MAULANA AZAD NATIONAL INSTITUTE OF TECHNOLOGY

BHOPAL

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

MINOR PROJECT REPORT, APRIL 2019

ON

CONVERSION OF ENGLISH TEXT FROM WRITTEN EXPRESSIONS TO SPOKEN FORMS

SUBMITTED BY:

JINIA KONAR (161112020)

AMIT KUMAR YADAV (161112068)

HARSH VARDHAN SINGH (161112067)

BHUKYATHIRUMAL(161112045)

UNDER THE GUIDANCE OF

Dr. VASUDEV DEHALWAR

(Associate. Professor)

MAULANA AZAD NATIONAL INSTITUTE OF TECHNOLOGY

Bhopal-462003

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

We, hereby declare that the following report which is being presented in the Minor Project Documentation entitled "**Convert English text from written expressions into spoken forms**" is the partial fulfillment of the requirements of the third year (sixth semester) Minor Project in the field of Computer Science and Engineering. It is an authentic documentation of our own original work carried out under the able guidance of **Dr. Vasudev Dehalwar** and acknowledgment of Dr. Sujoy Das. The work has been carried out entirely at **Maulana Azad National Institute of Technology, Bhopal**. The following project and its report, in part or whole, has not been presented or submitted by us for any purpose in any other institute or organization.

We, hereby, declare that the facts mentioned above are true to the best of our knowledge. In case of any unlikely discrepancies that may possibly occur, we will be the ones to take responsibility.

JiniaKonar (161112020)
Amit Kumar Yadav (161112068)
Harsh Vardhan Singh (161112067)
Bhukya Thirumal (161112045)

ACKNOWLEDGEMENT

We express our deep sense of gratitude to our respected and learned mentors Dr. Vasudev Dehalwar and Dr. Sujoy Das, HOD (Mathematics department) for their valuable guidance.

We are also grateful to respected director Dr. N.S. Raghuwanshi for permitting us to utilize all the necessary facilities of the college.

In addition we are grateful to respected HOD Dr. MeenuChawala for permitting us to utilize all the necessary laboratories of the department.

We are also thankful to all the other faculty and staff members of our department for their kind cooperation and help.

Lastly, we would like to express our deep apperception towards our classmates and our indebtedness to our parents for providing us the moral support.

MAULANA AZAD NATIONAL INSTITUTE OF TECHNOLOGY
BHOPAL

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that JiniaKonar, Amit Kumar Yadav, Harsh Vardhan Singh and Bhukya Thirumal students of B.Tech 3rd Year (Computer Science & Engineering), have successfully completed there minor project “Convert English text from written expressions into spoken forms” in partial fulfillment of their minor project in Computer Science & Engineering.

Dr. Vasudev Dehalwar
(Project Guide)

Dr.RajeshWadhvani
(Project Coordinator)

TABLE OF CONTENTS

ABSTRACT

We perform text normalization, i.e. the transformation of words from the written to the spoken form, using a memory augmented neural network. With the addition of dynamic memory access and storage mechanism, we present a neural architecture that will serve as a language-agnostic text normalization system while avoiding the kind of unacceptable errors made by the LSTM (Long Short Term Memory) - based recurrent neural networks. Our proposed system requires significantly lesser amounts of data, training time and compute resources. Additionally,we perform data up-sampling, circumventing the data sparsity problem in some semiotic classes, to show that sufficient examples in any particular class can improve the performance of our text normalization system. Although a few occurrences of these errors still remain in certain semiotic classes, we demonstrate that memory augmented networks with meta-learning capabilities can open many doors to a superior text normalization system.

The success of our simple LSTM-based approach suggests that it should do well on many other sequence learning problems, provided they have enough training data. Due to hardware restriction we were not able to train our model over large dataset (over 100 K),so there are some wrong predictions also, specially in dates. But we trained upto 20K sentences. However with very small dataset it gave one out of every fifth prediction correct with 100 percent accuracy.

1. INTRODUCTION

As many of us can attest, learning another language is tough. Picking up on nuances like slang, dates and times, and local expressions, can often be a distinguishing factor between proficiency and fluency.

