```
!pip install tensorflow-datasets
```

+ Code + Text

1.Importing Necessary Libraries

```
import numpy as np
import tensorflow as tf
import tensorflow_datasets as tdfs
```

2.Loading Data

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

```
imdb,info
```

```
({'train': <_PrefetchDataset element_spec=(TensorSpec(shape=(), dtype=tf.string, name=None), TensorSpec(shape=(), dtype=tf.int64, name=None))>,
  'test': <_PrefetchDataset element_spec=(TensorSpec(shape=(), dtype=tf.string, name=None), TensorSpec(shape=(), dtype=tf.int64, name=None))>,
  'unsupervised': <_PrefetchDataset element_spec=(TensorSpec(shape=(), dtype=tf.string, name=None), TensorSpec(shape=(), dtype=tf.int64, name=None))>},
 tfds.core.DatasetInfo(
    name='imdb_reviews',
    full_name='imdb_reviews/plain_text/1.0.0',
    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.
    """,
    config_description="""
    Plain text
    """,
    homepage='http://ai.stanford.edu/~amaas/data/sentiment/',
    data_dir='C:\\Users\\sritu\\tensorflow_datasets\\imdb_reviews\\plain_text\\1.0.0',
    file_format=tfrecord,
    download_size=80.23 MiB,
    dataset_size=129.83 MiB,
    features=FeaturesDict({
        'label': ClassLabel(shape=(), dtype=int64, num_classes=2),
        'text': Text(shape=(), dtype=string),
    }),
    supervised_keys=('text', 'label'),
    disable_shuffling=False,
    splits={
        'test': <SplitInfo num_examples=25000, num_shards=1>,
        'train': <SplitInfo num_examples=25000, num_shards=1>,
        'unsupervised': <SplitInfo num_examples=50000, num_shards=1>,
    },
    citation="""@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}
    }""",
))
```

3.Text Processing

1.Train test split

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

```
train_data
```

```
<_PrefetchDataset element_spec=(TensorSpec(shape=(), dtype=tf.string, name=None), TensorSpec(shape=(), dtype=tf.int64, name=None))>
```

2.Extracting text, labels from train_data, test_data

```
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_labels
```

```
[0,
 0,
 0,
 1,
```

```
1,
1,
0,
0,
0,
0,
0,
1,
1,
0,
1,
0,
1,
1,
1,
0,
1,
1,
1,
1,
1,
0,
0,
0,
1,
0,
0,
0,
0,
0,
0,
1,
1,
0,
0,
1,
0,
0,
0,
0,
0,
0,
1,
0,
1,
1,
1,
1,
1,
0,
1,
0,
1,
```

3.Converting to numpy arrays

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

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

print(training_sentences.shape,testing_sentences.shape)
print(training_sentences[0])
print(training_labels[0])
```

```
(25000,) (25000,)
b"This was an absolutely terrible movie. Don't be lured in by Christopher Walken or Michael Ironside. Both are great actors, but this must simply be their worst role
0
```

Unsupported Cell Type. Double-Click to inspect/edit the content.

4.Tokenization and padding

Unsupported Cell Type. Double-Click to inspect/edit the content.

```
vocab_size = 10000
embedding_dim = 16  #[16 dimensions]
max_length = 120
#Just considering first 120 words for every review
trunc_type = 'post'
oov_tok = ""#Any new words not in vocab_size will be denoted as <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)
#Stores words, index in a dictionary
word_index = tokenizer.word_index
print(word_index)
```

```
{'': 1, 'the': 2, 'and': 3, 'a': 4, 'of': 5, 'to': 6, 'is': 7, 'br': 8, 'in': 9, 'it': 10, 'i': 11, 'this': 12, 'that': 13, 'was': 14, 'as': 15, 'for': 16, 'with': 1
```

Double-click (or enter) to edit

```
#Integer encoding and padding
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)
```

4.Model Building

1.add embedding layer 2.add rnn 3.add hidden layer 4.add output layer 5.compile the model 6.fit the model

Unsupported Cell Type. Double-Click to inspect/edit the content.

