```
import matplotlib.pyplot as plt
import os
import re
import shutil
import string
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras import losses
print(tf.__version__)
```

```
2.15.0
```

```
url = "https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz"

dataset = tf.keras.utils.get_file("aclImdb_v1", url,
                                  untar=True, cache_dir='.',
                                  cache_subdir='')

dataset_dir = os.path.join(os.path.dirname(dataset), 'aclImdb')
```

```
Downloading data from https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz
84125825/84125825 [==============================] - 6s 0us/step
```

```
os.listdir(dataset_dir)
```

```
['imdb.vocab', 'test', 'train', 'imdbEr.txt', 'README']
```

```
train_dir = os.path.join(dataset_dir, 'train')
os.listdir(train_dir)
```

```
['urls_pos.txt',
 'unsupBow.feat',
 'neg',
 'urls_neg.txt',
 'unsup',
 'labeledBow.feat',
 'urls_unsup.txt',
 'pos']
```

```
sample_file = os.path.join(train_dir, 'pos/1181_9.txt')
with open(sample_file) as f:
  print(f.read())
```

```
Rachel Griffiths writes and directs this award winning short film. A heartwarming story about coping with grief and cherishing the m
```

```
remove_dir = os.path.join(train_dir, 'unsup')
shutil.rmtree(remove_dir)
```

```
batch_size = 32
seed = 42

raw_train_ds = tf.keras.utils.text_dataset_from_directory(
    'aclImdb/train',
    batch_size=batch_size,
    validation_split=0.2,
    subset='training',
    seed=seed)
```

```
Found 25000 files belonging to 2 classes.
Using 20000 files for training.
```

```
for text_batch, label_batch in raw_train_ds.take(1):
  for i in range(3):
    print("Review", text_batch.numpy()[i])
    print("Label", label_batch.numpy()[i])
```

```
Review b'"Pandemonium" is a horror movie spoof that comes off more stupid than funny. Believe me when I tell you, I love comedies. E
Label 0
Review b"David Mamet is a very interesting and a very un-equal director. His first movie 'House of Games' was the one I liked best,
Label 0
Review b'Great documentary about the lives of NY firefighters during the worst terrorist attack of all time.. That reason alone is v
Label 1
```

```
print("Label 0 corresponds to", raw_train_ds.class_names[0])
print("Label 1 corresponds to", raw_train_ds.class_names[1])
```

```
Label 0 corresponds to neg
Label 1 corresponds to pos
```

```
raw_val_ds = tf.keras.utils.text_dataset_from_directory(
    'aclImdb/train',
    batch_size=batch_size,
    validation_split=0.2,
    subset='validation',
    seed=seed)
```

```
Found 25000 files belonging to 2 classes.
Using 5000 files for validation.
```

```
raw_test_ds = tf.keras.utils.text_dataset_from_directory(
    'aclImdb/test',
    batch_size=batch_size)
```

```
Found 25000 files belonging to 2 classes.
```

```
def custom_standardization(input_data):
  lowercase = tf.strings.lower(input_data)
  stripped_html = tf.strings.regex_replace(lowercase, '<br />', ' ')
  return tf.strings.regex_replace(stripped_html,
                                  '[%s]' % re.escape(string.punctuation),
                                  '')
```

```
max_features = 10000
sequence_length = 250

vectorize_layer = layers.TextVectorization(
    standardize=custom_standardization,
    max_tokens=max_features,
    output_mode='int',
    output_sequence_length=sequence_length)
```

```
# Make a text-only dataset (without labels), then call adapt
train_text = raw_train_ds.map(lambda x, y: x)
vectorize_layer.adapt(train_text)
```

```
def vectorize_text(text, label):
  text = tf.expand_dims(text, -1)
  return vectorize_layer(text), label
```

```
# retrieve a batch (of 32 reviews and labels) from the dataset
text_batch, label_batch = next(iter(raw_train_ds))
first_review, first_label = text_batch[0], label_batch[0]
print("Review", first_review)
print("Label", raw_train_ds.class_names[first_label])
print("Vectorized review", vectorize_text(first_review, first_label))
```

```
Review tf.Tensor(b'Great movie - especially the music - Etta James - "At Last". This speaks volumes when you have finally found that
Label neg
Vectorized review (<tf.Tensor: shape=(1, 250), dtype=int64, numpy=
array([[  86,   17,  260,    2,  222,    1,  571,   31,  229,   11, 2418,
           1,   51,   22,   25,  404,  251,   12,  306,  282,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0]])>, <tf.Tensor: shape=(),
```

```
print("1287 ---> ",vectorize_layer.get_vocabulary()[1287])
print(" 313 ---> ",vectorize_layer.get_vocabulary()[313])
print('Vocabulary size: {}'.format(len(vectorize_layer.get_vocabulary())))
```

```
1287 --->  silent
 313 --->  night
Vocabulary size: 10000
```

```
train_ds = raw_train_ds.map(vectorize_text)
val_ds = raw_val_ds.map(vectorize_text)
test_ds = raw_test_ds.map(vectorize_text)
```

```
AUTOTUNE = tf.data.AUTOTUNE
train_ds = train_ds.cache().prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
test_ds = test_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

