NEXT_WORD_PREDICTION

```
# This cell will prompt an external url to accept permissions for Colab to access Google Drive

from google.colab import drive
drive.mount("/gdrive")

%ls
```

```
---------------------------------------------------------------------------
MessageError                              Traceback (most recent call last)
<ipython-input-1-2b928ad6c46f> in <cell line: 4>()
      2
      3 from google.colab import drive
----> 4 drive.mount("/gdrive")
      5
      6 get_ipython().run_line_magic('ls', '')

                                 3 frames
/usr/local/lib/python3.9/dist-packages/google/colab/_message.py in
read_reply_from_input(message_id, timeout_sec)
    101     ):
    102       if 'error' in reply:
--> 103         raise MessageError(reply['error'])
    104       return reply.get('data', None)
    105

MessageError: Error: credential propagation was unsuccessful
```

SEARCH STACK OVERFLOW

Import *

```
# Getting all required libraries

import os
import re
import gdown
import numpy
import string
import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf
from absl import logging
import tensorflow_hub as hub
from tensorflow import keras
import matplotlib.pyplot as plt
from keras.models import Sequential
import tensorflow.keras.backend as K
from keras.layers import LSTM
from keras.layers import Dense, Activation
from keras.callbacks import LambdaCallback
from keras.utils.data_utils import get_file
from tensorflow.keras.layers import Embedding
from sklearn.model_selection import train_test_split
```

Data preparation - *Generating Corpus*

```
# Download data from Google drive

'''

ORIGINAL DATASET URL:
    https://raw.githubusercontent.com/maxim5/stanford-tensorflow-tutorials/master/data/arxiv_abstracts.txt

'''

url = ' https://drive.google.com/uc?id=1YTBR7FiXssaKXHhOZbUbwoWw6jzQxxKW'
output = 'corpus.txt'
gdown.download(url, output, quiet=False)

# sentence_length = 40

# Read local file from directory
with open('corpus.txt') as subject:
```

```
  cache = subject.readlines()
translator = str.maketrans('', '', string.punctuation) # Remove punctuation
lines = [doc.lower().translate(translator) for doc in cache] # Switch to lower case
```

```
Downloading...
From:  https://drive.google.com/uc?id=1YTBR7FiXssaKXHhOZbUbwoWw6jzQxxKW
To: /content/corpus.txt
100%|██████████| 7.55M/7.55M [00:00<00:00, 105MB/s]
```

```
# PREVIEW OUTPUT ::

print(lines[0][:100])
len(lines)
```

```
in science and engineering intelligent processing of complex signals such as images sound or languag
7200
```

```
# Generate an list of single/independent words

vocabulary = list(set(' '.join(lines).replace('\n','').split(' ')))
primary_store = {}
for strings, texts in enumerate(vocabulary):
  primary_store[texts] = strings
```

```
# PREVIEW OUTPUT ::

print(vocabulary[:50])
len(vocabulary)
```

```
['augmented', '', 'two', '09', 'jacobian', 'implement', 'inputs', 'endowing', 'character', 'description', 'open', 'energy', 'scient
2694
```

```
# Splitting data into Train sets and test sets

X = []
y = []

for c in lines:
  xxxx = c.replace('\n','').split(' ')
  X.append(' '.join(xxxx[:-1])) # X from the corpus

  yyyy = [0 for i in range(len(vocabulary))] # Generate Y from the Vocabulary
  # yyyy[primary_store[xxxx[-1]]] = 1
  yyyy[primary_store[xxxx[-1]]] = 1
  y.append(yyyy)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
y_test = numpy.array(y_test)
y_train = numpy.array(y_train)
```

```
# PREVIEW OUTPUT ::

print(X_train[:10])
print(y_train[:10])
print(X_test[:10])
print(y_test[:10])
```

```
['in this paper we present an infinite hierarchical nonparametric bayesian model to extract the hidden factors over observed data w
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
['deep learning is a broad set of techniques that uses multiple layers of representation to automatically learn relevant features d
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

Embeddings!

