In [1]:

```
#@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.
```

Open in Colab

In [2]:

```
import json
import tensorflow as tf
import csv
import random
import numpy as np

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
from tensorflow.keras import regularizers


embedding_dim = 100
max_length = 16
trunc_type='post'
padding_type='post'
oov_tok = "<OOV>"
training_size=160000
test_portion=.1

corpus = []
```

In [3]:

```
# Note that I cleaned the Stanford dataset to remove LATIN1 encoding to make it easier for Python
CSV reader
# You can do that yourself with:
# iconv -f LATIN1 -t UTF8 training.1600000.processed.noemoticon.csv -o training_cleaned.csv
# I then hosted it on my site to make it easier to use in this notebook

!wget --no-check-certificate \
    https://storage.googleapis.com/laurencemoroney-blog.appspot.com/training_cleaned.csv \
    -O /tmp/training_cleaned.csv

num_sentences = 0

with open("/tmp/training_cleaned.csv") as csvfile:
    reader = csv.reader(csvfile, delimiter=',')
    for row in reader:
        list_item=[]
        list_item.append(row[5])
        this_label=row[0]
        if this_label=='0':
            list_item.append(0)
        else:
            list_item.append(1)
        num_sentences = num_sentences + 1
        corpus.append(list_item)
```

```
--2020-10-06 12:38:32--  https://storage.googleapis.com/laurencemoroney-
blog.appspot.com/training_cleaned.csv
Resolving storage.googleapis.com (storage.googleapis.com)... 74.125.31.128, 173.194.216.128, 173.1
```

```
94.217.128, ...
Connecting to storage.googleapis.com (storage.googleapis.com)|74.125.31.128|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 238942690 (228M) [application/octet-stream]
Saving to: '/tmp/training_cleaned.csv'

/tmp/training_clean 100%[===================>] 227.87M   171MB/s    in 1.3s

2020-10-06 12:38:33 (171 MB/s) - '/tmp/training_cleaned.csv' saved [238942690/238942690]
```

In [4]:

```
print(num_sentences)
print(len(corpus))
print(corpus[1])

# Expected Output:
# 1600000
# 1600000
# ["is upset that he can't update his Facebook by texting it... and might cry as a result  School today also. Blah!", 0]
```

```
1600000
1600000
["is upset that he can't update his Facebook by texting it... and might cry as a result  School to
day also. Blah!", 0]
```

In [5]:

```
sentences=[]
labels=[]
random.shuffle(corpus)
for x in range(training_size):
    sentences.append(corpus[x][0])
    labels.append(corpus[x][1])


tokenizer = Tokenizer()
tokenizer.fit_on_texts(sentences)

word_index = tokenizer.word_index
vocab_size=len(word_index)

sequences = tokenizer.texts_to_sequences(sentences)
padded = pad_sequences(sequences, maxlen=max_length, padding=padding_type, truncating=trunc_type)

split = int(test_portion * training_size)

test_sequences = padded[0:split]
training_sequences = padded[split:training_size]
test_labels = labels[0:split]
training_labels = labels[split:training_size]
```

In [6]:

```
print(vocab_size)
print(word_index['i'])
# Expected Output
# 138858
# 1
```

```
138455
1
```

In [7]:

```
# Note this is the 100 dimension version of GloVe from Stanford
# I unzipped and hosted it on my site to make this notebook easier
!wget --no-check-certificate \
    https://storage.googleapis.com/laurencemoroney-blog.appspot.com/glove.6B.100d.txt \
```

```
    -O /tmp/glove.6B.100d.txt
embeddings_index = {};
with open('/tmp/glove.6B.100d.txt') as f:
    for line in f:
        values = line.split();
        word = values[0];
        coefs = np.asarray(values[1:], dtype='float32');
        embeddings_index[word] = coefs;

embeddings_matrix = np.zeros((vocab_size+1, embedding_dim));
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word);
    if embedding_vector is not None:
        embeddings_matrix[i] = embedding_vector;
```

```
--2020-10-06 12:38:47--  https://storage.googleapis.com/laurencemoroney-blog.appspot.com/glove.6B.100d.txt
Resolving storage.googleapis.com (storage.googleapis.com)... 142.250.98.128, 173.194.214.128, 172.217.204.128, ...
Connecting to storage.googleapis.com (storage.googleapis.com)|142.250.98.128|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 347116733 (331M) [text/plain]
Saving to: '/tmp/glove.6B.100d.txt'

/tmp/glove.6B.100d. 100%[===================>] 331.04M   164MB/s    in 2.0s

2020-10-06 12:38:49 (164 MB/s) - '/tmp/glove.6B.100d.txt' saved [347116733/347116733]
```

