```
#@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.


from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout, Bidirectional
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
### YOUR CODE HERE
from tensorflow.keras import regularizers
###
import tensorflow.keras.utils as ku
import numpy as np


tokenizer = Tokenizer()
!wget --no-check-certificate \
    https://storage.googleapis.com/laurencemoroney-blog.appspot.com/sonnets.txt \
    -O /tmp/sonnets.txt
data = open('/tmp/sonnets.txt').read()

corpus = data.lower().split("\n")


tokenizer.fit_on_texts(corpus)
total_words = len(tokenizer.word_index) + 1

# create input sequences using list of tokens
input_sequences = []
for line in corpus:
  token_list = tokenizer.texts_to_sequences([line])[0]
  for i in range(1, len(token_list)):
    n_gram_sequence = token_list[:i+1]
    input_sequences.append(n_gram_sequence)


# pad sequences
max_sequence_len = max([len(x) for x in input_sequences])
input_sequences = np.array(pad_sequences(input_sequences, maxlen=max_sequence_len, padding='p

# create predictors and label
```

Licensed under the Apache License, Version 2.0 (the "License");

```
predictors, label = input_sequences[:,:-1],input_sequences[:,-1]

label = ku.to_categorical(label, num_classes=total_words)
```

```
--2020-09-20 17:58:08--  https://storage.googleapis.com/laurencemoroney-blog.appspot.com
Resolving storage.googleapis.com (storage.googleapis.com)... 74.125.124.128, 172.217.212
Connecting to storage.googleapis.com (storage.googleapis.com)|74.125.124.128|:443... con
HTTP request sent, awaiting response... 200 OK
Length: 93578 (91K) [text/plain]
Saving to: ‘/tmp/sonnets.txt’

/tmp/sonnets.txt    100%[===================>]  91.38K  --.-KB/s    in 0.001s

2020-09-20 17:58:08 (108 MB/s) - ‘/tmp/sonnets.txt’ saved [93578/93578]
```

```
model = Sequential()
model.add(Embedding(total_words, 100, input_length=max_sequence_len-1))
model.add(Bidirectional(LSTM(150, return_sequences=True)))
model.add(Dropout(0.2))
model.add(Bidirectional(LSTM(100)))
model.add(Dense(total_words/2, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model.add(Dense(total_words, activation='softmax'))
# Pick an optimizer
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 10, 100)           321100
_________________________________________________________________
bidirectional (Bidirectional (None, 10, 300)           301200
_________________________________________________________________
dropout (Dropout)            (None, 10, 300)           0
_________________________________________________________________
bidirectional_1 (Bidirection (None, 200)               320800
_________________________________________________________________
dense (Dense)                (None, 1605)              322605
_________________________________________________________________
dense_1 (Dense)              (None, 3211)              5156866
=================================================================
Total params: 6,422,571
Trainable params: 6,422,571
Non-trainable params: 0
_________________________________________________________________
None
```

```
history = model.fit(predictors, label, epochs=100, verbose=1)
```

```
Epoch 72/100
484/484 [==============================] - 7s 14ms/step - loss: 1.3996 - accuracy: 0.763
Epoch 73/100
484/484 [==============================] - 7s 14ms/step - loss: 1.3815 - accuracy: 0.768
Epoch 74/100
484/484 [==============================] - 7s 14ms/step - loss: 1.3648 - accuracy: 0.776
Epoch 75/100
484/484 [==============================] - 7s 14ms/step - loss: 1.3311 - accuracy: 0.777
Epoch 76/100
484/484 [==============================] - 7s 14ms/step - loss: 1.2987 - accuracy: 0.782
Epoch 77/100
484/484 [==============================] - 7s 14ms/step - loss: 1.2998 - accuracy: 0.780
Epoch 78/100
484/484 [==============================] - 7s 14ms/step - loss: 1.2910 - accuracy: 0.782
Epoch 79/100
484/484 [==============================] - 8s 16ms/step - loss: 1.2711 - accuracy: 0.787
Epoch 80/100
484/484 [==============================] - 7s 14ms/step - loss: 1.2725 - accuracy: 0.783
Epoch 81/100
484/484 [==============================] - 7s 14ms/step - loss: 1.2350 - accuracy: 0.794
Epoch 82/100
484/484 [==============================] - 7s 15ms/step - loss: 1.2240 - accuracy: 0.793
Epoch 83/100
484/484 [==============================] - 7s 14ms/step - loss: 1.2197 - accuracy: 0.796
Epoch 84/100
484/484 [==============================] - 7s 14ms/step - loss: 1.1975 - accuracy: 0.797
Epoch 85/100
484/484 [==============================] - 7s 14ms/step - loss: 1.1808 - accuracy: 0.799
Epoch 86/100
484/484 [==============================] - 7s 14ms/step - loss: 1.1784 - accuracy: 0.800
Epoch 87/100
484/484 [==============================] - 7s 14ms/step - loss: 1.1682 - accuracy: 0.801
Epoch 88/100
484/484 [==============================] - 7s 14ms/step - loss: 1.1415 - accuracy: 0.809
Epoch 89/100
484/484 [==============================] - 7s 14ms/step - loss: 1.1329 - accuracy: 0.809
Epoch 90/100
484/484 [==============================] - 7s 14ms/step - loss: 1.1358 - accuracy: 0.803
Epoch 91/100
484/484 [==============================] - 7s 14ms/step - loss: 1.1176 - accuracy: 0.812
Epoch 92/100
484/484 [==============================] - 7s 14ms/step - loss: 1.1110 - accuracy: 0.811
Epoch 93/100
484/484 [==============================] - 7s 14ms/step - loss: 1.1086 - accuracy: 0.809
Epoch 94/100
484/484 [==============================] - 7s 14ms/step - loss: 1.0912 - accuracy: 0.812
Epoch 95/100
484/484 [==============================] - 7s 14ms/step - loss: 1.0819 - accuracy: 0.814
Epoch 96/100
484/484 [==============================] - 7s 14ms/step - loss: 1.0724 - accuracy: 0.814
Epoch 97/100
484/484 [==============================] - 7s 14ms/step - loss: 1.0588 - accuracy: 0.817
Epoch 98/100
484/484 [==============================] - 7s 14ms/step - loss: 1.0702 - accuracy: 0.813
Epoch 99/100
484/484 [==============================] - 7s 14ms/step - loss: 1.0699 - accuracy: 0.813
Epoch 100/100
```

```
484/484 [==============================] - 7s 14ms/step - loss: 1.0456 - accuracy: 0.81
```