In [ ]:

Open in Colab
(https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/TensorF
%20NLP/Course%203%20-%20Week%202%20-%20Lesson%201.ipynb)

In [1]:
```
import tensorflow as tf
print(tf.__version__)

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

```
2.3.0
```

In [2]:
```
import tensorflow_datasets as tfds
imdb, info = tfds.load("imdb_reviews", with_info=True, as_supervise
```

```
Downloading and preparing dataset imdb_reviews/plain_text/1.0.0 (dc
3 MiB, generated: Unknown size, total: 80.23 MiB) to /root/tensorfl
imdb_reviews/plain_text/1.0.0...

HBox(children=(FloatProgress(value=1.0, bar_style='info', descripti
eted...', max=1.0, style=Progre…

HBox(children=(FloatProgress(value=1.0, bar_style='info', descripti
e...', max=1.0, style=ProgressSty…




HBox(children=(FloatProgress(value=1.0, bar_style='info', max=1.0),
='')))

Shuffling and writing examples to /root/tensorflow_datasets/imdb_re
text/1.0.0.incompleteUIHLO6/imdb_reviews-train.tfrecord

HBox(children=(FloatProgress(value=0.0, max=25000.0), HTML(value=''

HBox(children=(FloatProgress(value=1.0, bar_style='info', max=1.0),
='')))

Shuffling and writing examples to /root/tensorflow_datasets/imdb_re
text/1.0.0.incompleteUIHLO6/imdb_reviews-test.tfrecord

HBox(children=(FloatProgress(value=0.0, max=25000.0), HTML(value=''

HBox(children=(FloatProgress(value=1.0, bar_style='info', max=1.0),
='')))

Shuffling and writing examples to /root/tensorflow_datasets/imdb_re
text/1.0.0.incompleteUIHLO6/imdb_reviews-unsupervised.tfrecord

HBox(children=(FloatProgress(value=0.0, max=50000.0), HTML(value=''

Dataset imdb_reviews downloaded and prepared to /root/tensorflow_da
reviews/plain_text/1.0.0. Subsequent calls will reuse this data.
```

In [3]:
```
import numpy as np

#separate the data into training and test sets
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()
#the data is returned as tensors in which we can iterate over
#so we iterate over em and convert em into numpy so that we can ada
#a list
#for sentence, label in train, test
for s,l in train_data:
  training_sentences.append(s.numpy().decode('utf8'))
  training_labels.append(l.numpy())

for s,l in test_data:
  testing_sentences.append(s.numpy().decode('utf8'))
  testing_labels.append(l.numpy())

#the data is then needed as numpy arrays
training_labels_final = np.array(training_labels)
testing_labels_final = np.array(testing_labels)

print(training_labels_final[10: 20])
```

```
[0 1 1 0 1 0 1 1 1 0]
[0 1 1 0 1 0 1 1 1 0]
```

In [ ]:
```
#globalise the variables so that we can easily edit em
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 object with out-of-vocab token
tokenizer = Tokenizer(num_words = vocab_size, oov_token=oov_tok)
#create the tokens for the words in the training array
tokenizer.fit_on_texts(training_sentences)
#create the dictionary
word_index = tokenizer.word_index
#tokenize all the training sentences and add em into a list
sequences = tokenizer.texts_to_sequences(training_sentences)
#pad for uniformity of length, with a max length and truncate the r
padded = pad_sequences(sequences,maxlen=max_length, truncating=trun

#convert the testing sentences into tokens based on the testing fit
#we dont fit the testing as well cuz then what even is the point
testing_sequences = tokenizer.texts_to_sequences(testing_sentences)
testing_padded = pad_sequences(testing_sequences,maxlen=max_length)

print(word_index)
print(padded[10])
print(testing_padded[1])
```

```
{'<OOV>': 1, 'the': 2, 'and': 3, 'a': 4, 'of': 5, 'to': 6, 'is': 7,
'in': 9, 'it': 10, 'i': 11, 'this': 12, 'that': 13, 'was': 14, 'as'
r': 16, 'with': 17, 'movie': 18, 'but': 19, 'film': 20, 'on': 21, '
'you': 23, 'are': 24, 'his': 25, 'have': 26, 'he': 27, 'be': 28, 'c
'all': 30, 'at': 31, 'by': 32, 'an': 33, 'they': 34, 'who': 35, 'so
om': 37, 'like': 38, 'her': 39, 'or': 40, 'just': 41, 'about': 42,
3, 'out': 44, 'if': 45, 'has': 46, 'some': 47, 'there': 48, 'what':
d': 50, 'more': 51, 'when': 52, 'very': 53, 'up': 54, 'no': 55, 'ti
'she': 57, 'even': 58, 'my': 59, 'would': 60, 'which': 61, 'only':
y': 63, 'really': 64, 'see': 65, 'their': 66, 'had': 67, 'can': 68,
9, 'me': 70, 'well': 71, 'than': 72, 'we': 73, 'much': 74, 'been':
76, 'get': 77, 'will': 78, 'do': 79, 'also': 80, 'into': 81, 'peopl
ther': 83, 'first': 84, 'great': 85, 'because': 86, 'how': 87, 'him
st': 89, "don't": 90, 'made': 91, 'its': 92, 'then': 93, 'way': 94,
5, 'them': 96, 'too': 97, 'could': 98, 'any': 99, 'movies': 100, 'a
1, 'think': 102, 'characters': 103, 'watch': 104, 'two': 105, 'film
'character': 107, 'seen': 108, 'many': 109, 'being': 110, 'life': 1
t': 112, 'never': 113, 'acting': 114, 'little': 115, 'best': 116, '
7, 'over': 118, 'where': 119, 'did': 120, 'show': 121, 'know': 122,
3, 'ever': 124, 'does': 125, 'better': 126, 'your': 127, 'end': 128
```

In [ ]:
```
#in order to create files for the visualization, we need to shift t
#and values of the vocab dictionary
reverse_word_index = dict([(value, key) for (key, value) in word_ir
print(reverse_word_index)
def decode_review(text):
    return ' '.join([reverse_word_index.get(i, '?') for i in text])