In [8]:

```
print(len(embeddings_matrix))
# Expected Output
# 138859
```

```
138456
```

In [9]:

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size+1, embedding_dim, input_length=max_length, weights=[embeddings_matrix], trainable=False),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Conv1D(64, 5, activation='relu'),
    tf.keras.layers.MaxPooling1D(pool_size=4),
    tf.keras.layers.LSTM(64),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
model.summary()

num_epochs = 50

training_padded = np.array(training_sequences)
training_labels = np.array(training_labels)
testing_padded = np.array(test_sequences)
testing_labels = np.array(test_labels)

history = model.fit(training_padded, training_labels, epochs=num_epochs, validation_data=(testing_padded, testing_labels), verbose=2)

print("Training Complete")
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 16, 100)           13845600
_________________________________________________________________
dropout (Dropout)            (None, 16, 100)           0
_________________________________________________________________
conv1d (Conv1D)              (None, 12, 64)            32064
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 3, 64)             0
```

```
max_pooling1d (MaxPooling1D) (None, 3, 64)              0
_________________________________________________________________
lstm (LSTM)                  (None, 64)                33024
_________________________________________________________________
dense (Dense)                (None, 1)                 65
=================================================================
Total params: 13,910,753
Trainable params: 65,153
Non-trainable params: 13,845,600
_________________________________________________________________
Epoch 1/50
4500/4500 - 20s - loss: 0.5671 - accuracy: 0.6979 - val_loss: 0.5261 - val_accuracy: 0.7344
Epoch 2/50
4500/4500 - 19s - loss: 0.5271 - accuracy: 0.7323 - val_loss: 0.5140 - val_accuracy: 0.7394
Epoch 3/50
4500/4500 - 19s - loss: 0.5106 - accuracy: 0.7430 - val_loss: 0.5122 - val_accuracy: 0.7458
Epoch 4/50
4500/4500 - 19s - loss: 0.4975 - accuracy: 0.7531 - val_loss: 0.5237 - val_accuracy: 0.7439
Epoch 5/50
4500/4500 - 20s - loss: 0.4896 - accuracy: 0.7580 - val_loss: 0.5010 - val_accuracy: 0.7504
Epoch 6/50
4500/4500 - 20s - loss: 0.4824 - accuracy: 0.7635 - val_loss: 0.5018 - val_accuracy: 0.7543
Epoch 7/50
4500/4500 - 19s - loss: 0.4770 - accuracy: 0.7660 - val_loss: 0.4978 - val_accuracy: 0.7563
Epoch 8/50
4500/4500 - 20s - loss: 0.4712 - accuracy: 0.7695 - val_loss: 0.5004 - val_accuracy: 0.7541
Epoch 9/50