```
# Import the Universal Sentence Encoder's TF Hub module (Here we're making use of version 4)
# This will take a while but won't be long :)

module_url = "https://tfhub.dev/google/universal-sentence-encoder/4"
appreciate = hub.load(module_url)



# REVIEW OUTPUT ::

appreciate.variables
```

```
ListWrapper([<tf.Variable 'Embeddings/sharded_0:0' shape=(26667, 320) dtype=float32, numpy=
array([[-0.44756386, -0.7523224 , -2.8879747 , ..., -4.275423  ,
        -0.50384414, -2.5144944 ],
       [ 0.19551526,  0.18014883,  0.24043915, ..., -0.2736297 ,
         0.10923431,  0.20877579],
       [-0.14282258,  0.0094786 , -0.02283357, ...,  0.05814617,
         0.07281963,  0.1362249 ],
       ...,
       [ 0.25273287, -0.06458262, -0.0383645 , ..., -0.05067257,
        -0.04441866,  0.03095888],
       [-0.01667571,  0.05448845,  0.009569  , ...,  0.00490127,
        -0.04033831,  0.24144703],
       [ 0.03673933, -0.1211024 , -0.03498175, ...,  0.07811887,
        -0.00703663,  0.22115262]], dtype=float32)>, <tf.Variable 'Embeddings/sharded_1:0' shape=(26667, 320) dtype=float32,
numpy=
array([[ 3.1123035e+00,  1.9035040e+00, -2.4504054e+00, ...,
        -3.6000068e+00,  1.6262866e+00,  3.8986406e+00],
       [ 3.1896163e-02,  4.0815596e-02,  3.2115210e-02, ...,
        -8.2207076e-02, -1.8845551e-02, -3.4206249e-02],
       [-3.9929196e-01,  1.4832838e-01,  1.0934803e-01, ...,
        -2.6172850e-01,  1.2915832e-01,  1.9364612e-01],
       ...,
       [-6.2232528e-02,  1.3353802e-01,  4.8703369e-02, ...,
         2.1820575e-02, -1.2878563e-01,  1.6058730e-02],
       [ 1.4297476e-01, -3.2328725e-01,  6.4715120e-04, ...,
         3.7849057e-03, -5.3975854e-02, -4.6804391e-02],
       [-5.9856247e-04,  2.9029913e-02,  4.4148143e-02, ...,
         5.0696049e-02, -8.5392945e-02,  1.9613512e-01]], dtype=float32)>, <tf.Variable 'Embeddings/sharded_2:0' shape=(26667,
320) dtype=float32, numpy=
array([[-0.02090245, -0.006236  , -0.01645498, ...,  0.0606927 ,
        -0.07000386,  0.00829312],
       [ 0.1025753 ,  0.01868178,  0.02571048, ...,  0.08652586,
         0.02998906, -0.06182057],
       [-0.02455923, -0.06302338, -0.08392006, ..., -0.03867638,
         0.01050155, -0.03097543],
       ...,
       [-0.06306946,  0.0579569 ,  0.01718161, ..., -0.10172892,
        -0.13607465,  0.08426023],
       [ 0.08371381,  0.28850275, -0.00061765, ...,  0.03360831,
         0.08284857, -0.02343786],
       [-0.11769697, -0.04799766,  0.03528275, ...,  0.07886682,
         0.01514925,  0.06530713]], dtype=float32)>, <tf.Variable 'Embeddings/sharded_3:0' shape=(26667, 320) dtype=float32,
numpy=
array([[-0.06236736,  0.03764867,  0.04050368, ..., -0.04933948,
         0.01231335,  0.01034924],
       [-0.88279176, -0.22071695,  0.36131623, ..., -0.47532606,
         0.34528401, -0.01592679],
       [ 0.20341368,  0.16160628,  0.09076487, ..., -0.07658941,
         0.10173877,  0.08435548],
       ...,
       [ 0.03058651, -0.04680122,  0.10461159, ..., -0.0586392 ,
         0.07568187,  0.12159212],
       [ 0.07071938,  0.15675475,  0.0238188 , ..., -0.08145189,
         0.03842169, -0.05588495],
       [-0.14082842, -0.05251335, -0.01304664, ..., -0.00758777,
         0.02956374, -0.00818606]], dtype=float32)>, <tf.Variable 'Embeddings/sharded_4:0' shape=(26667, 320) dtype=float32,
numpy=
array([[ 1.60431303e-02, -1.37231708e-01, -8.44638199e-02, ...,
```

