**TAITA TAVETA UNIVERSITY**

**COURSE : BACHELOR OF BUSINESS INFORMATION TECHNOLOGY**

**UNIT CODE : HBT 2403**

**UNIT : SYSTEM PROJECT \*SUBMISSION OF THE PROJECT REPORT**

**GROUP 4**

**NAME** **REG**

**JACKLINE CHEMNGENO SANG** **TU01-BE213-0389/2017**

**HASSAN KOMBO KHALID** **TU01-BE213-0348/2017**

**KAREN KIMWELE** **TU01-BE213-0371/2017**

**ERICK MUSYIMI MATATA** **TU01-BE213-0369/2017**

**ELIZABETH NIFA WANGA** **TU01-BE213-0417/2016**

**NICODEMOUS KIPKORIR** **TU01-BE213-0403/2017**

**BI-DIRECTIONAL MACHINE TRANSLATION BETWEEN ENGLISH AND KISWAHILI .**

1. **Import Libraries**

import numpy as np # linear algebra

import pandas as pd # data processing

import os

import string

from string import digits

import matplotlib.pyplot as plt

# %matplotlib inline

import re

import seaborn as sns

from sklearn.utils import shuffle

from sklearn.model\_selection import train\_test\_split

from keras.layers import Input, LSTM, Embedding, Dense

from keras.models import Model

1. lines=pd.read\_csv("kiswahili\_to\_english.csv",encoding='utf-8')

lines.shape

print(lines)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Number** | **Swahili** | **English** | **length\_eng\_sentence** | **length\_swa\_sentence** |  |
| **956** | 957 | Ondoa | subtract | 1 | 1 |
| **778** | 779 | tabia ya | character | 1 | 2 |
| **4** | 5 | yeye | he | 1 | 1 |
| **614** | 615 | mbegu | seed | 1 | 1 |
| **22** | 23 | baadhi | some | 1 | 1 |
| **...** | ... | ... | ... | ... | ... |
| **912** | 913 | mechi | match | 1 | 1 |
| **168** | 169 | kufanya | make | 1 | 1 |
| **492** | 493 | ramani | map | 1 | 1 |
| **714** | 715 | atakuwa | shall | 1 | 1 |
| **692** | 693 | fimbo | stick | 1 | 1 |

1000 rows × 5 columns

1. **Get English and Swahili vocabulary**

all\_eng\_words=set()

for eng in lines['English']:

for word in eng.split():

if word not in all\_eng\_words:

all\_eng\_words.add(word)

all\_swahili\_words=set()

for swa in lines['Swahili']:

for word in swa.split():

if word not in all\_swahili\_words:

all\_swahili\_words.add(word)

lines['length\_eng\_sentence']=lines['English'].apply(lambda x:len(x.split(" "))) #fuctions that takes in another function

lines['length\_swa\_sentence']=lines['Swahili'].apply(lambda x:len(x.split(" ")))

1. **Now before training the language translation model we need to set the input and target values:**

lines=lines[lines['length\_eng\_sentence']<=20]

lines=lines[lines['length\_swa\_sentence']<=20]

max\_length\_src=max(lines['length\_swa\_sentence'])

max\_length\_tar=max(lines['length\_eng\_sentence'])

input\_words = sorted(list(all\_eng\_words))

target\_words = sorted(list(all\_swahili\_words))

num\_encoder\_tokens = len(all\_eng\_words)

num\_decoder\_tokens = len(all\_swahili\_words)

print(num\_encoder\_tokens, num\_decoder\_tokens)

num\_decoder\_tokens += 1 #for zero padding

input\_token\_index = dict([(word, i+1) for i, word in enumerate(input\_words)])

target\_token\_index = dict([(word, i+1) for i, word in enumerate(target\_words)])

reverse\_input\_char\_index = dict((i, word) for word, i in input\_token\_index.items())

reverse\_target\_char\_index = dict((i, word) for word, i in target\_token\_index.items())

lines = shuffle(lines)

print(lines)

1. **Now as we have prepared our dataset let’s train a model for the task of Language translation model.**

**For this task we will first split the data and then we will move forward to train our model:**

X, y = lines['English'], lines['Swahili']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2,random\_state=42)

