



**NANYANG  
TECHNOLOGICAL  
UNIVERSITY**  

---

**SINGAPORE**

**CE/CZ4042**

**Neural Network & Deep Learning**

**The Next “Singlish” Sentence Generator**

**Date: 22 November 2020**

## Table of Contents

<b>Miscellaneous</b> .....	<b>4</b>
<b>Introduction</b> .....	<b>8</b>
Brief Introduction to our project.....	8
What is NLG?.....	8
Limitations of the current NLG models available.....	8
Aim of our project: The Next “Singlish” Sentence Generator .....	8
Project Pipeline: Overview.....	9
Project Phases: How it is executed in each phase .....	10
<b>Phase 1: Data Scraping</b> .....	<b>11</b>
Rationale of scraping.....	11
The Scraper Code: Phase1-FBScraper.py .....	11
Demonstration .....	12
Data scraped in this project.....	14
Challenges faced in this phase .....	15
<b>Phase 2: Data Cleaning/Pre-processing</b> .....	<b>16</b>
The need to do Data Cleaning .....	16
The Data Cleaning/Pre-processing code: Phase2-data-pre-processing.ipynb .....	16
Challenges faced in this phase .....	20
<b>Phase 3: Basic Exploratory Data Analysis</b> .....	<b>21</b>
EDA Code: Phase3-EDA.ipynb.....	21
<b>Phase 4a: Training of Model (LSTM &amp; GRU)</b> .....	<b>24</b>
LSTM.....	24
GRU.....	25
Models trained in the experiments.....	25
Why stacked LSTM or GRU architecture .....	26
Helper functions used in this phase .....	26
Implementation: Phase4a-next-sentence-generator.ipynb .....	27
Preparing the data.....	27
Building and training the model .....	29
Performance of each models.....	30
Text generation for a given input sentence .....	31
Results of the text generation .....	32
Choice of models for POS tagging and insertion of Singlish words in Phase 5 .....	33

<b>Phase 5: POS tagging and insertion of Singlish words .....</b>	<b>34</b>
What is POS tagging .....	34
Motivation of this phase.....	34
Helper functions used in this phase .....	35
Implementation: Phase5-POS-insertion-Singlish-words.ipynb .....	36
Preparing the data & loading of the model.....	36
Insertion of Singlish words into the generated sentence.....	36
Result of the finalised Singlish sentence.....	36
Challenges faced in this phase .....	42
Conclusion of the results.....	42
<b>BONUS PHASE - Phase 4b: Training of Model (GPT-2).....</b>	<b>43</b>
What is a transformer? .....	43
What is GPT-2 (Generative Pretrained Transformer 2)? .....	43
Implementing GPT-2 into our project .....	44
Result from the fine-tuned GPT-2 .....	46
Conclusion of the results.....	48
<b>Conclusion for this project .....</b>	<b>49</b>
<b>References.....</b>	<b>50</b>

# Miscellaneous

## Libraries/modules used in this project

### 1. Phase 1: Data Scraping

- `xlsxwriter`
- `facebook_scraper` (`get_posts`)

### 2. Phase 2: Data Cleaning/Pre-processing

- `pandas`
- `os`
- `requests`
- `urllib`
- `xlrd`
- `re`

### 3. Phase 3: Basic Exploratory Data Analysis

- `wordcloud` (`WordCloud`, `ImageColorGenerator`)
- `matplotlib.pyplot`
- `pandas`
- `string`
- `PIL` (`Image`)
- `numpy`
- `os`
- `itertools` (`islice`)
- `sklearn.feature_extraction.text` (`CountVectorizer`, `TfidfTransformer`)

### 4. Phase 4a: Training of Model (LSTM & GRU)

- `numpy`
- `string`
- `tensorflow`
- `time`

### 5. Phase 4b: Training of Model (Attention: GPT-2)

- `gpt_2_simple`
- `datetime`

## 6. Phase 5: POS tagging and insertion of Singlish words

- nltk
- string
- tensorflow
- random

### Coded in

- Python

### Software used

- Spyder & Visual Studio Code IDE - Phase 1
- Jupyter Notebook - Phase 2
- Google Colaboratory (GPU instance) - Phase 3, 4a & 4b