4500/4500 - 19s - loss: 0.4661 - accuracy: 0.7725 - val_loss: 0.4986 - val_accuracy: 0.7527
Epoch 10/50
4500/4500 - 19s - loss: 0.4626 - accuracy: 0.7751 - val_loss: 0.4979 - val_accuracy: 0.7575
Epoch 11/50
4500/4500 - 19s - loss: 0.4592 - accuracy: 0.7780 - val_loss: 0.5012 - val_accuracy: 0.7542
Epoch 12/50
4500/4500 - 19s - loss: 0.4556 - accuracy: 0.7802 - val_loss: 0.5053 - val_accuracy: 0.7547
Epoch 13/50
4500/4500 - 19s - loss: 0.4533 - accuracy: 0.7800 - val_loss: 0.5013 - val_accuracy: 0.7556
Epoch 14/50
4500/4500 - 19s - loss: 0.4510 - accuracy: 0.7817 - val_loss: 0.5046 - val_accuracy: 0.7521
Epoch 15/50
4500/4500 - 19s - loss: 0.4484 - accuracy: 0.7838 - val_loss: 0.5100 - val_accuracy: 0.7531
Epoch 16/50
4500/4500 - 19s - loss: 0.4484 - accuracy: 0.7832 - val_loss: 0.5104 - val_accuracy: 0.7524
Epoch 17/50
4500/4500 - 19s - loss: 0.4448 - accuracy: 0.7859 - val_loss: 0.5046 - val_accuracy: 0.7562
Epoch 18/50
4500/4500 - 19s - loss: 0.4443 - accuracy: 0.7871 - val_loss: 0.5071 - val_accuracy: 0.7544
Epoch 19/50
4500/4500 - 19s - loss: 0.4422 - accuracy: 0.7871 - val_loss: 0.5151 - val_accuracy: 0.7555
Epoch 20/50
4500/4500 - 19s - loss: 0.4401 - accuracy: 0.7889 - val_loss: 0.5139 - val_accuracy: 0.7503
Epoch 21/50
4500/4500 - 20s - loss: 0.4397 - accuracy: 0.7893 - val_loss: 0.5190 - val_accuracy: 0.7505
Epoch 22/50
4500/4500 - 20s - loss: 0.4377 - accuracy: 0.7906 - val_loss: 0.5167 - val_accuracy: 0.7487
Epoch 23/50
4500/4500 - 20s - loss: 0.4379 - accuracy: 0.7900 - val_loss: 0.5126 - val_accuracy: 0.7551
Epoch 24/50
4500/4500 - 20s - loss: 0.4360 - accuracy: 0.7914 - val_loss: 0.5135 - val_accuracy: 0.7525
Epoch 25/50
4500/4500 - 19s - loss: 0.4368 - accuracy: 0.7916 - val_loss: 0.5115 - val_accuracy: 0.7522
Epoch 26/50
4500/4500 - 20s - loss: 0.4353 - accuracy: 0.7918 - val_loss: 0.5154 - val_accuracy: 0.7526
Epoch 27/50
4500/4500 - 19s - loss: 0.4349 - accuracy: 0.7928 - val_loss: 0.5150 - val_accuracy: 0.7483
Epoch 28/50
4500/4500 - 19s - loss: 0.4333 - accuracy: 0.7930 - val_loss: 0.5146 - val_accuracy: 0.7494
Epoch 29/50
4500/4500 - 19s - loss: 0.4327 - accuracy: 0.7930 - val_loss: 0.5107 - val_accuracy: 0.7508
Epoch 30/50
4500/4500 - 19s - loss: 0.4307 - accuracy: 0.7946 - val_loss: 0.5170 - val_accuracy: 0.7512
Epoch 31/50
4500/4500 - 19s - loss: 0.4316 - accuracy: 0.7945 - val_loss: 0.5111 - val_accuracy: 0.7511
Epoch 32/50
4500/4500 - 19s - loss: 0.4309 - accuracy: 0.7948 - val_loss: 0.5108 - val_accuracy: 0.7521
Epoch 33/50
4500/4500 - 19s - loss: 0.4301 - accuracy: 0.7954 - val_loss: 0.5262 - val_accuracy: 0.7499
Epoch 34/50
4500/4500 - 19s - loss: 0.4287 - accuracy: 0.7961 - val_loss: 0.5164 - val_accuracy: 0.7511
```

