```
#@title Licensed under the Apache License, Version 2.0 (the "License")
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# https://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
```

Licensed under the Apache License, Version 2.0 (the "License");


```
# NOTE: PLEASE MAKE SURE YOU ARE RUNNING THIS IN A PYTHON3 ENVIRONMENT

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

```
2.3.0
```

```
# Double check TF 2.0x is installed. If you ran the above block, there was a
# 'reset all runtimes' button at the bottom that you needed to press
import tensorflow as tf
print(tf.__version__)
```

```
2.3.0
```

```
# If the import fails, run this
# !pip install -q tensorflow-datasets

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

```
Downloading and preparing dataset imdb_reviews/subwords8k/1.0.0 (download: 80.23 MiB, ge
```

```
Dl Completed...: 100%  1/1 [00:03<00:00, 3.41s/ url]
Dl Size...: 100%  80/80 [00:03<00:00, 23.63 MiB/s]
```

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

```
21%  7873/25000 [00:00<00:00, 78725.76 examples/s]
```

```
#build in subwords encoder
tokenizer = info.features['text'].encoder
```

```
print(tokenizer.subwords)
```

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

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

#using the built in tokenizer, encoding and decoding
#sentences are as simple as calling the appropriate methods
tokenized_string = tokenizer.encode(sample_string)
print ('Tokenized string is {}'.format(tokenized_string))

original_string = tokenizer.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.decode([ts])))
```

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

```
BUFFER_SIZE = 10000
BATCH_SIZE = 64
```

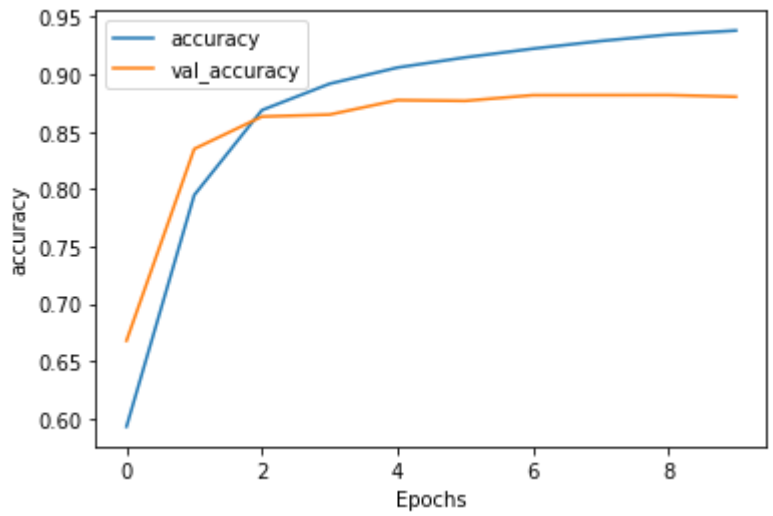
```
train_dataset = train_data.shuffle(BUFFER_SIZE)
train_dataset = train_dataset.padded_batch(BATCH_SIZE, tf.compat.v1.data.get_output_shapes(tr
test_dataset = test_data.padded_batch(BATCH_SIZE, tf.compat.v1.data.get_output_shapes(test_da


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

model.summary()
```

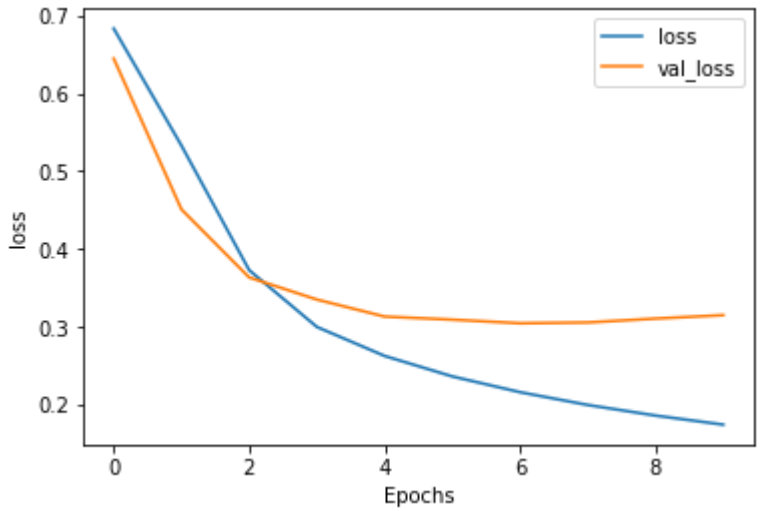
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_1 (Embedding)      (None, None, 64)          523840    
_________________________________________________________________
global_average_pooling1d_1 ( (None, 64)                0         
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 390       
_________________________________________________________________
dense_3 (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_dataset)
```

```
Epoch 1/10
391/391 [==============================] - 13s 33ms/step - loss: 0.6832 - accuracy: 0.59
Epoch 2/10
391/391 [==============================] - 13s 33ms/step - loss: 0.5328 - accuracy: 0.79
```

```
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")
```






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

```
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, tokenizer.vocab_size):
  word = tokenizer.decode([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')
```

```
(8185, 64)
```