### Files included in this project (File Structure)

- Project\_Report.pdf
- Project\_Codes.zip
  - Phase1-FBScraper.py
  - Phase2-data-pre-processing.ipynb
  - Phase3-EDA.ipynb
  - Phase4a-next-sentence-generator.ipynb
  - Phase4b-next-sentence-generator-gpt2.ipynb
  - Phase5-POS-insertion-Singlish-words.ipynb
  - finalData/
    - SgCorpus.txt
  - saved\_weights/
    - 2LAYER\_GRU-100\_epoch-64\_batch\_size-Adam.csv
    - 2LAYER\_GRU-100\_epoch-64\_batch\_size-Adam.h5
    - 2LAYER\_GRU-100\_epoch-64\_batch\_size-RMSprop.csv
    - 2LAYER\_GRU-100\_epoch-64\_batch\_size-RMSprop.h5
    - 2LAYER\_GRU-100\_epoch-128\_batch\_size-Adam.csv
    - 2LAYER\_GRU-100\_epoch-128\_batch\_size-Adam.h5
    - 2LAYER\_GRU-100\_epoch-128\_batch\_size-RMSprop.csv
    - 2LAYER\_GRU-100\_epoch-128\_batch\_size-RMSprop.h5
    - 2LAYER\_LSTM-1\_epoch-128\_batch\_size-Adam.csv
    - 2LAYER\_LSTM-1\_epoch-128\_batch\_size-Adam.h5
    - 2LAYER\_LSTM-20\_epoch-128\_batch\_size-Adam.csv
    - 2LAYER\_LSTM-20\_epoch-128\_batch\_size-Adam.h5
    - 2LAYER\_LSTM-20\_epoch-128\_batch\_size-SGD.csv

- 2LAYER\_LSTM-20\_epoch-128\_batch\_size-SGD.h5
- 2LAYER\_LSTM-100\_epoch-64\_batch\_size-Adam.csv
- 2LAYER\_LSTM-100\_epoch-64\_batch\_size-Adam.h5
- 2LAYER\_LSTM-100\_epoch-64\_batch\_size-RMSprop.csv
- 2LAYER\_LSTM-100\_epoch-64\_batch\_size-RMSprop.h5
- 2LAYER\_LSTM-100\_epoch-128\_batch\_size-Adam.csv
- 2LAYER\_LSTM-100\_epoch-128\_batch\_size-Adam.h5
- 2LAYER\_LSTM-100\_epoch-128\_batch\_size-RMSprop.csv
- 2LAYER\_LSTM-100\_epoch-128\_batch\_size-RMSprop.h5
- scrapedData/
  - scrapingData\_ACJC-Confessions-365341810240423\_FULL.xlsx
  - scrapingData\_andiechen\_FULL.xlsx
  - scrapingData\_asrjccconfessions\_FULL.xlsx
  - scrapingData\_bellywellyjelly\_FULL.xlsx
  - scrapingData\_benjamin.kheng\_FULL.xlsx
  - scrapingData\_bossyflossie\_FULL.xlsx
  - scrapingData\_cjcroxx\_FULL.xlsx
  - scrapingData\_DanielFoodDiary\_FULL.xlsx
  - scrapingData\_DHS-Confessions-103690209814932\_FULL.xlsx
  - scrapingData\_DollarsAndSenseSG\_FULL.xlsx
  - scrapingData\_dreachongofficial\_FULL.xlsx
  - scrapingData\_HwaChongConfessions\_FULL.xlsx
  - scrapingData\_ieatishootipost\_FULL.xlsx
  - scrapingData\_InnovaConfessions\_FULL.xlsx
  - scrapingData\_ITE-College-Central-Confessions-102332676616681\_FULL.xlsx
  - scrapingData\_ITE-College-West-Confessions-123845157793064\_FULL.xlsx
  - scrapingData\_JjcConfessions\_FULL.xlsx
  - scrapingData\_ladyironchef\_FULL.xlsx
  - scrapingData\_moneysmartsg\_FULL.xlsx
  - scrapingData\_mongabong\_FULL.xlsx
  - scrapingData\_mrbrownlah\_FULL.xlsx
  - scrapingData\_mykxii\_FULL.xlsx
  - scrapingData\_NJC-Confessions-414721038609037\_FULL.xlsx
  - scrapingData\_npconfession\_FULL.xlsx
  - scrapingData\_NTUConfess\_FULL.xlsx
  - scrapingData\_nuswhispers\_FULL.xlsx
  - scrapingData\_NYP-Confessions-118332335013023\_FULL.xlsx
  - scrapingData\_RepublicPolyConfessions\_FULL.xlsx
  - scrapingData\_RJConfessions\_FULL.xlsx
  - scrapingData\_SajcConfessions\_FULL.xlsx
  - scrapingData\_sethluimarketing\_FULL.xlsx