```
import tensorflow as tf

# Assuming you have defined vocab_size, embedding_dim, and max_length

model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim,input_shape=(120,)),
    tf.keras.layers.SimpleRNN(32),
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
C:\Users\sritu\anaconda3\Lib\site-packages\keras\src\layers\core\embedding.py:81: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When usi
  super().__init__(**kwargs)
```

```
model.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
embedding_2 (Embedding)	(None, 120, 16)	160,000
simple_rnn_2 (SimpleRNN)	(None, 32)	1,568
dense_4 (Dense)	(None, 10)	330
dense_5 (Dense)	(None, 1)	11

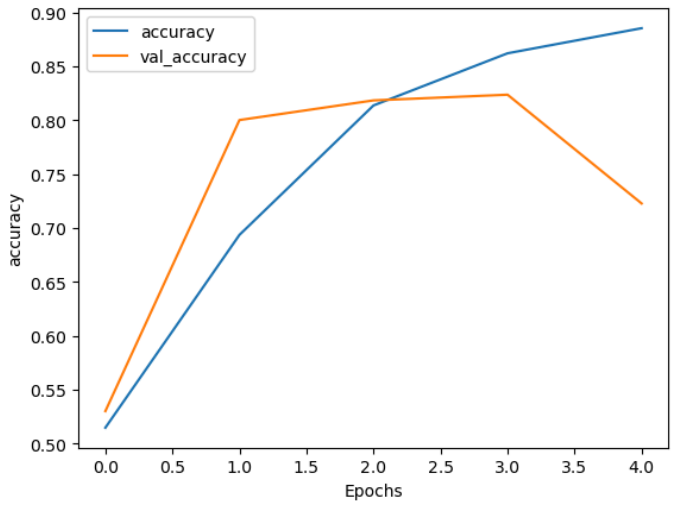
Total params: 161,909 (632.46 KB)
Trainable params: 161,909 (632.46 KB)
Non-trainable params: 0 (0.00 B)

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

```
num_epochs = 5
history = model.fit(padded,training_labels_final,epochs = num_epochs,validation_data = (testing_padded,testing_labels_final))
```

```
Epoch 1/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 12s 13ms/step - accuracy: 0.5114 - loss: 0.6939 - val_accuracy: 0.5300 - val_loss: 0.6878
Epoch 2/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 10s 13ms/step - accuracy: 0.6494 - loss: 0.6244 - val_accuracy: 0.8000 - val_loss: 0.4619
Epoch 3/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 10s 13ms/step - accuracy: 0.8098 - loss: 0.4430 - val_accuracy: 0.8182 - val_loss: 0.4119
Epoch 4/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 10s 13ms/step - accuracy: 0.8694 - loss: 0.3334 - val_accuracy: 0.8235 - val_loss: 0.4023
Epoch 5/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 10s 13ms/step - accuracy: 0.8934 - loss: 0.2858 - val_accuracy: 0.7226 - val_loss: 0.6954
```

```
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 = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim,
                              input_shape=(120,)),
    tf.keras.layers.Bidirectional(tf.keras.layers.GRU(32)),
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
model.summary()
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
embedding_4 (Embedding)	(None, 120, 16)	160,000
bidirectional (Bidirectional)	(None, 64)	9,600
dense_6 (Dense)	(None, 10)	650
dense_7 (Dense)	(None, 1)	11

Total params: 170,261 (665.08 KB)
Trainable params: 170,261 (665.08 KB)

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

```
num_epochs = 5
history = model.fit(padded,training_labels_final,epochs = num_epochs,validation_data = (testing_padded,testing_labels_final))
```

```
Epoch 1/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 26s 28ms/step - accuracy: 0.6201 - loss: 0.6234 - val_accuracy: 0.8330 - val_loss: 0.3744
Epoch 2/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 21s 27ms/step - accuracy: 0.8695 - loss: 0.3201 - val_accuracy: 0.8460 - val_loss: 0.3568
Epoch 3/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 22s 28ms/step - accuracy: 0.9147 - loss: 0.2266 - val_accuracy: 0.8359 - val_loss: 0.3941
Epoch 4/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 21s 27ms/step - accuracy: 0.9410 - loss: 0.1664 - val_accuracy: 0.8252 - val_loss: 0.4121
Epoch 5/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 21s 27ms/step - accuracy: 0.9595 - loss: 0.1177 - val_accuracy: 0.8178 - val_loss: 0.5202
```

Start coding or generate with AI.