In [ ]:
```
#@title Licensed under the Apache License, Version 2.0 (the "Licens
# 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, softwa
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or i
# See the License for the specific language governing permissions a
# limitations under the License.
```

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

In [6]:
```
# Run this to ensure TensorFlow 2.x is used
try:
  # %tensorflow_version only exists in Colab.
  %tensorflow_version 2.x
except Exception:
  pass
```

In [7]:
```
import json
import tensorflow as tf

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

In [8]:
```
vocab_size = 10000
embedding_dim = 16
max_length = 100
trunc_type='post'
padding_type='post'
oov_tok = "<OOV>"
training_size = 20000
```

In [9]:
```
!wget --no-check-certificate \
    https://storage.googleapis.com/laurencemoroney-blog.appspot.com
    -O /tmp/sarcasm.json
```

```
'wget' is not recognized as an internal or external command,
operable program or batch file.
```

In [5]:
```
#tokenize sarcarm
with open("/tmp/sarcasm.json", 'r') as f:
    datastore = json.load(f)

sentences = []
labels = []
#separate the json file
#each item is an object, so it's really easy to get data
for item in datastore:
    sentences.append(item['headline'])
    labels.append(item['is_sarcastic'])
```

In [6]:
```
#separate into testing and training by slicing at indexes
training_sentences = sentences[0:training_size]
testing_sentences = sentences[training_size:]
training_labels = labels[0:training_size]
testing_labels = labels[training_size:]
```

In [7]:
```
tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(training_sentences)

word_index = tokenizer.word_index

training_sequences = tokenizer.texts_to_sequences(training_sentence
training_padded = pad_sequences(training_sequences, maxlen=max_leng
#same process, obtain tokenized diciontary of training sentences
#then try it on the testing sentences

testing_sequences = tokenizer.texts_to_sequences(testing_sentences)
testing_padded = pad_sequences(testing_sequences, maxlen=max_length
```

In [8]:
```
# Need this block to get it to work with TensorFlow 2.x
import numpy as np
training_padded = np.array(training_padded)
training_labels = np.array(training_labels)
testing_padded = np.array(testing_padded)
testing_labels = np.array(testing_labels)
```

In [9]:
```
#the vector shape returned is not easily flattened, so the
#global average pooling layer is used
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_leng
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(24, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=[
```

In [10]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding (Embedding)        (None, 100, 16)           160000    
_________________________________________________________________
global_average_pooling1d (Gl (None, 16)                0         
_________________________________________________________________
dense (Dense)                (None, 24)                408       
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 25        
=================================================================
Total params: 160,433
Trainable params: 160,433
Non-trainable params: 0
_________________________________________________________________
```

In [11]:
```
num_epochs = 30
history = model.fit(training_padded, training_labels, epochs=num_ep
```

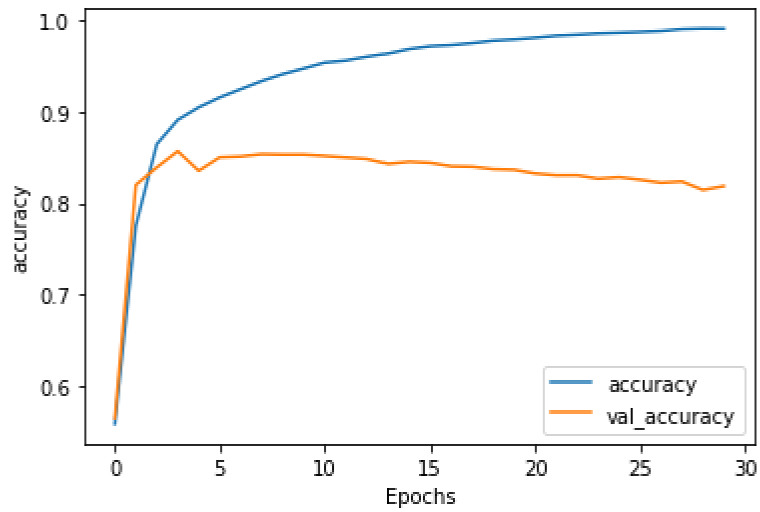
```
Epoch 1/30
625/625 - 2s - loss: 0.6803 - accuracy: 0.5584 - val_loss: 0.6548 -
y: 0.5646
Epoch 2/30
625/625 - 2s - loss: 0.5135 - accuracy: 0.7754 - val_loss: 0.4231 -
y: 0.8201
Epoch 3/30
625/625 - 2s - loss: 0.3404 - accuracy: 0.8648 - val_loss: 0.3699 -
y: 0.8393
Epoch 4/30
625/625 - 2s - loss: 0.2822 - accuracy: 0.8913 - val_loss: 0.3457 -
y: 0.8571
Epoch 5/30
625/625 - 2s - loss: 0.2447 - accuracy: 0.9051 - val_loss: 0.3641 -
y: 0.8356
Epoch 6/30
625/625 - 2s - loss: 0.2182 - accuracy: 0.9159 - val_loss: 0.3524 -
y: 0.8504
Epoch 7/30
625/625 - 2s - loss: 0.1947 - accuracy: 0.9249 - val_loss: 0.3497 -
y: 0.8512
Epoch 8/30
625/625 - 2s - loss: 0.1764 - accuracy: 0.9338 - val_loss: 0.3661 -
y: 0.8538
Epoch 9/30
625/625 - 2s - loss: 0.1586 - accuracy: 0.9413 - val_loss: 0.3682 -
y: 0.8533
Epoch 10/30
625/625 - 2s - loss: 0.1462 - accuracy: 0.9475 - val_loss: 0.3833 -
y: 0.8533
Epoch 11/30
625/625 - 2s - loss: 0.1328 - accuracy: 0.9540 - val_loss: 0.4007 -
y: 0.8518
Epoch 12/30
625/625 - 3s - loss: 0.1234 - accuracy: 0.9563 - val_loss: 0.4147 -
y: 0.8502
Epoch 13/30
625/625 - 2s - loss: 0.1138 - accuracy: 0.9604 - val_loss: 0.4334 -
y: 0.8486
Epoch 14/30
625/625 - 2s - loss: 0.1054 - accuracy: 0.9639 - val_loss: 0.4548 -
y: 0.8432
Epoch 15/30
625/625 - 2s - loss: 0.0955 - accuracy: 0.9689 - val_loss: 0.4701 -
y: 0.8454
Epoch 16/30
625/625 - 2s - loss: 0.0876 - accuracy: 0.9720 - val_loss: 0.4956 -
y: 0.8442
Epoch 17/30
625/625 - 2s - loss: 0.0831 - accuracy: 0.9732 - val_loss: 0.5174 -
y: 0.8405
Epoch 18/30
625/625 - 2s - loss: 0.0764 - accuracy: 0.9754 - val_loss: 0.5421 -
```