Many speech and language applications, including text-to-speech synthesis (TTS) and automatic speech recognition (ASR), require text to be converted from written expressions into appropriate "spoken" forms. This is a process known as text normalization, and helps convert 12:47 to "twelve forty-seven" and $3.16 into "three dollars, sixteen cents." In this, we are challenged to automate the process of developing text normalization grammars via machine learning and we will focus on English.

While various approaches have been taken and some NN architectures have surely been carefully designed for the specific task, there is also a widespread feeling that with deep enough architectures, and enough data, one can simply feed the data to one's NN and have it learnt the necessary function. For example: Not only do such networks require less human effort than traditional approaches, they generally deliver superior performance. This is particularly true when very large amounts of training data are available, as the benefits of holistic optimization tend to outweigh those of prior knowledge.

Text Normalization, specifically in the sense of a system that converts from a written representation of a text into a representation of how that text is to be read aloud. The target applications are TTS and ASR — in the latter case mostly for generating language modeling data from raw written text. This problem, while often considered mundane, is in fact very important, and a major source of degradation of perceived quality in TTS systems in particular can be traced to problems with text normalization.

We start by describing the prior work in this area, which includes gaps identified in text normalization. We describe the proposed work which include our approach and experiments we will perform with different Neural Network architectures.

2. LITERATURE REVIEW

Text normalization is the process of transforming text into a single canonical form that it might not have had before. Normalizing text before storing or processing it allows for separation of concerns, since input is guaranteed to be consistent before operations are performed on it. Text normalization requires being aware of what type of text is to be normalized and how it is to be processed afterwards; there is no all-purpose normalization procedure.

Text normalization is frequently used when converting text to speech. Numbers, dates, acronyms, and abbreviations are non-standard "words" that need to be pronounced differently depending on context. For example:

"$200" would be pronounced as "two hundred dollars" in English, but as "luaselautālā" in Samoan.

"vi" could be pronounced as "vie," "vee," or "the sixth" depending on the surrounding words.

Text can also be normalized for storing and searching in a database.

- For instance, if a search for "resume" is to match the word "résumé," then the text would be normalized by removing diacritical marks;
- and if "john" is to match "John", the text would be converted to a single case.
- To prepare text for searching, it might also be stemmed (e.g. converting "flew" and "flying" both into "fly"),
- canonicalized (e.g. consistently using American or British English spelling), or have stop words removed.

For simple, context-independent normalization, such as removing non-alphanumeric characters or diacritical marks, regular expressions would suffice. For example, the sed script sed ‐ e "s/\s+/ /g" inputfile would normalize runs of whitespace characters into a single space. More complex normalization requires correspondingly complicated algorithms, including domain knowledge of the language and vocabulary being normalized. Among other approaches, text normalization has been modeled as a problem of tokenizing and tagging streams of text and as a special case of machine translation

Text simplification is an operation used in natural language processing to modify, enhance, classify or otherwise process an existing corpus of human-readable text in such a way that the grammar and structure of the prose is greatly simplified, while the underlying meaning and information remains the same. Text simplification is an important area of research, because natural human languages ordinarily contain complex compound constructions that are not easily processed through automation. In terms of reducing language diversity, semantic compression can be employed to limit and simplify a set of words used in given texts.

In computer science, canonicalization (sometimes standardization or normalization) is a process for converting data that has more than one possible representation into a "standard", "normal", or canonical form. This can be done to compare different representations for equivalence, to count the number of distinct data structures, to improve the efficiency of various algorithms by eliminating repeated calculations, or to make it possible to impose a meaningful sorting order.

2.1 DATASET

The original work by Sproat and Jaitly uses 1.1 billion words for English text and 290 words for Russian text. In this work we used a subset of the dataset submitted by the authors for the Kaggle competition 2. The dataset is derived from Wikipedia regions which could be decoded as UTF8.The text is then divided into sentences and through the Google TTS system's Kestrel text normalization system to produce the normalized version of that text. A snippet is shown in the below. As described in (Ebden and Sproat, 2014), Kestrel's verbalizations are produced by first tokenizing the input and classifying the tokens, and then verbalizing each token according to its semiotic class. The majority of the rules are hand - built using the Thrax finite - state grammar development system (Roarketal.,2012). Most ordinary words are of course left alone (represented here as <self>), and punctuation symbols are mostly transduced to <sil> (for "silence").