```
# Wrapping up with the U-S-E

X_train = appreciate(X_train)
X_test = appreciate(X_test)
X_train = X_train.numpy()
X_test = X_test.numpy()


# PREVIEW OUTPUT ::

print(X_train[:10])
print(y_train[:10])
print(X_test[:10])
print(y_test[:10])
print(X_train.shape, X_test.shape, y_test.shape, y_train.shape)
```

```
[[-0.03771045 -0.06771192 -0.05370539 ...  0.06933318 -0.00784061
  -0.05528935]
 [ 0.01183544 -0.06316999 -0.00105096 ...  0.06420517 -0.05883574
  -0.02919191]
 [-0.03916728 -0.03120759 -0.05427598 ...  0.06412318 -0.06295583
  -0.06137133]
 ...
 [ 0.03922532 -0.06610487 -0.06339797 ...  0.07350148 -0.03273619
  -0.02008302]
 [ 0.00030298 -0.05887232 -0.02081315 ...  0.06502399 -0.05824979
  -0.05000952]
 [ 0.05783224 -0.06084208 -0.03166337 ...  0.06078244 -0.05897183
  -0.04893991]]
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
[[-0.06120554 -0.04776438 -0.05084331 ...  0.06391487 -0.05102952
  -0.06147792]
 [ 0.05910347 -0.0636731  -0.06219527 ...  0.05805291 -0.0441683
  -0.05484253]
 [-0.04787393 -0.0406386  -0.06472679 ...  0.06678559 -0.06455693
  -0.06263316]
 ...
 [-0.05151311 -0.04795231 -0.03584858 ...  0.06181873 -0.03392373
  -0.05439768]
 [ 0.05783226 -0.06084208 -0.0316634  ...  0.06078245 -0.05897183
  -0.04893992]
 [-0.03465602 -0.05981057 -0.04657362 ...  0.0626708  -0.05501131
  -0.06100787]]
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
(5400, 512) (1800, 512) (1800, 2694) (5400, 2694)
```

Building the model

```
model = Sequential()
# model.add(Embedding(input_dim=len(vocabulary), output_dim=100))
model = Sequential()
# model.add(LSTM(units=100, input_shape=[512]))
model.add(Dense(512, input_shape=[512], activation = 'relu'))
model.add(Dense(units=len(vocabulary), activation = 'softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc'])
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 512)               262656

 dense_1 (Dense)             (None, 2694)              1382022

=================================================================
Total params: 1,644,678
Trainable params: 1,644,678
Non-trainable params: 0
_________________________________________________________________
```

```
# Training the model.

model.fit(X_train, y_train, batch_size=512, shuffle=True, epochs=20, validation_data=(X_test, y_test), callbacks=[LambdaCallback()])
```

```
Epoch 1/20
11/11 [==============================] - 2s 45ms/step - loss: 7.6413 - acc: 0.1100 - val_loss: 7.0036 - val_acc: 0.0450
Epoch 2/20
11/11 [==============================] - 0s 21ms/step - loss: 5.9249 - acc: 0.0504 - val_loss: 4.5271 - val_acc: 0.0333
Epoch 3/20
11/11 [==============================] - 0s 19ms/step - loss: 4.2236 - acc: 0.0983 - val_loss: 4.0784 - val_acc: 0.0978
Epoch 4/20
11/11 [==============================] - 0s 21ms/step - loss: 3.9859 - acc: 0.1437 - val_loss: 3.8681 - val_acc: 0.2422
Epoch 5/20
11/11 [==============================] - 0s 19ms/step - loss: 3.7588 - acc: 0.2491 - val_loss: 3.6136 - val_acc: 0.2489
Epoch 6/20
11/11 [==============================] - 0s 18ms/step - loss: 3.4656 - acc: 0.2952 - val_loss: 3.2819 - val_acc: 0.4139
Epoch 7/20
```