- scrapingData\_SGAG\_FULL.xlsx',
- scrapingData\_simconfessions\_FULL.xlsx',
- scrapingData\_SMUConfessionsPage\_FULL.xlsx',
- scrapingData\_SP-Confessions-329564370479100\_FULL.xlsx
- scrapingData\_SUSSConfessions\_FULL.xlsx
- scrapingData\_therealnaomineo\_FULL.xlsx
- scrapingData\_TheSmartLocal\_FULL.xlsx
- scrapingData\_TJC-Confessions-133149643521172\_FULL.xlsx
- scrapingData\_tpconfession\_FULL.xlsx
- scrapingData\_yoyokulala\_FULL.xlsx
- scrappingData\_MemedefSG\_FULL.xlsx
- scrappingData\_SGAG.xlsx
- scrappingData\_SGAG\_FULL.xlsx
- scrappingData\_singlish101\_FULL.xlsx
- README.md
- singaporeMap.jpg

# Introduction

## Brief introduction to our project

In this project, we will be focusing on natural language generation (NLG) problems from the Singapore context. Mainly, we will be focusing on how we can generate sentences from a corpus with Singapore context in it. Then, we will do some hand engineering on the generated text by including the most common singlish words used in Singapore with the help of POS (part-of-speech) tagging.

## What is NLG?

NLG is a subsection of Natural Language Processing (NLP). NLG software turns structured data into written narrative, writing like a human being. NLG makes data universally understandable and seeks to automate the writing of data-driven narratives like financial reports, product descriptions, meeting memos, and more.

## Limitations of the current NLG models available

There are many implementations related to sentence generation on the internet. However, most of them were trained with default datasets, such as Shakespeare's "sonnets.txt", or ready-made twitter dataset. From the current implementations available, there are only very few datasets available online that are in the Singapore context, particularly, Singapore text data with some singlish context in it. There was one suitable corpus that we could use for this project which is the Singapore Short Message Service (SMS) corpus from the National University of Singapore (NUS). However, there were only 35,598 english SMSes, with only a limited number of unique tokens available, resulting in our text generation not being diverse if we were to use this corpus. This gives us the motivation to scrape our own Singapore dataset with some essence of Singlish inside.