Sproat and Jaitly report that a manual analysis of about 1,000 examples from the test data suggests an overall error rate of approximately 0.1% for English. Note that although the test data were of course taken from a different portion of the Wikipedia text than the training and development data, nonetheless a huge percentage of the individual tokens of the test data 99.5% in the case of English - were found in the training set.This in itself is perhaps not so surprising but it does raise the concern that the RNN models may in fact be memorizing their results, without doing much generalization.

Data	No.of tokens
Train	9,918,442
Test	1,088,565

KaggleDataset

3. GAPS IDENTIFIED

However, one of the biggest challenges when developing a TTS or ASR system for new language is to develop and test the grammar for all these rules, task that requires quite a bit of linguistic sophistication and native speaker intuition. Let us consider a simple example such as the following:

“A baby giraffe is 6ft tall and weighs 150 lb.”

If one were to ask a speaker of English to read this sentence, or if one were to feed it to an English TTS system one would expect that it to be read more or less as follows:

A <self>
baby <self>
giraffe <self>
is <self>
6ft six feet
tall <self>
and <self>
weighs <self>
150lb one hundred fifty pounds.

In this example, the token <self> indicates that the token is to be left alone.

In the original written form there are two nonstandard words, namely the two measure expressions 6 ft and 150 lb. In order to read the text, each of these must be normalized into a sequence of ordinary words. In this case both examples are instances of the same semiotic class, namely measure phrases. But in general texts may include non-standard word sequences from a variety of different semiotic classes, including measures, currency amounts, dates, times, telephone numbers, cardinal or ordinal numbers, fractions, among many others and in many different contexts. Each of these involves a specific function mapping between the written input form and the spoken output form.

4. TOOLS AND TECHNOLOGY USED

- Python
- RNNP
- ML
- Tensorflow
- Keras
- Matplotlib
- Numpy
- Scipy
- Kaggle Dataset for Training
- Google Colab to train model

5. PROPOSED WORK, IMPLEMENTATION AND CODING

5.1 IMPLEMENTATION LANGUAGE (PYTHON 3.7)

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. Python has a large comprehensive standard library and more than 25 thousand extension modules. We are aiming Python libraries like Numpy, Scipy, Matplotlib etc for developing system that can pre-process data as well as normalize texts.

The module used are discussed below:

- Read original text from dataset in csv file.
- Normalize original dataset and create a new file which the text to learn.
- One-Hot-Encoding of words and characters
- Compute train and validation loss and update the model over time
- Pick text from test dataset and predict normalized text

5.2 MODULE 1

- Read original text from dataset

We used pandas dataframe to bring data from csv in secondary memory to main memory. We have not use plain and puct text for training since those are finite in count and could be learned easily.

Import necessary libraries and configure hardware:

```
# import modules
import pandas as pd
```

```
importnumpyas np
import csv

importtensorflowastf
fromkerasimport backend as K

#set up hardware config for current session
num_cores = 4

# we are using both GPU and CPU for training
GPU = True
CPU = True
if GPU:
num_GPU = 1
num_CPU = 1
elif CPU:
num_CPU = 1
num_GPU = 0

# since our backend is on Tensorflow, setup the session with Tensorflow
config = tf.ConfigProto(intra_op_parallelism_threads=num_cores,
inter_op_parallelism_threads=num_cores,
allow_soft_placement=True,
device_count = {'CPU' : num_CPU,
'GPU' : num_GPU}
                    )

session = tf.Session(config=config)
K.set_session(session)
# setup completed
```

Declare global variables such as data path :

```
in_path = '/content/datasets/en_train.csv'
out_path = '/content/datasets/en_train_normalised.csv'
```

Open and read file :

```
out_file = open(out_path, 'w')
file_en = open(in_path)
df_pandas = pd.read_csv(path,  names = ["sentence_id", "token_id", "class", "before", "after"],
low_memory=False)
```