```
11/11 [==============================] - 0s 18ms/step - loss: 3.1052 - acc: 0.4707 - val_loss: 2.8881 - val_acc: 0.5472
Epoch 8/20
11/11 [==============================] - 0s 20ms/step - loss: 2.6815 - acc: 0.6852 - val_loss: 2.4487 - val_acc: 0.7806
Epoch 9/20
11/11 [==============================] - 0s 17ms/step - loss: 2.2289 - acc: 0.8789 - val_loss: 1.9964 - val_acc: 0.9144
Epoch 10/20
11/11 [==============================] - 0s 17ms/step - loss: 1.7768 - acc: 0.9611 - val_loss: 1.5542 - val_acc: 0.9783
Epoch 11/20
11/11 [==============================] - 0s 19ms/step - loss: 1.3524 - acc: 0.9915 - val_loss: 1.1636 - val_acc: 1.0000
Epoch 12/20
11/11 [==============================] - 0s 20ms/step - loss: 0.9915 - acc: 1.0000 - val_loss: 0.8437 - val_acc: 1.0000
Epoch 13/20
11/11 [==============================] - 0s 17ms/step - loss: 0.7106 - acc: 1.0000 - val_loss: 0.6071 - val_acc: 1.0000
Epoch 14/20
11/11 [==============================] - 0s 19ms/step - loss: 0.5076 - acc: 1.0000 - val_loss: 0.4348 - val_acc: 1.0000
Epoch 15/20
11/11 [==============================] - 0s 18ms/step - loss: 0.3677 - acc: 1.0000 - val_loss: 0.3203 - val_acc: 1.0000
Epoch 16/20
11/11 [==============================] - 0s 17ms/step - loss: 0.2720 - acc: 1.0000 - val_loss: 0.2433 - val_acc: 1.0000
Epoch 17/20
11/11 [==============================] - 0s 16ms/step - loss: 0.2091 - acc: 1.0000 - val_loss: 0.1904 - val_acc: 1.0000
Epoch 18/20
11/11 [==============================] - 0s 17ms/step - loss: 0.1652 - acc: 1.0000 - val_loss: 0.1531 - val_acc: 1.0000
Epoch 19/20
11/11 [==============================] - 0s 16ms/step - loss: 0.1342 - acc: 1.0000 - val_loss: 0.1264 - val_acc: 1.0000
Epoch 20/20
11/11 [==============================] - 0s 16ms/step - loss: 0.1117 - acc: 1.0000 - val_loss: 0.1062 - val_acc: 1.0000
<keras.callbacks.History at 0x7f917969d520>
```

Unto the tests!

```
# Create function to predict and show detailed output

def next_word(collection=[], extent=1):

  for item in collection:
    text = item
    for i in range(extent):
      prediction = model.predict(x=appreciate([item]).numpy())
      idx = np.argmax(prediction[-1])
      item += ' ' + vocabulary[idx]

      print(text + ' --> ' + item + '\nNEXT WORD: ' + item.split(' ')[-1] + '\n')


# Tests - please feel free to explore

single_text = ['and some other essential']

next_word(single_text)
```

```
1/1 [==============================] - 0s 66ms/step
and some other essential --> and some other essential experiments
NEXT WORD: experiments
```

```
# Testing on a collection of words

text_collection = ['deep convolutional', 'simple and effective', 'a nonconvex', 'a']

next_word(text_collection)
```

```
1/1 [==============================] - 0s 18ms/step
deep convolutional --> deep convolutional networks
NEXT WORD: networks

1/1 [==============================] - 0s 17ms/step
simple and effective --> simple and effective acceleration
NEXT WORD: acceleration

1/1 [==============================] - 0s 19ms/step
a nonconvex --> a nonconvex dataset
NEXT WORD: dataset

1/1 [==============================] - 0s 17ms/step
a --> a accuracy
NEXT WORD: accuracy
```

For the record

The Dataset is based on a Tensorflow tutorial from Stanford, so all predicted words will be based on Deep learning and Machine learning *common terms*.

```
# Storing data

vocabulary = numpy.array(vocabulary)
numpy.save('./vocabulary.npy', vocabulary)
model.save('./NWP-USE')
```