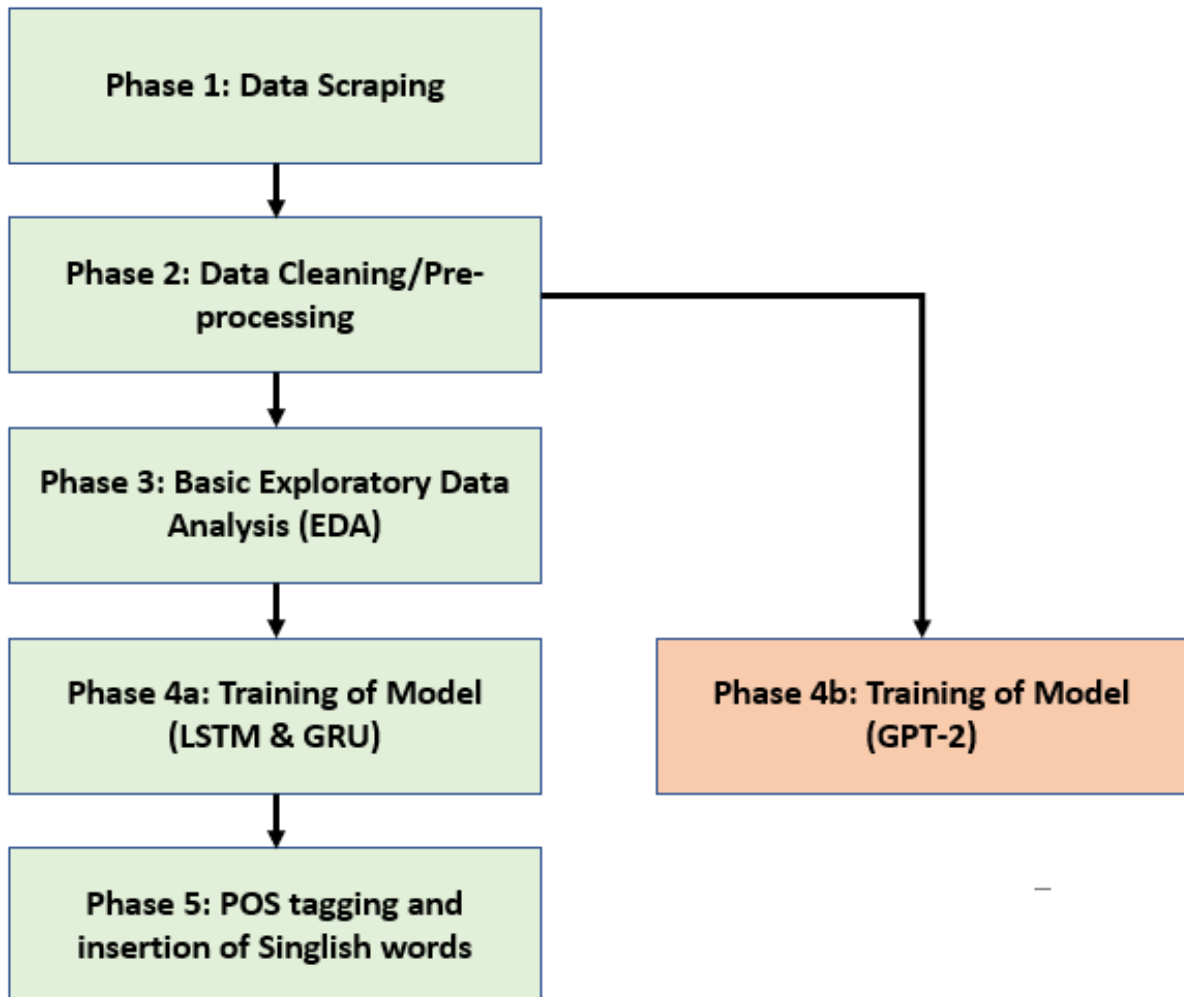
## Aim of our project: The Next "Singlish" Sentence Generator

Our project aims to generate coherent sentences given a particular start words, phrases or sentences, based on the various models trained with our scraped text corpus. We hope that the sentences generated will have Singapore sentiments and also some traces of Singlish. We break our project into 5 phases, from the start with data scraping to the last phase with POS tagging of Singlish words to the generated sentences.



## Project Pipeline: Overview

Our project pipeline is as shown below.



We will first go through Phase 1, then Phase 2, Phase 3, Phase 4a, Phase 5 and then go back to Phase 4b.

## **Project Phases: How it is executed in each phase**

These are the brief explanations of how these phases were implemented:

### **Phase 1: Data Scraping**

To do web scraping of texts from Singapore Facebook pages' posts.

### **Phase 2: Data Cleaning/Pre-processing**

To combine the individual scraped .xlsx file into one big data file, then do data cleaning using regular expression (regex) and save the 'cleaner' data as SgCorpus.txt for the next phase.

### **Phase 3: Basic Exploratory Data Analysis (EDA)**

To perform some simple data analysis like term frequency–inverse document frequency (tfidf) and WordCloud for visualisation.

### **Phase 4a: Training of Model (LSTM & GRU)**

To build a few models using the text corpus we have scraped and cleaned in Phase 1 and Phase 2 respectively to analyse which model can generate the most coherent sentences.

### **Phase 5: POS tagging and insertion of Singlish words**

To perform some hand engineering on the generated text and insertion of singlish words.

### **BONUS - Phase 4b: Training of Model (GPT-2)**

To fine-tune the GPT-2 (small version) in Google Colaboratory such that we are able to train our model using the corpus we have scraped, and generate the sentence based on a given start string of words.

# Phase 1: Data Scraping

## Rationale of scraping and forming our text corpus

The main rationale of scraping data is due to the lack of representative corpus online. Thus, we scraped our own data, hoping to generate text with the Singapore or Singlish sentiment. Thus, we wrote a code named Phase1-FBScraper.py to allow us to automate this process of retrieving the text data we need to build up our corpus.