Print length of samples to read:

```
print(len(df_pandas))

Iterate over each sample and write to out_fileif this text classtypeisnotPLAINorPUNCT:
k=0
forsent_id, class_type, before, after in zip(df_pandas['sentence_id'], df_pandas['class'],
df_pandas['before'], df_pandas['after']):
ifnot (class_type == 'PLAIN'orclass_type == 'PUNCT'):
        words  = [before, after]
        writer = csv.writer(out_file)
writer.writerow(words)
if k < 10:
            print(sent_id, before, after)
        k = k + 1

Close file :
out_file.close()
```

Output:

```
310919

sentence_id before after

1 2006 two thousand six

1 IUCN i u c n

3 2007 two thousand seven

5 2008 two thousand eight

6 91 ninety one

8 04-Mar-14 the fourth of march twenty fourteen

9 BC b c

9 3 three

10 35 thirty five
```

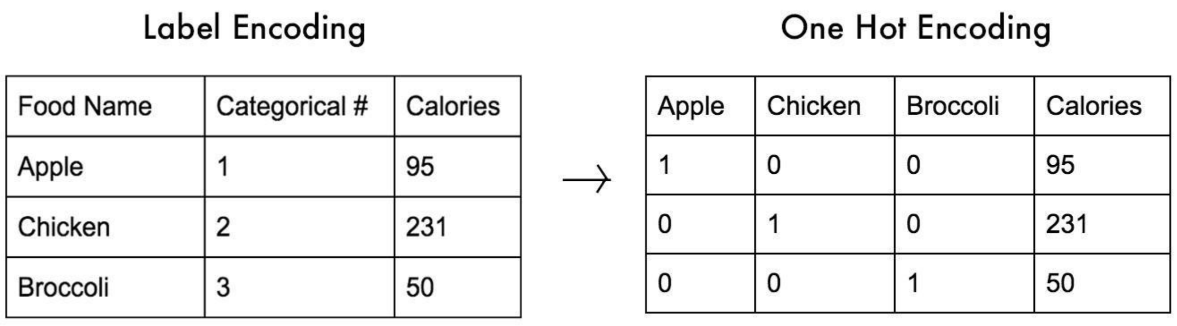
5.3 MODULE 2

- Pre-Process text in dataset
  - ➢ Separate input and target sentences
  - ➢ Assign each character and each word token a unique int id
  - ➢ Extract the exact word from sentences
  - ➢ Convert each word in form of One-Hot-Encoding

One-Hot-Encoding :

One hot encoding is a process by which categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction.

Example: fig 1



Label Encoding

Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50

→

One Hot Encoding

Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

Import necessary libraries:

```
# import modules
from __future__ importprint_function

fromkeras.modelsimport Model
fromkeras.layersimport Input, LSTM, Dense
importnumpyas np
import pandas as pd
```

```
importgc
from matplotlib importpyplotasplt

Declare global variables :

batch_size = 128  # Batch size for training.
epochs = 100  # Number of epochs to train for.
latent_dim = 256  # Latent dimensionality of the encoding space.
num_samples = 684241
data_path = '/content/datasets/en_train_normalised.csv'
Separate input and target words and characters :
input_texts = [] # input sentences
target_texts = [] # output sentences
input_characters = set() # set of unique characters in input texts
target_characters = set() # set of unique characters in target texts

df = pd.read_csv(data_path) # load data into memory

k=0
for line in range(len(df)-1):
input_text=df.iloc[line][0]
target_text=df.iloc[line][1]
# We use "tab" as the "start sequence" character
# for the targets, and "\n" as "end sequence" character.
# to identify each word starting and ending positions
target_text = '\t' + target_text + '\n'
# insert word tokens in set
input_texts.append(input_text)
target_texts.append(target_text)

# insert each character in set
# since set will contain only unique characters
```

```
for char ininput_text:
if char notininput_characters:
input_characters.add(char)
for char intarget_text:
if char notintarget_characters:
target_characters.add(char)

# print few texts to justify it's working
if k<5:
print(input_text, target_text)
    k = k + 1
```

Output:

```
2006    two thousand six

IUCN  i u c n

2007    two thousand seven

2008    two thousand eight

91      ninety one
```

Print an overview of current data :

```
print(len(target_characters))  # 27
print(target_characters)  # a b c d /n /t special character
print(input_characters)  # all characters
print(target_texts)  # \t list of target entities \n
print(input_texts)  # list of source characters
```

