```
#@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");

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

# !pip install -q tensorflow-datasets
```

```
2.3.0
```

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

```
Downloading and preparing dataset imdb_reviews/plain_text/1.0.0 (download: 80.23 MiB, ge
Dl Completed...: 100%  1/1 [00:09<00:00, 9.51s/ url]
Dl Size...: 100%  80/80 [00:09<00:00, 8.44 MiB/s]

Shuffling and writing examples to /root/tensorflow_datasets/imdb_reviews/plain_text/1.0
40%  9939/25000 [00:00<00:00, 99387.49 examples/s]
Shuffling and writing examples to /root/tensorflow_datasets/imdb_reviews/plain_text/1.0
40%  10047/25000 [00:00<00:00, 100465.54 examples/s]
Shuffling and writing examples to /root/tensorflow_datasets/imdb_reviews/plain_text/1.0
94%  46959/50000 [00:00<00:00, 87793.45 examples/s]
Dataset imdb_reviews downloaded and prepared to /root/tensorflow_datasets/imdb_reviews/p
```

```
import numpy as np

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

training_sentences = []
training_labels = []
```

```
testing_sentences = []
testing_labels = []

# str(s.tonumpy()) is needed in Python3 instead of just s.numpy()
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_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)



reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])

def decode_review(text):
    return ' '.join([reverse_word_index.get(i, '?') for i in text])

print(decode_review(padded[1]))
print(training_sentences[1])
```

```
to a combination of things including really tired being warm and comfortable on the <OOV
ion of things including, really tired, being warm and comfortable on the sette and havin
```

```
model = tf.keras.Sequential([
    #using Gated Recurrent Units
```

```
    #using Gated-Recurrent-Units
    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()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding (Embedding)        (None, 120, 16)           160000    
_________________________________________________________________
bidirectional (Bidirectional (None, 64)                9600      
_________________________________________________________________
dense (Dense)                (None, 6)                 390       
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 7         
=================================================================
Total params: 169,997
Trainable params: 169,997
Non-trainable params: 0
_________________________________________________________________
```

```
num_epochs = 50
history = model.fit(padded, training_labels_final, epochs=num_epochs, validation_data=(testin
```

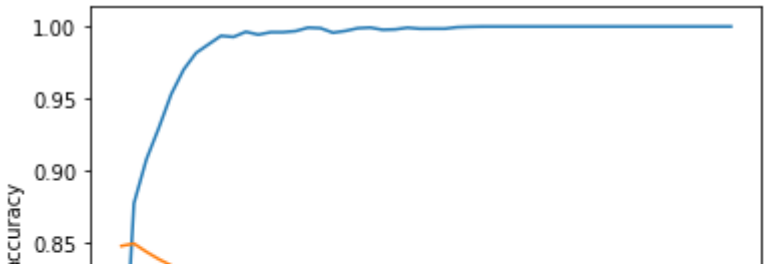
```
Epoch 22/50
782/782 [==============================] - 16s 21ms/step - loss: 0.0069 - accuracy: 0.99
Epoch 23/50
782/782 [==============================] - 16s 21ms/step - loss: 0.0065 - accuracy: 0.99
Epoch 24/50
782/782 [==============================] - 16s 21ms/step - loss: 0.0036 - accuracy: 0.99
Epoch 25/50
782/782 [==============================] - 16s 21ms/step - loss: 0.0041 - accuracy: 0.99
Epoch 26/50
782/782 [==============================] - 16s 21ms/step - loss: 0.0042 - accuracy: 0.99
Epoch 27/50
782/782 [==============================] - 16s 21ms/step - loss: 0.0041 - accuracy: 0.99
Epoch 28/50
782/782 [==============================] - 16s 21ms/step - loss: 0.0015 - accuracy: 0.99
Epoch 29/50
782/782 [==============================] - 16s 21ms/step - loss: 6.7440e-04 - accuracy:
Epoch 30/50
782/782 [==============================] - 16s 21ms/step - loss: 4.0974e-05 - accuracy:
Epoch 31/50
782/782 [==============================] - 16s 21ms/step - loss: 1.4797e-05 - accuracy:
Epoch 32/50
782/782 [==============================] - 17s 21ms/step - loss: 1.0032e-05 - accuracy:
Epoch 33/50
782/782 [==============================] - 16s 21ms/step - loss: 7.0181e-06 - accuracy:
Epoch 34/50
782/782 [==============================] - 16s 21ms/step - loss: 4.8897e-06 - accuracy:
Epoch 35/50
782/782 [==============================] - 16s 21ms/step - loss: 3.3824e-06 - accuracy:
Epoch 36/50
782/782 [==============================] - 17s 22ms/step - loss: 2.3240e-06 - accuracy:
Epoch 37/50
782/782 [==============================] - 16s 21ms/step - loss: 1.5892e-06 - accuracy:
Epoch 38/50
782/782 [==============================] - 16s 21ms/step - loss: 1.0780e-06 - accuracy:
Epoch 39/50
782/782 [==============================] - 16s 21ms/step - loss: 7.2558e-07 - accuracy:
Epoch 40/50
782/782 [==============================] - 16s 21ms/step - loss: 4.8802e-07 - accuracy:
Epoch 41/50
782/782 [==============================] - 16s 21ms/step - loss: 3.2830e-07 - accuracy:
Epoch 42/50
782/782 [==============================] - 16s 21ms/step - loss: 2.1964e-07 - accuracy:
Epoch 43/50
782/782 [==============================] - 16s 20ms/step - loss: 1.4755e-07 - accuracy:
Epoch 44/50
782/782 [==============================] - 16s 21ms/step - loss: 9.9085e-08 - accuracy:
Epoch 45/50
782/782 [==============================] - 16s 21ms/step - loss: 6.6814e-08 - accuracy:
Epoch 46/50
782/782 [==============================] - 16s 21ms/step - loss: 4.5283e-08 - accuracy:
Epoch 47/50
782/782 [==============================] - 16s 21ms/step - loss: 3.0816e-08 - accuracy:
Epoch 48/50
782/782 [==============================] - 16s 21ms/step - loss: 2.1145e-08 - accuracy:
Epoch 49/50
782/782 [==============================] - 16s 21ms/step - loss: 1.4621e-08 - accuracy:
Epoch 50/50
```

```
782/782 [==============================] - 17s 21ms/step - loss: 1.0212e-08 - accuracy:
```

```
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')
```




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

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_1 (Embedding)      (None, 120, 16)           160000    
_________________________________________________________________
bidirectional_1 (Bidirection (None, 64)                12544     
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 390       
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 7         
=================================================================
Total params: 172,941
Trainable params: 172,941
Non-trainable params: 0
_________________________________________________________________
```

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