X\_train.to\_pickle('X\_train.pkl')

X\_test.to\_pickle('X\_test.pkl')

1. **Training**

def generate\_batch(X = X\_train, y = y\_train, batch\_size = 128):

''' Generate a batch of data '''

while True:

for j in range(0, len(X), batch\_size):

encoder\_input\_data = np.zeros((batch\_size, max\_length\_src),dtype='float32')

decoder\_input\_data = np.zeros((batch\_size, max\_length\_tar),dtype='float32')

decoder\_target\_data = np.zeros((batch\_size, max\_length\_tar, num\_decoder\_tokens),dtype='float32')

for i, (input\_text, target\_text) in enumerate(zip(X[j:j+batch\_size], y[j:j+batch\_size])):

for t, word in enumerate(input\_text.split()):

encoder\_input\_data[i, t] = input\_token\_index[word] # encoder input seq

for t, word in enumerate(target\_text.split()):

try:

if t<len(target\_text.split())-1:

decoder\_input\_data[i, t] = target\_token\_index[word] # decoder input seq

if t>0:

* decoder target sequence (one hot encoded)
* does not include the START\_ token
* Offset by one timestep

decoder\_target\_data[i, t - 1, target\_token\_index[word]] = 1.

except Exception as e:

print("")

yield([encoder\_input\_data, decoder\_input\_data], decoder\_target\_data)

latent\_dim=300

encoder\_inputs = Input(shape=(None,))

enc\_emb = Embedding(num\_encoder\_tokens, latent\_dim, mask\_zero = True)(encoder\_inputs)

encoder\_lstm = LSTM(latent\_dim, return\_state=True)

encoder\_outputs, state\_h, state\_c = encoder\_lstm(enc\_emb)

* **We discard `encoder\_outputs` and only keep the states.**

encoder\_states = [state\_h, state\_c]

decoder\_inputs = Input(shape=(None,))

dec\_emb\_layer = Embedding(num\_decoder\_tokens, latent\_dim, mask\_zero = True)

dec\_emb = dec\_emb\_layer(decoder\_inputs)

* **We set up our decoder to return full output sequences, and to return internal states as well. We don't use the return states in the training model, but we will use them in inference.**

decoder\_lstm = LSTM(latent\_dim, return\_sequences=True, return\_state=True)

decoder\_outputs, \_, \_ = decoder\_lstm(dec\_emb,

initial\_state=encoder\_states)

decoder\_dense = Dense(num\_decoder\_tokens, activation='softmax') into probability

decoder\_outputs = decoder\_dense(decoder\_outputs)

* **Define the model that will turn, `encoder\_input\_data` & `decoder\_input\_data` into `decoder\_target\_data`**

model = Model([encoder\_inputs, decoder\_inputs], decoder\_outputs)

model.compile(optimizer='rmsprop', loss='categorical\_crossentropy')

model.summary()

train\_samples = len(X\_train)

val\_samples = len(X\_test)

batch\_size = 128

epochs = 100

model.fit(generate\_batch(X\_train, y\_train, batch\_size = batch\_size),

steps\_per\_epoch = train\_samples//batch\_size,

epochs=epochs,

validation\_data = generate\_batch(X\_test, y\_test, batch\_size = batch\_size),

validation\_steps = val\_samples//batch\_size)

model.save\_weights('nmt\_weights.h5')

* **Encode the input sequence to get the "thought vectors"**

Encode the input sequence to get the "thought vectors"

* **Decoder setup, Below tensors will hold the states of the previous time step**

decoder\_state\_input\_h = Input(shape=(latent\_dim,))

decoder\_state\_input\_c = Input(shape=(latent\_dim,))

decoder\_states\_inputs = [decoder\_state\_input\_h, decoder\_state\_input\_c]

* **Get the embeddings of the decoder sequence**

dec\_emb2= dec\_emb\_layer(decoder\_inputs)

* **To predict the next word in the sequence, set the initial states to the states from the previous time step**

decoder\_outputs2, state\_h2, state\_c2 = decoder\_lstm(dec\_emb2, initial\_state=decoder\_states\_inputs) j

decoder\_states2 = [state\_h2, state\_c2]

decoder\_outputs2 = decoder\_dense(decoder\_outputs2)

