```
#@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");

Multiple Layer LSTM

```
from __future__ import absolute_import, division, print_function, unicode_literals


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

```
2.3.0
```

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

```
2.3.0
```

```
# Get the data
dataset, info = tfds.load('imdb_reviews/subwords8k', with_info=True, as_supervised=True)
train_dataset, test_dataset = dataset['train'], dataset['test']
```

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

```
Dl Completed...: 100%  1/1 [00:08<00:00, 8.21s/ url]
Dl Size...: 100%  80/80 [00:08<00:00, 9.77 MiB/s]
```

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

```
Shuffling and writing examples to /root/tensorflow_datasets/imdb_reviews/subwords8k/1.0
```

```
BUFFER_SIZE = 10000
BATCH_SIZE = 64

train_dataset = train_dataset.shuffle(BUFFER_SIZE)
train_dataset = train_dataset.padded_batch(BATCH_SIZE, train_dataset.output_shapes)
test_dataset = test_dataset.padded_batch(BATCH_SIZE, test_dataset.output_shapes)
```

```
WARNING:tensorflow:From <ipython-input-5-51766d5ffb66>:5: DatasetV1.output_shapes (from
Instructions for updating:
Use `tf.compat.v1.data.get_output_shapes(dataset)`.
WARNING:tensorflow:From <ipython-input-5-51766d5ffb66>:5: DatasetV1.output_shapes (from
Instructions for updating:
Use `tf.compat.v1.data.get_output_shapes(dataset)`.
```

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(tokenizer.vocab_size, 64),
    #double layer LSTM, whenever we feed an LSTM to an LSTM, specify return_sequences=True
    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.summary()
```

```
Model: "sequential"
_________________________________________________________________
```

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

```
embedding (Embedding)        (None, None, 64)          523840
```

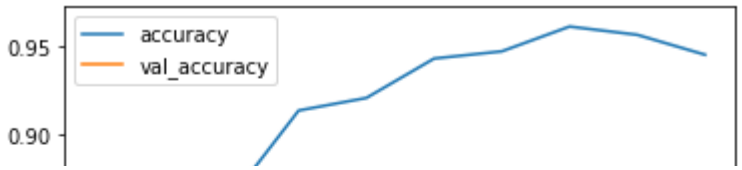
```
NUM_EPOCHS = 10
history = model.fit(train_dataset, epochs=NUM_EPOCHS, validation_data=test_dataset)
```

```
Epoch 1/10
391/391 [==============================] - 109s 278ms/step - loss: 0.5338 - accuracy: 0
Epoch 2/10
391/391 [==============================] - 107s 273ms/step - loss: 0.3458 - accuracy: 0
Epoch 3/10
391/391 [==============================] - 108s 277ms/step - loss: 0.3328 - accuracy: 0
Epoch 4/10
391/391 [==============================] - 110s 280ms/step - loss: 0.2315 - accuracy: 0
Epoch 5/10
391/391 [==============================] - 109s 280ms/step - loss: 0.2122 - accuracy: 0
Epoch 6/10
391/391 [==============================] - 109s 279ms/step - loss: 0.1603 - accuracy: 0
Epoch 7/10
391/391 [==============================] - 109s 279ms/step - loss: 0.1486 - accuracy: 0
Epoch 8/10
391/391 [==============================] - 109s 278ms/step - loss: 0.1149 - accuracy: 0
Epoch 9/10
391/391 [==============================] - 108s 277ms/step - loss: 0.1222 - accuracy: 0
Epoch 10/10
391/391 [==============================] - 107s 274ms/step - loss: 0.1498 - accuracy: 0
```

```
import matplotlib.pyplot as plt
#graphs are more smoother

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')
```