Output :

```
350

{'Э', 'Я', '얄', 'ド’}

{'Э', 'Я', '얄', 'ド'}

['\ttwo thousand six\n', '\ti u c n\n', '\ttwo thousand seven\n']

['2006', 'IUCN', '2007']
```

Print an overview of current data :

```
input_characters = sorted(list(input_characters))
target_characters = sorted(list(target_characters))
num_encoder_tokens = len(input_characters)
num_decoder_tokens = len(target_characters)

max_encoder_seq_length = 44
max_decoder_seq_length = 45

print('Number of samples:', len(input_texts))
print('Number of unique input tokens:', num_encoder_tokens)
print('Number of unique output tokens:', num_decoder_tokens)
print('Max sequence length for inputs:', max_encoder_seq_length)
print('Max sequence length for outputs:', max_decoder_seq_length)

print('encoder_input_data', len(input_texts), max_encoder_seq_length, num_encoder_tokens)
print('decoder_input_data', len(input_texts), max_decoder_seq_length, num_decoder_tokens)
print('decoder_target_data', len(input_texts), max_decoder_seq_length, num_decoder_tokens)
```

Output :

```
Number of samples: 22637

Number of unique input tokens: 409

Number of unique output tokens: 350
```

```
Max sequence length for inputs: 44

Max sequence length for outputs: 45

encoder_input_data 22637 44 409

decoder_input_data 22637 45 350

decoder_target_data 22637 45 350
```

Assign each character a unique id :

```
encoder_input_data = np.zeros(
    (len(input_texts), max_encoder_seq_length, num_encoder_tokens),
dtype='float32')

decoder_input_data = np.zeros(
    (len(input_texts), max_decoder_seq_length, num_decoder_tokens),
dtype='float32')

decoder_target_data = np.zeros(
    (len(input_texts), max_decoder_seq_length, num_decoder_tokens),
dtype='float32')

print(encoder_input_data.shape)
print(decoder_input_data.shape)
print(decoder_target_data.shape)
```

Output:

```
(22637, 44, 409)

(22637, 45, 350)

(22637, 45, 350)
```

Enumerate over each character of each word and do one-hot-encoding :

```
k=0
fori, (input_text, target_text) in enumerate(zip(input_texts, target_texts)):
if k<20:
        print(input_text, target_text)
    k = k + 1
for t, char in enumerate(input_text):
if t <max_encoder_seq_length:
encoder_input_data[i, t, input_token_index[char]] = 1.
for t, char in enumerate(target_text):
# decoder_target_data is ahead of decoder_target_data by one timestep
ifi<10:
            print(i, t)
if t <max_decoder_seq_length:
decoder_input_data[i, t, target_token_index[char]] = 1.
if t > 0 and t <max_decoder_seq_length:
# decoder_target_data will be ahead by one timestep
# and will not include the start character.
decoder_target_data[i, t - 1, target_token_index[char]] = 1.
```

Output :

```
IUCN  i u c n


1 0

1 1

1 2
```

1 3

1 4

1 5

1 6

1 7

1 8

5.4 MODULE 3

- Compute train and validation loss over each epoch
- Update model parameters weight over each epoch

We use built in functions and classes of Tensorflow to build the model.

E.g.

- LSTM : RNN based learning layer
- rmsprop for optimization : This function called each time to update weights of parameters of LSTM
- categorical_crossentropy : This function is called after each epoch to calculate training and validation loss
- EarlyStopping : This function call builds a callback which is invoked after every epoch, if parameters are satisfied, the iteration is stopped before computing for each epoch.
- TensorBoard : Callback to store logs generated during training.
- Refer fig 2.