* **Final decoder model**

decoder\_model = Model(

[decoder\_inputs] + decoder\_states\_inputs,

[decoder\_outputs2] + decoder\_states2)

def decode\_sequence(input\_seq)

* **encode the input as state vectors**.

states\_value = encoder\_model.predict(input\_seq)

* **Generate empty target sequence of length 1.**

target\_seq = np.zeros((1,1))

* **Populate the first character of target sequence with the start character.**

target\_seq[0, 0] = target\_token\_index['START\_'**]**

1. **Sampling loop for a batch of sequences , to simplify, here we assume a batch of size 1).**

stop\_condition = False

decoded\_sentence = ''

while not stop\_condition:

output\_tokens, h, c = decoder\_model.predict([target\_seq] + states\_value)

1. **Sample a token**

sampled\_token\_index = np.argmax(output\_tokens[0, -1, :])

sampled\_char = reverse\_target\_char\_index[sampled\_token\_index]

decoded\_sentence += ' '+sampled\_char

1. **Exit condition: either hit max length, or find stop character.**

if (sampled\_char == '\_END' or

len(decoded\_sentence) > 50):

stop\_condition = True

1. **Update the target sequence (of length 1).**

target\_seq = np.zeros((1,1))

target\_seq[0, 0] = sampled\_token\_index

1. **Update states**

states\_value = [h, c]

return decoded\_sentence

1. **train\_gen = generate\_batch(X\_train, y\_train, batch\_size = 1)**

k=-1

k+=1

input\_seq = "This is good"

actual\_output = "Hii ni mzuri"

1. **(input\_seq, actual\_output), \_ = next(train\_gen)**

decoded\_sentence = decode\_sequence(input\_seq)

print('Input English sentence:', X\_train[k:k+1].values[0])

print('Actual Swahili Translation:', y\_train[k:k+1].values[0][6:-4])

print('Predicted Swahili Translation:', decoded\_sentence[:-4])

1. **OUTPUT**

Number Swahili English  
0 1 kama as  
1 2 mimi I  
2 3 yake his  
3 4 kwamba that  
4 5 yeye he  
.. ... ... ...  
995 996 pua nose  
996 997 wingi plural  
997 998 hasira anger  
998 999 madai claim  
999 1000 bara continent  
  
[1000 rows x 3 columns]  
999 810  
 Number Swahili English length\_eng\_sentence length\_swa\_sentence  
956 957 Ondoa subtract 1 1  
778 779 tabia ya character 1 2  
4 5 yeye he 1 1  
614 615 mbegu seed 1 1  
22 23 baadhi some 1 1  
.. ... ... ... ... ...  
912 913 mechi match 1 1  
168 169 kufanya make 1 1  
492 493 ramani map 1 1  
714 715 atakuwa shall 1 1  
692 693 fimbo stick 1 1  
  
[1000 rows x 5 columns]

Model: "model"  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 Layer (type) Output Shape Param # Connected to   
==================================================================================================  
 input\_1 (InputLayer) [(None, None)] 0 []   
   
 input\_2 (InputLayer) [(None, None)] 0 []   
   
 embedding (Embedding) (None, None, 300) 299700 ['input\_1[0][0]']   
   
 embedding\_1 (Embedding) (None, None, 300) 243300 ['input\_2[0][0]']   
   
 lstm (LSTM) [(None, 300), 721200 ['embedding[0][0]']   
 (None, 300),   
 (None, 300)]   
   
 lstm\_1 (LSTM) [(None, None, 300), 721200 ['embedding\_1[0][0]',   
 (None, 300), 'lstm[0][1]',   
 (None, 300)] 'lstm[0][2]']   
   
 dense (Dense) (None, None, 811) 244111 ['lstm\_1[0][0]']   
   
==================================================================================================  
Total params: 2,229,511  
Trainable params: 2,229,511  
Non-trainable params: 0  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
  
  
  
Epoch 1/100  
  
  
  
  
2/6 [=========>....................] - ETA: 0s - loss: 0.3399   
  
3/6 [==============>...............] - ETA: 0s - loss: 0.2961