```
#@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");

```
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 = []



# Note that I cleaned the Stanford dataset to remove LATIN1 encoding to make it easier for Py
# 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:
```

```
    # Your Code here. Create list items where the first item is the text, found in row[5],
    # your label to be 0, otherwise 1. Keep a count of the number of sentences in num_sente
      list_item=[]
      list_item.append(row[5])
      label = row[0]
      if label=='0':
        list_item.append(0)
      else:
        list_item.append(1)
      num_sentences = num_sentences + 1
      corpus.append(list_item)
```

```
--2020-09-20 11:21:43--  https://storage.googleapis.com/laurencemoroney-blog.appspot.com
Resolving storage.googleapis.com (storage.googleapis.com)... 108.177.126.128, 108.177.12
Connecting to storage.googleapis.com (storage.googleapis.com)|108.177.126.128|:443... co
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  82.4MB/s    in 2.8s

2020-09-20 11:21:46 (82.4 MB/s) - '/tmp/training_cleaned.csv' saved [238942690/238942690
```

```
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  Sc
```

```
1600000
1600000
["is upset that he can't update his Facebook by texting it... and might cry as a result
```

```
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_t

split = int(test_portion * training_size)

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


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

```
138589
1
```

```
# 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;
```

```
--2020-09-20 11:22:19--  https://storage.googleapis.com/laurencemoroney-blog.appspot.com
Resolving storage.googleapis.com (storage.googleapis.com)... 108.177.126.128, 108.177.12
Connecting to storage.googleapis.com (storage.googleapis.com)|108.177.126.128|:443... co
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  75.5MB/s    in 4.4s

2020-09-20 11:22:23 (75.5 MB/s) - ‘/tmp/glove.6B.100d.txt’ saved [347116733/347116733]
```

```
print(len(embeddings_matrix))
```

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

```
138590
```

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size+1, embedding_dim, input_length=max_length, weights=[
    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(512, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')

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

num_epochs = 50
#remember, necessary to convert it into a numpy array
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=(tes

print("Training Complete")
```

```
4500/4500 - 37s - loss: 0.1182 - accuracy: 0.9517 - val_loss: 1.3060 - val_accuracy: 0.7
Epoch 23/50
4500/4500 - 37s - loss: 0.1122 - accuracy: 0.9545 - val_loss: 1.4461 - val_accuracy: 0.7
Epoch 24/50
4500/4500 - 37s - loss: 0.1079 - accuracy: 0.9558 - val_loss: 1.4563 - val_accuracy: 0.7
Epoch 25/50
4500/4500 - 37s - loss: 0.1034 - accuracy: 0.9582 - val_loss: 1.5059 - val_accuracy: 0.7
Epoch 26/50
4500/4500 - 37s - loss: 0.0990 - accuracy: 0.9604 - val_loss: 1.5365 - val_accuracy: 0.7
Epoch 27/50
4500/4500 - 37s - loss: 0.0963 - accuracy: 0.9617 - val_loss: 1.4236 - val_accuracy: 0.7
Epoch 28/50
4500/4500 - 37s - loss: 0.0900 - accuracy: 0.9644 - val_loss: 1.6182 - val_accuracy: 0.7
Epoch 29/50
4500/4500 - 37s - loss: 0.0891 - accuracy: 0.9647 - val_loss: 1.5838 - val_accuracy: 0.7
Epoch 30/50
4500/4500 - 37s - loss: 0.0855 - accuracy: 0.9663 - val_loss: 1.6543 - val_accuracy: 0.7
Epoch 31/50
4500/4500 - 37s - loss: 0.0831 - accuracy: 0.9679 - val_loss: 1.5713 - val_accuracy: 0.7
Epoch 32/50
4500/4500 - 37s - loss: 0.0804 - accuracy: 0.9681 - val_loss: 1.6991 - val_accuracy: 0.7
Epoch 33/50
4500/4500 - 37s - loss: 0.0780 - accuracy: 0.9694 - val_loss: 1.6883 - val_accuracy: 0.7
Epoch 34/50
4500/4500 - 36s - loss: 0.0773 - accuracy: 0.9707 - val_loss: 1.7188 - val_accuracy: 0.7
Epoch 35/50
4500/4500 - 36s - loss: 0.0749 - accuracy: 0.9711 - val_loss: 1.6867 - val_accuracy: 0.7
Epoch 36/50
4500/4500 - 36s - loss: 0.0731 - accuracy: 0.9724 - val_loss: 1.7900 - val_accuracy: 0.7
Epoch 37/50
4500/4500 - 36s - loss: 0.0726 - accuracy: 0.9719 - val_loss: 1.7484 - val_accuracy: 0.7
Epoch 38/50
4500/4500 - 36s - loss: 0.0707 - accuracy: 0.9726 - val_loss: 1.7046 - val_accuracy: 0.7
Epoch 39/50
4500/4500 - 37s - loss: 0.0680 - accuracy: 0.9740 - val_loss: 1.7999 - val_accuracy: 0.7
Epoch 40/50
4500/4500 - 36s - loss: 0.0681 - accuracy: 0.9739 - val_loss: 1.7551 - val_accuracy: 0.7
Epoch 41/50
4500/4500 - 36s - loss: 0.0669 - accuracy: 0.9744 - val_loss: 1.7741 - val_accuracy: 0.7
Epoch 42/50
4500/4500 - 36s - loss: 0.0639 - accuracy: 0.9759 - val_loss: 1.8474 - val_accuracy: 0.7
Epoch 43/50
4500/4500 - 36s - loss: 0.0634 - accuracy: 0.9761 - val_loss: 1.8055 - val_accuracy: 0.7
Epoch 44/50
4500/4500 - 36s - loss: 0.0631 - accuracy: 0.9763 - val_loss: 1.8305 - val_accuracy: 0.7
Epoch 45/50
4500/4500 - 36s - loss: 0.0619 - accuracy: 0.9758 - val_loss: 1.8472 - val_accuracy: 0.7
Epoch 46/50
4500/4500 - 36s - loss: 0.0615 - accuracy: 0.9763 - val_loss: 1.9773 - val_accuracy: 0.7
Epoch 47/50
4500/4500 - 37s - loss: 0.0613 - accuracy: 0.9770 - val_loss: 1.7948 - val_accuracy: 0.7
Epoch 48/50
4500/4500 - 36s - loss: 0.0593 - accuracy: 0.9769 - val_loss: 1.8398 - val_accuracy: 0.7
Epoch 49/50
4500/4500 - 36s - loss: 0.0589 - accuracy: 0.9778 - val_loss: 1.8199 - val_accuracy: 0.7
Epoch 50/50
4500/4500 - 36s - loss: 0.0587 - accuracy: 0.9778 - val_loss: 1.8801 - val_accuracy: 0.7
Training Complete
```

```
training complete
```

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