Define and compile the model :

```
# Define an input sequence and process it.
encoder_inputs = Input(shape=(None, num_encoder_tokens))
encoder = LSTM(latent_dim, return_state=True)
encoder_outputs, state_h, state_c = encoder(encoder_inputs)
# We discard `encoder_outputs` and only keep the states.
encoder_states = [state_h, state_c]
```

```
# Set up the decoder, using `encoder_states` as initial state.
decoder_inputs = Input(shape=(None, num_decoder_tokens))
# 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(decoder_inputs,
initial_state=encoder_states)
decoder_dense = Dense(num_decoder_tokens, activation='softmax')
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)
# Run training
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')
```

Set Callback :

```
tensorboard = TensorBoard(log_dir="logs/{}".format(time()))
# modify early stopping to save machine state when val loss is min
early = EarlyStopping(monitor='val_loss', min_delta=0.00001, verbose=1, patience=5)
```

Fit the data / Train the model

```
history = model.fit([encoder_input_data, decoder_input_data],
decoder_target_data,
batch_size=batch_size,
                epochs=epochs,
                verbose=1,
validation_split=0.1, callbacks=[tensorboard, early])
```

Save the model :

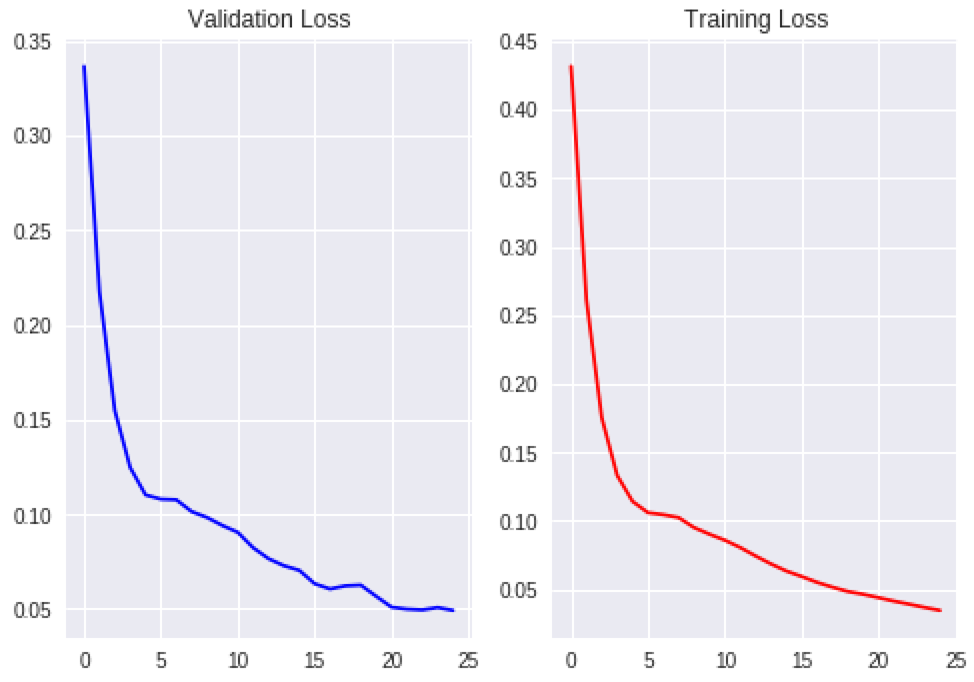
```
# Save model
model.save('s2s_demorun_demo_1.h5')
```

Output :

```
Train on 17999 samples, validate on 2000 samples

Epoch 1/25

17999/17999 [==============================] - 78s 4ms/step - loss: 0.4317 - val_loss:
0.3365

Epoch 2/25

17999/17999 [==============================] - 77s 4ms/step - loss: 0.2609 - val_loss:
0.2173

Epoch 3/25

17999/17999 [==============================] - 77s 4ms/step - loss: 0.1752 - val_loss:
0.1548

Epoch 4/25

17999/17999 [==============================] - 77s 4ms/step - loss: 0.1331 - val_loss:
0.1247

Epoch 5/25

17999/17999 [==============================] - 77s 4ms/step - loss: 0.1142 - val_loss:
0.1103

.

.

.

Epoch 21/25
```

```
17999/17999 [==============================] - 77s 4ms/step - loss: 0.0438 - val_loss: 0.0510

Epoch 22/25

17999/17999 [==============================] - 77s 4ms/step - loss: 0.0413 - val_loss: 0.0499

Epoch 23/25

17999/17999 [==============================] - 77s 4ms/step - loss: 0.0390 - val_loss: 0.0495

Epoch 24/25

17999/17999 [==============================] - 77s 4ms/step - loss: 0.0367 - val_loss: 0.0508

Epoch 25/25

17999/17999 [==============================] - 77s 4ms/step - loss: 0.0345 - val_loss: 0.0492
```

Fig 2: Plot statistical data of Training Loss and Validation Loss :



5.5 MODULE 4

Predict normalized text output :

- Encode input and retrieve decoder state.
- Run one step of decoder with its initial state and start of a sequence token as target
- Since this is LSTM, output will be the next target token
- Repeat with the current target token and current states (refer fig 3).