```
embedding_dim = 16
```

```
model = tf.keras.Sequential([
  layers.Embedding(max_features, embedding_dim),
  layers.Dropout(0.2),
  layers.GlobalAveragePooling1D(),
  layers.Dropout(0.2),
  layers.Dense(1)])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, None, 16)          160000

 dropout (Dropout)           (None, None, 16)          0

 global_average_pooling1d (  (None, 16)                0
 GlobalAveragePooling1D)

 dropout_1 (Dropout)         (None, 16)                0

 dense (Dense)               (None, 1)                 17

=================================================================
Total params: 160017 (625.07 KB)
Trainable params: 160017 (625.07 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model.compile(loss=losses.BinaryCrossentropy(from_logits=True),
              optimizer='adam',
              metrics=tf.metrics.BinaryAccuracy(threshold=0.0))
```

```
epochs = 10
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=epochs)
```

```
Epoch 1/10
625/625 [==============================] - 11s 16ms/step - loss: 0.6623 - binary_accuracy: 0.6956 - val_loss: 0.6113 - val_binary_ac
Epoch 2/10
625/625 [==============================] - 6s 9ms/step - loss: 0.5447 - binary_accuracy: 0.8048 - val_loss: 0.4949 - val_binary_accu
Epoch 3/10
625/625 [==============================] - 4s 7ms/step - loss: 0.4410 - binary_accuracy: 0.8478 - val_loss: 0.4179 - val_binary_accu
Epoch 4/10
625/625 [==============================] - 5s 8ms/step - loss: 0.3767 - binary_accuracy: 0.8670 - val_loss: 0.3725 - val_binary_accu
Epoch 5/10
625/625 [==============================] - 4s 6ms/step - loss: 0.3342 - binary_accuracy: 0.8803 - val_loss: 0.3444 - val_binary_accu
Epoch 6/10
625/625 [==============================] - 4s 6ms/step - loss: 0.3033 - binary_accuracy: 0.8905 - val_loss: 0.3251 - val_binary_accu
Epoch 7/10
625/625 [==============================] - 8s 13ms/step - loss: 0.2808 - binary_accuracy: 0.8986 - val_loss: 0.3124 - val_binary_ac
Epoch 8/10
625/625 [==============================] - 8s 12ms/step - loss: 0.2606 - binary_accu
Epoch 9/10
625/625 [==============================] - 5s 8ms/step - loss: 0.2453 - binary_a
Epoch 10/10
625/625 [==============================] - 4s 6ms/step - loss: 0.2305 - binary_acc
```

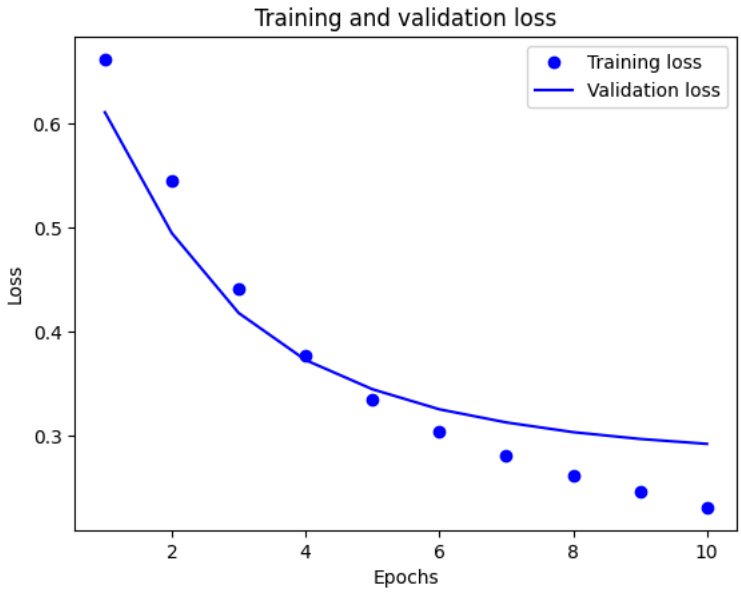
```
loss, accuracy = model.evaluate(test_ds)
print("Loss: ", loss)
print("Accuracy: ", accuracy)
```