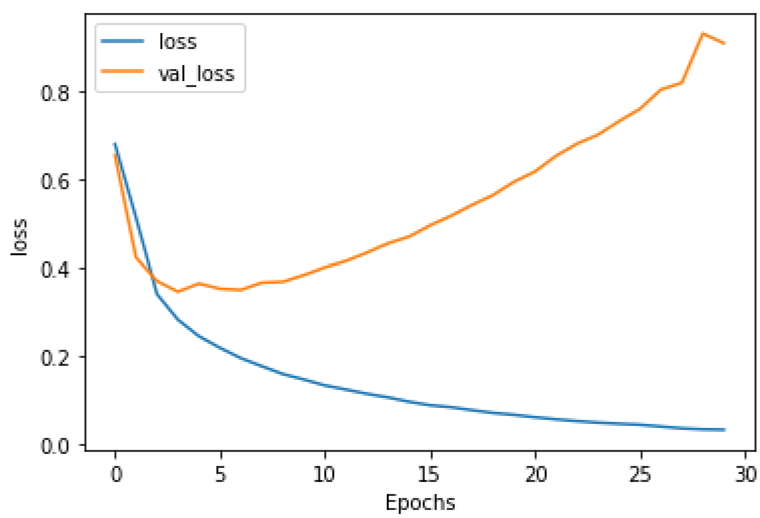
```
y: 0.8401
Epoch 19/30
625/625 - 2s - loss: 0.0702 - accuracy: 0.9780 - val_loss: 0.5645 -
y: 0.8374
Epoch 20/30
625/625 - 2s - loss: 0.0659 - accuracy: 0.9793 - val_loss: 0.5946 -
y: 0.8366
Epoch 21/30
625/625 - 2s - loss: 0.0603 - accuracy: 0.9812 - val_loss: 0.6179 -
y: 0.8326
Epoch 22/30
625/625 - 2s - loss: 0.0556 - accuracy: 0.9834 - val_loss: 0.6535 -
y: 0.8307
Epoch 23/30
625/625 - 2s - loss: 0.0516 - accuracy: 0.9845 - val_loss: 0.6811 -
y: 0.8307
Epoch 24/30
625/625 - 2s - loss: 0.0481 - accuracy: 0.9859 - val_loss: 0.7016 -
y: 0.8269
Epoch 25/30
625/625 - 2s - loss: 0.0455 - accuracy: 0.9867 - val_loss: 0.7322 -
y: 0.8286
Epoch 26/30
625/625 - 2s - loss: 0.0434 - accuracy: 0.9876 - val_loss: 0.7603 -
y: 0.8258
Epoch 27/30
625/625 - 2s - loss: 0.0392 - accuracy: 0.9886 - val_loss: 0.8041 -
y: 0.8225
Epoch 28/30
625/625 - 2s - loss: 0.0351 - accuracy: 0.9906 - val_loss: 0.8197 -
y: 0.8238
Epoch 29/30
625/625 - 3s - loss: 0.0328 - accuracy: 0.9913 - val_loss: 0.9317 -
y: 0.8146
Epoch 30/30
625/625 - 3s - loss: 0.0319 - accuracy: 0.9912 - val_loss: 0.9097 -
y: 0.8188
```

In [12]:
```
import matplotlib.pyplot as plt
#overtime the loss increases, due to overfitting

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")
```





In [13]:
```
reverse_word_index = dict([(value, key) for (key, value) in word_ir

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

print(decode_sentence(training_padded[0]))
print(training_sentences[2])
print(labels[2])
```

```
former <OOV> store clerk sues over secret 'black <OOV> for minority
? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ?
mom starting to fear son's web series closest thing she will have t
1
```

In [14]:
```
e = model.layers[0]
weights = e.get_weights()[0]
print(weights.shape) # shape: (vocab_size, embedding_dim)
```

```
(10000, 16)
```

In [15]:
```
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, 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 [16]:
```
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 [17]:
```
sentence = ["granny starting to fear spiders in the garden might be
sequences = tokenizer.texts_to_sequences(sentence)
padded = pad_sequences(sequences, maxlen=max_length, padding=paddir
print(model.predict(padded))
```

```
[[9.844283e-01]
 [8.191375e-04]]
```