Encode input and retrieve decoder state :

```
encoder_model = Model(encoder_inputs, encoder_states)
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]
decoder_outputs, state_h, state_c = decoder_lstm(
decoder_inputs, initial_state=decoder_states_inputs)
decoder_states = [state_h, state_c]
decoder_outputs = decoder_dense(decoder_outputs)
decoder_model = Model(
    [decoder_inputs] + decoder_states_inputs,
    [decoder_outputs] + decoder_states)
```

Convert id to char so that it isin readable form :

```
# Reverse-lookup token index to decode sequences back to something readable.
reverse_input_char_index = dict(
    (i, char) for char, iininput_token_index.items())
reverse_target_char_index = dict(
    (i, char) for char, iintarget_token_index.items())
```

We use a Open Source public code available on github to decode the text given

Input_text in encoded form as input and encoder and decoder states:

```
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, num_decoder_tokens))
# Populate the first character of target sequence with the start character.
target_seq[0, 0, target_token_index['\t']] = 1.

# Sampling loop for a batch of sequences
# (to simplify, here we assume a batch of size 1).
stop_condition = False
decoded_sentence = ''
whilenotstop_condition:
output_tokens, h, c = decoder_model.predict(
          [target_seq] + states_value)

# 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

# Exit condition: either hit max length
# or find stop character.
if (sampled_char == '\n'or
len(decoded_sentence) >max_decoder_seq_length):
stop_condition = True
# Update the target sequence (of length 1).
target_seq = np.zeros((1, 1, num_decoder_tokens))
target_seq[0, 0, sampled_token_index] = 1.

# Update states
states_value = [h, c]
returndecoded_sentence
```

```
  Predict or infer normalized text:
  forseq_indexin range(50):
  # Take one sequence (part of the training test)
  # for trying out decoding.
  input_seq = encoder_input_data[seq_index: seq_index + 1]
  decoded_sentence = decode_sequence(input_seq)
     print('-')
     print('Input sentence:', input_texts[seq_index])
     print('Decoded sentence:', decoded_sentence)
```

Output :

