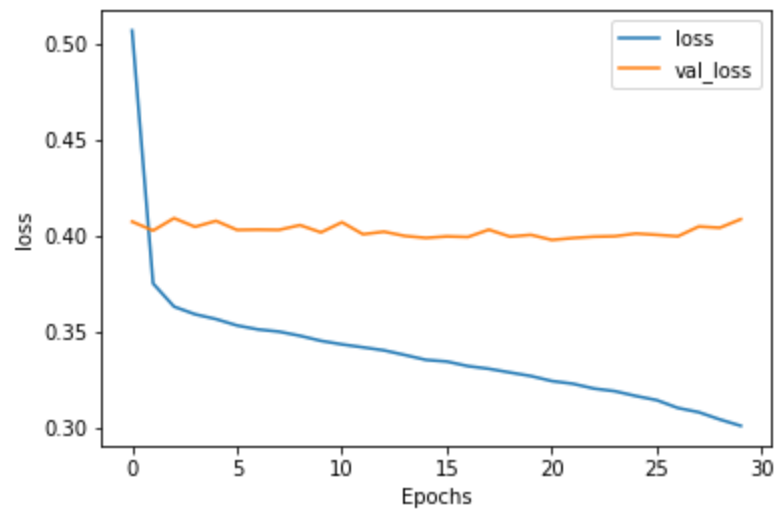
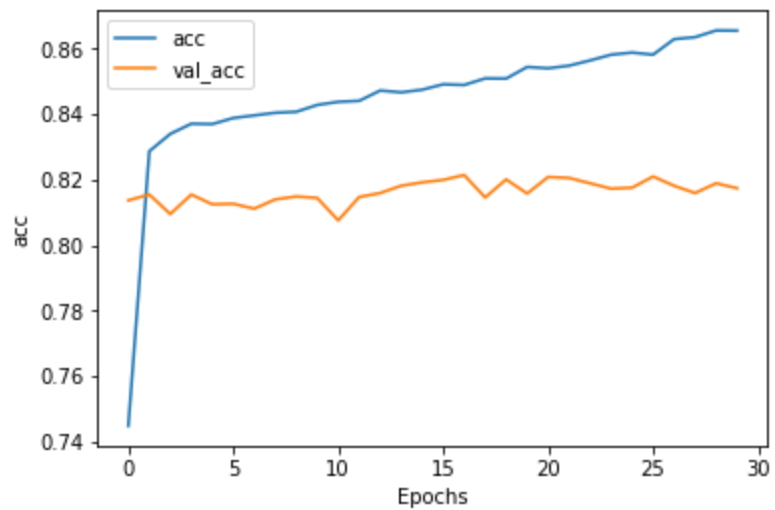
max_length = 16 (was 32)

trunc_type='post'

padding_type='post'

oov_tok = "<OOV>"

training_size = 20000



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

275 lines (209 sloc) | 9.65 KB

<> [icon] Raw Blame [icon] [icon] [icon]

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
<code>'test'</code>	25,000
<code>'train'</code>	25,000
<code>'unsupervised'</code>	50,000

275 lines (209 sloc) | 9.65 KB

<> [icon] Raw Blame [icon] [icon] [icon]

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 %}

275 lines (209 sloc) | 9.65 KB

<> [icon] Raw Blame [icon] [icon] [icon]

imdb_reviews/bytes

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

- **Features:**

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

- **Examples** ([tfds.as_dataframe](#)):

```
{% framebox %}
```

Display examples...

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

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

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

```
{% endframebox %}
```

imdb_reviews/bytes

275 lines (209 sloc) | 9.65 KB

<> [icon] Raw Blame [icon] [icon] [icon]

imdb_reviews/subwords8k

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

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

- **Examples** ([tfds.as_dataframe](#)):

{% framebox %}

Display examples...

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

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

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

{% endframebox %}

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



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

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

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

```
print(tokenizer_subwords.subwords)
```

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

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

```
tokenized_string = tokenizer_subwords.encode(sample_string)
```

```
print ('Tokenized string is {}'.format(tokenized_string))
```

```
original_string = tokenizer_subwords.decode(tokenized_string)
```

```
print ('The original string: {}'.format(original_string))
```

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

The original string: TensorFlow, from basics to mastery

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

```
tokenized_string = tokenizer_subwords.encode(sample_string)
```

```
print ('Tokenized string is {}'.format(tokenized_string))
```

```
original_string = tokenizer_subwords.decode(tokenized_string)
```

```
print ('The original string: {}'.format(original_string))
```

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

The original string: TensorFlow, from basics to mastery

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

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

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

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

The original string: TensorFlow, from basics to mastery

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

```
tokenized_string = tokenizer_subwords.encode(sample_string)
```

```
print ('Tokenized string is {}'.format(tokenized_string))
```

```
original_string = tokenizer_subwords.decode(tokenized_string)
```

```
print ('The original string: {}'.format(original_string))
```

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

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

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

6307 ----> Ten

2327 ----> sor

4043 ----> FI

2120 ----> ow

2 ----> ,

48 ----> from

4249 ----> basi

4429 ----> cs

7 ----> to

2652 ----> master

8050 ----> y


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

model.summary()
```

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

model.summary()
```

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

```
num_epochs = 10
```

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

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

```
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")
```