```
782/782 [==============================] - 4s 5ms/step - loss: 0.3103 - binary_accuracy: 0.8730
Loss:  0.3102893829345703
Accuracy:  0.8729599714279175
```

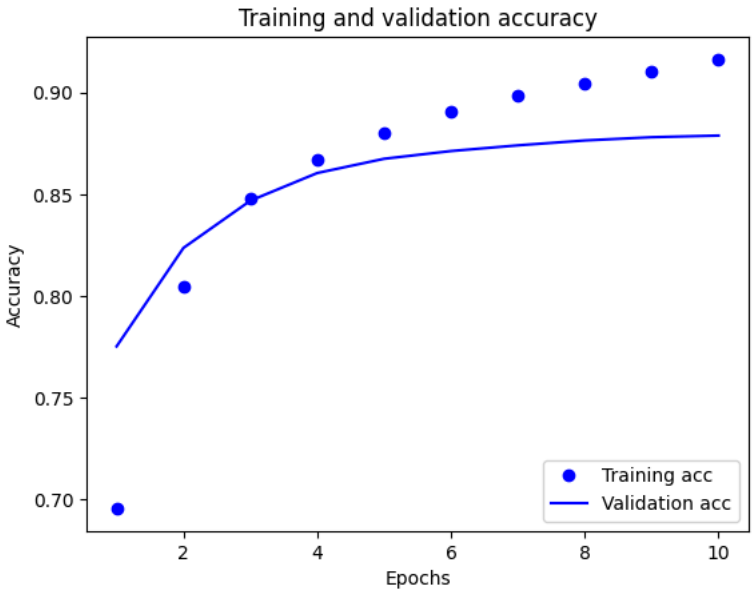
```
history_dict = history.history
history_dict.keys()
```

```
dict_keys(['loss', 'binary_accuracy', 'val_loss', 'val_binary_accuracy'])
```

```
acc = history_dict['binary_accuracy']
val_acc = history_dict['val_binary_accuracy']
loss = history_dict['loss']
val_loss = history_dict['val_loss']

epochs = range(1, len(acc) + 1)

# "bo" is for "blue dot"
plt.plot(epochs, loss, 'bo', label='Training loss')
# b is for "solid blue line"
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```




```
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')

plt.show()
```




```
export_model = tf.keras.Sequential([
  vectorize_layer,
  model,
  layers.Activation('sigmoid')
])
export_model.compile(
    loss=losses.BinaryCrossentropy(from_logits=False), optimizer="adam", metrics=['accuracy']
)
# Test it with `raw_test_ds`, which yields raw strings
loss, accuracy = export_model.evaluate(raw_test_ds)
print(accuracy)
```

```
     782/782 [==============================] - 5s 6ms/step - loss: 0.3103 - accuracy: 0.8730
     0.8729599714279175
```

```
examples = [
  "The movie was great!",
  "The movie was okay.",
  "The movie was terrible..."
]
export_model.predict(examples)
```

```
     1/1 [==============================] - 0s 218ms/step
     array([[0.60859257],
            [0.4288    ],
            [0.34703887]], dtype=float32)
```