print(decode_review(padded[3]))
print(training_sentences[3])
```

```
 "alyson's", 54717: 'loafer', 54718: 'ama', 54719: 'vaccarro', 5472
ically', 54721: "rickles'", 54722: 'babson', 54723: 'leonidas', 547
a', 54725: 'coercible', 54726: "21's", 54727: "def's", 54728: "nwh'
"tap's", 54730: 'combusted', 54731: 'crispy', 54732: "fortier's", 5
oooo', 54734: 'schya', 54735: 'voudon', 54736: 'loki', 54737: 'vouc
8: 'dysantry', 54739: 'wonderously', 54740: 'pied', 54741: 'pipers'
 'sugimoto', 54743: 'crossings', 54744: 'himalaya', 54745: "emanuel
6: 'keeling', 54747: 'undeath', 54748: "''nice", 54749: "pair''", 5
heir", 54751: 'chessy', 54752: 'crudy', 54753: 'gayson', 54754: 'zc
5: 'headband', 54756: 'harchard', 54757: 'pseudolesbian', 54758: 'z
 54759: 'pillsbury', 54760: 'kornhauser', 54761: "tart'n'tangy", 54
alee's", 54763: 'gerri', 54764: 'viveca', 54765: 'lindfors', 54766:
y's", 54767: 'tatta', 54768: "wrestler's", 54769: 'corpse\x97the',
ntinas', 54771: 'sparklers', 54772: 'pessimist', 54773: 'bondian',
i'll", 54775: 'coilition', 54776: "'charming'", 54777: "'monster",
rte'", 54779: "'butterfield", 54780: "mates'", 54781: "holliman's",
 "'cookie", 54783: 'statuettes', 54784: 'frighteners', 54785: 'carv
86: 'bmovies', 54787: '65m', 54788: "5'5", 54789: 'stinger', 54790:
r's", 54791: "policemen's", 54792: 'striked', 54793: "lebrun's", 54
rate', 54795: 'denser', 54796: 'mclachlan', 54797: 'bettis', 54798:
```

In [ ]:
```
#same,except for addition of embedding layer
#the embedding layer is what does all the magic (create the high di
#vectors that're similar for words of similar sentiment, making the
#cluster together on both ends)
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_leng
    #the array returned is 2D so, just like for images, it requires
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(6, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=[
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding (Embedding)        (None, 120, 16)           160000    
_________________________________________________________________
flatten (Flatten)            (None, 1920)              0         
_________________________________________________________________
dense (Dense)                (None, 6)                 11526     
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 7         
=================================================================
Total params: 171,533
Trainable params: 171,533
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
num_epochs = 10
#input -> padded, training_labels_final
#validation on -> testing_padded, testing_labels_final
model.fit(padded, training_labels_final, epochs=num_epochs, validat
```

```
Epoch 1/10
782/782 [==============================] - 6s 8ms/step - loss: 0.50
y: 0.7358 - val_loss: 0.3403 - val_accuracy: 0.8520
Epoch 2/10
782/782 [==============================] - 6s 8ms/step - loss: 0.24
y: 0.9045 - val_loss: 0.3709 - val_accuracy: 0.8396
Epoch 3/10
782/782 [==============================] - 6s 8ms/step - loss: 0.09
y: 0.9760 - val_loss: 0.4476 - val_accuracy: 0.8308
Epoch 4/10
782/782 [==============================] - 6s 8ms/step - loss: 0.02
y: 0.9963 - val_loss: 0.5134 - val_accuracy: 0.8304
Epoch 5/10
782/782 [==============================] - 6s 8ms/step - loss: 0.01
y: 0.9984 - val_loss: 0.5962 - val_accuracy: 0.8280
Epoch 6/10
782/782 [==============================] - 6s 8ms/step - loss: 0.00
y: 0.9992 - val_loss: 0.6410 - val_accuracy: 0.8257
Epoch 7/10
782/782 [==============================] - 6s 8ms/step - loss: 0.00
y: 0.9998 - val_loss: 0.6904 - val_accuracy: 0.8250
Epoch 8/10
782/782 [==============================] - 6s 8ms/step - loss: 8.36
uracy: 1.0000 - val_loss: 0.7365 - val_accuracy: 0.8254
Epoch 9/10
782/782 [==============================] - 6s 8ms/step - loss: 3.21
uracy: 1.0000 - val_loss: 0.7734 - val_accuracy: 0.8266
Epoch 10/10
782/782 [==============================] - 6s 8ms/step - loss: 1.76
uracy: 1.0000 - val_loss: 0.8071 - val_accuracy: 0.8272
```

Out[9]:
```
<tensorflow.python.keras.callbacks.History at 0x7f904ec95748>
```

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

```
(10000, 16)
```

In [ ]:
```
import io

#write the data onto files
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, vocab_size):
  word = reverse_word_index[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()
```

In [ ]:
```
try:
  from google.colab import files
except ImportError:
  pass
else:
  files.download('vecs.tsv')
  files.download('meta.tsv')
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

In [ ]:
```

sentence = "I really think this is amazing. honest."
sequence = tokenizer.texts_to_sequences([sentence])
print(sequence)
```

```
[[11, 64, 102, 12, 7, 478, 1200]]
```