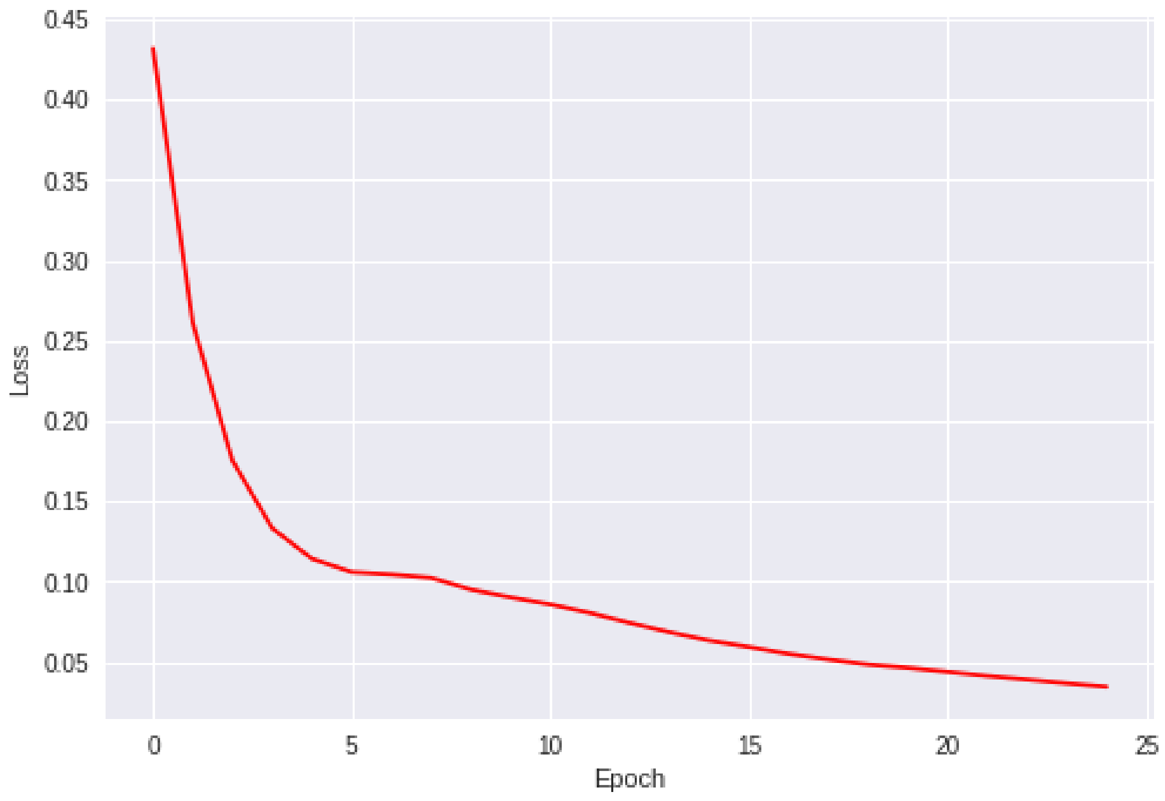
```
Input sentence: 2006

Decoded sentence: two thousand six

-

Input sentence: 2008

Decoded sentence: two thousand eight

-

Input sentence: 1987

Decoded sentence: nineteen eighty seven

-

Input sentence: 4

Decoded sentence: four

-

Input sentence: 1595

Decoded sentence: fifteen ninety five

-

Input sentence: "August 16, 2005"

Decoded sentence: august twenty first twenty fifteen
```

Due to hardware restriction we were not able to train our model over large dataset (over 50K),so there are some wrong predictions also, specially in dates. But we trained upto 20K sentences. However these results are really surprising, with very small dataset it gave one out of every fifth prediction correct with 100 percent accuracy.

Plot statistical data:

```
plt.plot(history.history['loss'], 'r')
plt.show()
```

Output :Fig 3:



6. RESULT ANALYSIS

In this work, we showed that a large deep LSTM with a limited vocabulary can outperform many other approaches e.g. Bag of Words, Contextual prediction etc. The success of our simple LSTM-based approach suggests that it should do well on many other sequence learning problems, provided they have enough training data.

The extent of the improvement obtained by reversing the words in the source sentences was not unexpected. We concluded that it is important to find a problem encoding that has the greatest number of short term dependencies, as they make the learning problem much simpler. In particular, while we were unable to train a standard RNN on the non-reversed translation problem , we believe that a standard RNN should be easily trainable when the source sentences are reversed (although we did not verify it experimentally).

Due to hardware restriction we were not able to train our model over large dataset (over 100 K),so there are some wrong predictions also, specially in dates. But we trained upto 20K sentences. However with very small dataset it gave one out of every fifth prediction correct with 100 percent accuracy.

7. CONCLUSION

In this project we have proposed a model for the task of normalization. We presented a context aware classification model and how we used it to clear out "noisy" samples. We discussed different approaches for normalizing text. We share our insights and analysis like where our models may shine and where we can improve. We also list out possible ways of improving the gaps identified.

8. REFERENCES

1. https://keras.io/layers/recurrent/#lstm
2. https://keras.io/getting-started/faq/
3. https://www.tensorflow.org/api_docs/python/tf/keras/models/Model#fit
4. http://colah.github.io/posts/2015-08-Understanding-LSTMs/
5. http://blog.echen.me/2017/05/30/exploring-lstms/
6. Wikipedia
7. https://classroom.udacity.com/courses/ud188/
8. Kaggle
9. Research Papers (arxiv.org)
10. Text Normalization-Frank Kane
11. Introduction to information retrieval
12. Textbook by Christopher D. Manning, HinrichSchütze, and Prabhakar Raghavan