```
4500/4500 - 19s - loss: 0.4287 - accuracy: 0.7961 - val_loss: 0.5164 - val_accuracy: 0.7511
Epoch 35/50
4500/4500 - 19s - loss: 0.4280 - accuracy: 0.7975 - val_loss: 0.5164 - val_accuracy: 0.7530
Epoch 36/50
4500/4500 - 19s - loss: 0.4287 - accuracy: 0.7952 - val_loss: 0.5165 - val_accuracy: 0.7513
Epoch 37/50
4500/4500 - 20s - loss: 0.4293 - accuracy: 0.7955 - val_loss: 0.5173 - val_accuracy: 0.7524
Epoch 38/50
4500/4500 - 20s - loss: 0.4278 - accuracy: 0.7966 - val_loss: 0.5181 - val_accuracy: 0.7510
Epoch 39/50
4500/4500 - 20s - loss: 0.4262 - accuracy: 0.7969 - val_loss: 0.5232 - val_accuracy: 0.7491
Epoch 40/50
4500/4500 - 19s - loss: 0.4277 - accuracy: 0.7983 - val_loss: 0.5202 - val_accuracy: 0.7479
Epoch 41/50
4500/4500 - 19s - loss: 0.4275 - accuracy: 0.7972 - val_loss: 0.5201 - val_accuracy: 0.7515
Epoch 42/50
4500/4500 - 19s - loss: 0.4258 - accuracy: 0.7982 - val_loss: 0.5224 - val_accuracy: 0.7472
Epoch 43/50
4500/4500 - 19s - loss: 0.4256 - accuracy: 0.7972 - val_loss: 0.5221 - val_accuracy: 0.7527
Epoch 44/50
4500/4500 - 19s - loss: 0.4264 - accuracy: 0.7973 - val_loss: 0.5136 - val_accuracy: 0.7492
Epoch 45/50
4500/4500 - 19s - loss: 0.4254 - accuracy: 0.7976 - val_loss: 0.5222 - val_accuracy: 0.7548
Epoch 46/50
4500/4500 - 19s - loss: 0.4253 - accuracy: 0.7981 - val_loss: 0.5239 - val_accuracy: 0.7487
Epoch 47/50
4500/4500 - 19s - loss: 0.4247 - accuracy: 0.7978 - val_loss: 0.5234 - val_accuracy: 0.7504
Epoch 48/50
4500/4500 - 19s - loss: 0.4264 - accuracy: 0.7973 - val_loss: 0.5248 - val_accuracy: 0.7497
Epoch 49/50
4500/4500 - 19s - loss: 0.4250 - accuracy: 0.7979 - val_loss: 0.5278 - val_accuracy: 0.7501
Epoch 50/50
4500/4500 - 19s - loss: 0.4254 - accuracy: 0.7973 - val_loss: 0.5221 - val_accuracy: 0.7476
Training Complete
```

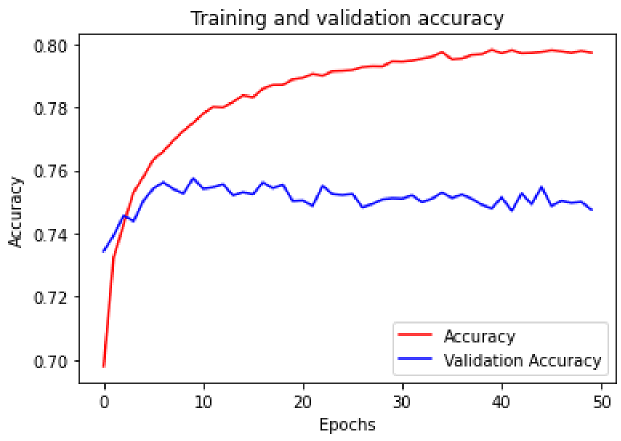
In [10]:

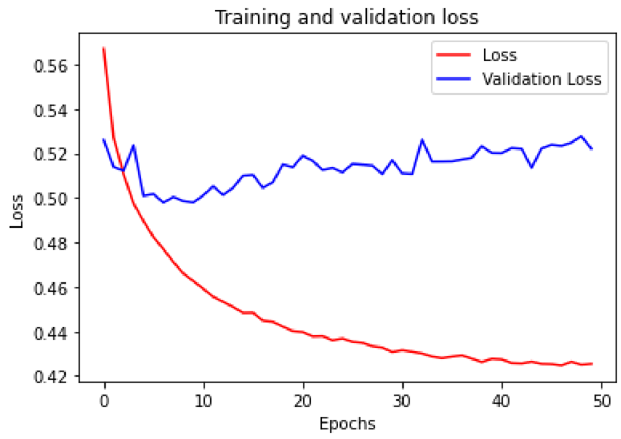
```
import matplotlib.image  as mpimg
import matplotlib.pyplot as plt

#-----------------------------------------------------------
# Retrieve a list of list results on training and test data
# sets for each training epoch
#-----------------------------------------------------------
acc=history.history['accuracy']
val_acc=history.history['val_accuracy']
loss=history.history['loss']
val_loss=history.history['val_loss']

epochs=range(len(acc)) # Get number of epochs

#------------------------------------------------
# Plot training and validation accuracy per epoch
#------------------------------------------------
plt.plot(epochs, acc, 'r')
plt.plot(epochs, val_acc, 'b')
plt.title('Training and validation accuracy')
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend(["Accuracy", "Validation Accuracy"])

plt.figure()

#------------------------------------------------
# Plot training and validation loss per epoch
#------------------------------------------------
plt.plot(epochs, loss, 'r')
plt.plot(epochs, val_loss, 'b')
plt.title('Training and validation loss')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend(["Loss", "Validation Loss"])

plt.figure()


# Expected Output
```

```
# Expected Output
# A chart where the validation loss does not increase sharply!
```

Out[10]:

```
<Figure size 432x288 with 0 Axes>
```






```
<Figure size 432x288 with 0 Axes>
```

In [ ]: