TAITA TAVETA UNIVERSITY

**HBT: 2403**

SYSTEM PROJECT; SUBMISSION OF PROJECT REPORT

**GROUP 3**

**NAME REG. NO**

THOMAS ONGWAE TU01-BE213-0380/2017

RUKIA SALIM TU01-BE213-0393/2017

STEPHEN MUNGAI TU01-BE213-0346/2016

SHANON MGOI KITI TU01-BE213-0344/2017

LEYANOI MIKE TU01-BE213-0407/2017

REUBEN MUGANE TU01-BE213-0399/2017

**BIDIRECTIONAL MACHINE TRANSLATION FROM ENGLISH TO FRENCH**

**Step 1: Importing Required Libraries**

In this phase, we will make use of the TensorFlow and Keras libraries for creating our model and training it. TensorFlow is an open-source library used for building models which would result in the tasks like machine translation. Keras is in high level application programming interface that can use TensorFlow’s functions to provide for loading, translating languages.

**import** helper

**import** numpy **as** np

**from** keras.preprocessing.text **import** Tokenizer

**from** keras.preprocessing.sequence **import** pad\_sequences

**from** keras.models **import** Model, Sequential

**from** keras.layers **import** GRU, Input, Dense, TimeDistributed, Activation, RepeatVector, Bidirectional, Dropout, LSTM

**from** keras.layers.embeddings **import** Embedding

**from** keras.optimizers **import** Adam

**from** keras.losses **import** sparse\_categorical\_crossentropy

**import** tensorflow **as** tf

**STEP 2: LOAD DATA**

The data is located in . The small\_vocab\_en file contains English sentences with their French translations in the small\_vocab\_fr file. Load the English and French data from these files from running the cell below.

english\_path**=**'https://raw.githubusercontent.com/projjal1/datasets/master/small\_vocab\_en.txt'

french\_path**=**'https://raw.githubusercontent.com/projjal1/datasets/master/small\_vocab\_fr.txt'

load the dataset and split file by lines

**import** os

**def** load\_data(path):

input\_file **=** os**.**path**.**join(path)

**with** open(input\_file, "r") **as** f:

data **=** f**.**read()

**return** data**.** split('\n')

using helper to load the dataset

english\_data**=**tf**.**keras**.**utils**.**get\_file('file1',english\_path)

french\_data**=**tf**.**keras**.**utils**.**get\_file('file2',french\_path)

loading data

english\_sentences=load\_data(english\_data)

french\_sentences=load\_data(french\_data)

**STEP 4:ANALYZE THE DATASET**

**for** i **in** range(5):

print('Sample:',i)

print(english\_sentences[i])

print(french\_sentences[i])

print('-'**\***50)

Sample : 0

new jersey is sometimes quiet during autumn , and it is snowy in april .

new jersey est parfois calme pendant l' automne , et il est neigeux en avril .

--------------------------------------------------

Sample : 1

the united states is usually chilly during july , and it is usually freezing in november .

les états-unis est généralement froid en juillet , et il gèle habituellement en novembre .

--------------------------------------------------

Sample : 2

california is usually quiet during march , and it is usually hot in june .

california est généralement calme en mars , et il est généralement chaud en juin .

--------------------------------------------------

Sample : 3

the united states is sometimes mild during june , and it is cold in september .

les états-unis est parfois légère en juin , et il fait froid en septembre .

--------------------------------------------------

Sample : 4

your least liked fruit is the grape , but my least liked is the apple .

**STEP 5: CONVERT TO VOCABULARY**

**import collections**

**english\_words\_counter = collections.Counter([word for sentence in english\_sentences for word in sentence.split()])**

**french\_words\_counter = collections.Counter([word for sentence in french\_sentences for word in sentence.split()])**

**print('English Vocab:',len(english\_words\_counter))**

**print('French Vocab:',len(french\_words\_counter))**

English Vocab: 227

French Vocab: 355

**STEP 5: TOKENIZE**

The neural network predicts text data and turns it in an aspect in which it can understand. Since a neural network is a series of multiplication and addition operations, the input data needs to be number(s).

Turns each character into a number. Character ids generate text predictions for each character. Word level model uses word ids that generate text predictions for each word.

Turns each sentence into a sequence of words ids using Keras's Tokenizer function.

**def** tokenize(x):

tokenizer **=** Tokenizer()

tokenizer**.**fit\_on\_texts(x)

**return** tokenizer**.**texts\_to\_sequences(x), tokenizer

**STEP 6: PADDING**

We add padding to the end of the sequences to make them the same length.

**def** pad (x, length**=None**):

**return** pad\_sequences(x, maxlen**=**length, padding**=**'post')

**def** preprocess(x, y):

"""

Preprocess x and y

:param x: Feature List of sentences

:param y: Label List of sentences

:return: Tuple of (Preprocessed x, Preprocessed y, x tokenizer, y tokenizer)

"""

preprocess\_x, x\_tk **=** tokenize(x)

preprocess\_y, y\_tk **=** tokenize(y)

preprocess\_x **=** pad(preprocess\_x)

preprocess\_y **=** pad(preprocess\_y)

*# Keras's sparse\_categorical\_crossentropy function requires the labels to be in 3 dimensions*

*#Expanding dimensions*

preprocess\_y **=** preprocess\_y**.**reshape(**\***preprocess\_y**.**shape, 1)

**return** preprocess\_x, preprocess\_y, x\_tk, y\_tk

preproc\_english\_sentences, preproc\_french\_sentences, english\_tokenizer, french\_tokenizer **=**preprocess(english\_sentences, french\_sentences)

max\_english\_sequence\_length **=** preproc\_english\_sentences**.**shape[1]

max\_french\_sequence\_length **=** preproc\_french\_sentences**.**shape[1]

english\_vocab\_size **=** len(english\_tokenizer**.**word\_index)

french\_vocab\_size **=** len(french\_tokenizer**.**word\_index)

print('Data Preprocessed')

print("Max English sentence length:", max\_english\_sequence\_length)

print("Max French sentence length:", max\_french\_sequence\_length)

print("English vocabulary size:", english\_vocab\_size)

print ("French vocabulary size:", french\_vocab\_size)

OUTPUT

Data Preprocessed

Max English sentence length: 15

Max French sentence length: 21

English vocabulary size: 199

French vocabulary size: 344

**STEP 7: CREATE MODEL**

We have four relatively simple architectures.

Model 1 is a simple RNN model, Model 2 is a RNN with Embedding, Model 3 is a Bidirectional RNN, Model 4 is an optional Encoder-Decoder RNN.

**IDS BACK TO TEXT**

**def logits\_to\_text(logits, tokenizer):**

**index\_to\_words = {id: word for word, id in tokenizer.word\_index. items()}**

**index\_to\_words[0] = '<PAD>'**

**#***So basically we are predicting output for a given word and then selecting best answer*

*#Then selecting that label we reverse-enumerate the word from id*

**return ' '.join([index\_to\_words[prediction] for prediction in np.argmax(logits, 1)])**

**STEP 8 : BUILDING MODEL**

We use RNN model combined with GRU nodes for translation

**def** embed\_model(input\_shape, output\_sequence\_length, english\_vocab\_size, french\_vocab\_size):

"""

Build and train a RNN model using word embedding on x and y

:param input\_shape: Tuple of input shape

:param output\_sequence\_length: Length of output sequence

:param english\_vocab\_size: Number of unique English words in the dataset

:param french\_vocab\_size: Number of unique French words in the dataset

:return: Keras model built, but not trained

"""

*# Implement*

*# Hyperparameters*

learning rate **=** 0.005

*# Build the layers*

model **=** Sequential()

model**.**add(Embedding(english\_vocab\_size, 256, input\_length**=**input\_shape[1], input\_shape**=**input\_shape[1:]))

model**.**add(GRU(256, return\_sequences**=True**))

model**.**add(TimeDistributed(Dense(1024, activation**=**'relu')))

model**.**add(Dropout(0.5))

model**.**add (TimeDistributed (Dense (french\_vocab\_size, activation**=**'softmax')))

*# Compile model*

model**.**compile(loss**=**sparse\_categorical\_crossentropy,

optimizer**=**Adam(learning\_rate),

metrics**=**['accuracy'])

**return** model

**STEP 9: Reshaping the input to work with a basic RNN**

tmp\_x **=** pad(preproc\_english\_sentences, preproc\_french\_sentences**.**shape[1])

tmp\_x **=** tmp\_x**.** reshape((**-**1, preproc\_french\_sentences**.**shape[**-**2]))

C

**STEP 10: Calling the model function**

simple\_rnn\_model **=** embed\_model(

tmp\_x**.**shape,

preproc\_french\_sentences**.**shape[1],

len(english\_tokenizer**.**word\_index)**+**1,

len(french\_tokenizer**.**word\_index)**+**1)

simple\_rnn\_model**.**summary()

OUTPUT

Model: "sequential\_1"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

embedding\_1 (Embedding) (None, 21, 256) 51200

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

gru\_1 (GRU) (None, 21, 256) 393984

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

time\_distributed\_1 (TimeDist (None, 21, 1024) 263168

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_1 (Dropout) (None, 21, 1024) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

time\_distributed\_2 (TimeDist (None, 21, 345) 353625

=================================================================

Total params: 1,061,977

Trainable params: 1,061,977

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**STEP 11: TRAINING THE MODEL**

We train the model and pass the english text and the max\_sequence\_length, with vocab size for both english and french text.

history**=**simple\_rnn\_model**.** Fit(tmp\_x, preproc\_french\_sentences, batch\_size**=**1024, epochs**=**20, validation\_split**=**0.2)

OUTPUT

Tensor of unknown shape. This may consume a large amount of memory.

"Converting sparse IndexedSlices to a dense Tensor of unknown shape. "

Train on 110288 samples, validate on 27573 samples

Epoch 1/20

110288/110288 [==============================] - 10s 87us/step - loss: 1.5003 - accuracy: 0.6585 - val\_loss: 0.5446 - val\_accuracy: 0.8263

Epoch 2/20

110288/110288 [==============================] - 9s 79us/step - loss: 0.4507 - accuracy: 0.8529 - val\_loss: 0.3248 - val\_accuracy: 0.8912

Epoch 3/20

110288/110288 [==============================] - 9s 80us/step - loss: 0.3109 - accuracy: 0.8964 - val\_loss: 0.2486 - val\_accuracy: 0.9156

Epoch 4/20

110288/110288 [==============================] - 9s 80us/step - loss: 0.2547 - accuracy: 0.9145 - val\_loss: 0.2265 - val\_accuracy: 0.9235

Epoch 5/20

110288/110288 [==============================] - 9s 80us/step - loss: 0.2280 - accuracy: 0.9222 - val\_loss: 0.2084 - val\_accuracy: 0.9280

Epoch 6/20

110288/110288 [==============================] - 9s 80us/step - loss: 0.2114 - accuracy: 0.9269 - val\_loss: 0.2013 - val\_accuracy: 0.9302

Epoch 7/20

110288/110288 [==============================] - 9s 80us/step - loss: 0.2010 - accuracy: 0.9301 - val\_loss: 0.1937 - val\_accuracy: 0.9321

Epoch 8/20

110288/110288 [==============================] - 9s 80us/step - loss: 0.1924 - accuracy: 0.9325 - val\_loss: 0.1876 - val\_accuracy: 0.9346

Epoch 9/20

110288/110288 [==============================] - 9s 81us/step - loss: 0.1866 - accuracy: 0.9339 - val\_loss: 0.1859 - val\_accuracy: 0.9349

Epoch 10/20

110288/110288 [==============================] - 9s 82us/step - loss: 0.1836 - accuracy: 0.9348 - val\_loss: 0.1856 - val\_accuracy: 0.9352

Epoch 11/20

110288/110288 [==============================] - 9s 81us/step - loss: 0.1788 - accuracy: 0.9361 - val\_loss: 0.1855 - val\_accuracy: 0.9352

Epoch 12/20

110288/110288 [==============================] - 9s 82us/step - loss: 0.1763 - accuracy: 0.9367 - val\_loss: 0.1791 - val\_accuracy: 0.9371

Epoch 13/20

110288/110288 [==============================] - 9s 81us/step - loss: 0.1719 - accuracy: 0.9378 - val\_loss: 0.1787 - val\_accuracy: 0.9376

Epoch 14/20

110288/110288 [==============================] - 9s 82us/step - loss: 0.1696 - accuracy: 0.9385 - val\_loss: 0.1811 - val\_accuracy: 0.9370

Epoch 15/20

110288/110288 [==============================] - 9s 82us/step - loss: 0.1721 - accuracy: 0.9378 - val\_loss: 0.1799 - val\_accuracy: 0.9377

Epoch 16/20

110288/110288 [==============================] - 9s 82us/step - loss: 0.1719 - accuracy: 0.9380 - val\_loss: 0.1812 - val\_accuracy: 0.9373

Epoch 17/20

110288/110288 [==============================] - 9s 82us/step - loss: 0.1679 - accuracy: 0.9389 - val\_loss: 0.1796 - val\_accuracy: 0.9384

Epoch 18/20

110288/110288 [==============================] - 9s 82us/step - loss: 0.1632 - accuracy: 0.9401 - val\_loss: 0.1788 - val\_accuracy: 0.9380

Epoch 19/20

110288/110288 [==============================] - 9s 82us/step - loss: 0.1613 - accuracy: 0.9405 - val\_loss: 0.1779 - val\_accuracy: 0.9387

Epoch 20/20

110288/110288 [==============================] - 9s 82us/step - loss: 0.1621 - accuracy: 0.9403 - val\_loss: 0.1779 - val\_accuracy: 0.9389

**STEP 12: SAVE MODEL**

**simple\_rnn\_model.save('model.h5')**

**STEP 13: PREDICTION**

Performing predictions on the models using User Input

**def** final\_predictions(text):

y\_id\_to\_word **=** {value: key **for** key, value **in** french\_tokenizer**.**word\_index**.**items()}

y\_id\_to\_word[0] **=** '<PAD>'

sentence **=** [english\_tokenizer**.**word\_index[word] **for** word **in** text**.**split()]

sentence **=** pad\_sequences([sentence], maxlen**=**preproc\_french\_sentences**.**shape[**-**2], padding**=**'post')

print(sentence**.**shape)

print(logits\_to\_text(simple\_rnn\_model**.**predict(sentence[:1])[0], french\_tokenizer))

**STEP 14: USER INPUT**

Enter your input here to get predictions

**import** re

txt**=**input ()**.** lower ()

final\_predictions(re**.**sub(r'[^\w]', ' ', txt))

OUTPUT

she liked the big green automobile .

(1, 21)

elle aimait la grande voiture verte