## The Scraper Code: Phase1-FBScraper.py

We wrote a simple scraper code by using an Application Programming Interface (API) named 'facebook\_scraper'. We used the 'get\_posts' function to scrape facebook posts from various pages written by famous Singaporean authors and also confession pages by various Singapore institutions. The rationale of scraping these data is that we want the text data to be from a more 'informal' context as compared to official webpages, such as e-newspapers, food review web pages etc. which uses a more 'formal' english to write. We believe that the opinion-based Facebook posts written by Singapore social media users will give us a better approach in generating texts of how a normal Singaporean would speak.

We first take in the input of the users. We will take in two inputs: the target Facebook page we want to scrape and how many pages we want to scrape. The number of pages here refers to how many posts the webpage can show without you scrolling down to check out on new posts. The code to prompt for input is as shown below:

```
# Prompt for input
targetPage = input("Target facebook page to scrape: ")
noOfPages = int(input("Number of pages: "))
```

We then saved the scraped texts of a Facebook page into a .xlsx file by initialising the .xlsx file as shown below:

```
# Set up workbook/worksheet
workbook = xlswriter.Workbook('./scrapedData/scrapingData_'+str(targetPage)+'_FULL.xlsx')
worksheet = workbook.add_worksheet()
```

We named our data files according to the name of our target facebook pages so as to show consistency in the naming of files.

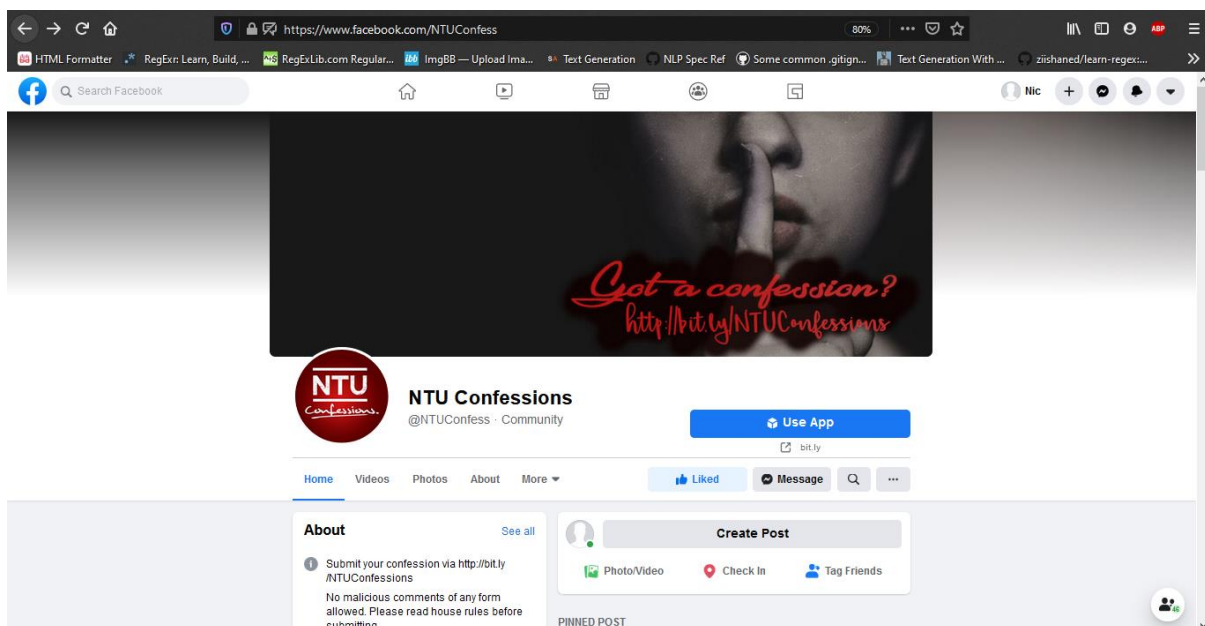
Then, the text data will be scraped with the following code as shown below:

```
# Search for the posts
for post in get_posts(str(targetPage), pages=noOfPages):
    # write operation perform
    try:
        if (len(post['text'])): # Check length of the post
            print("Scraping post #" + str(row) + ": ")
            try:
                print(post['text'])
            except:
                print("Failed to print text")
            try:
                worksheet.write(row, column, post['text'])
                # incrementing the value of row by one
                # with each iterations.
                row += 1
                print("-----Post saved-----")
            except:
                print("-----Invalid post! Failed to save post-----")
            print("")
        except:
            print("-----Post has no message. Skipping to the next post-----")
```

As seen above, there are try and except blocks. This is because the data scraped might contain text with unsupported format (e.g. Chinese characters and emojis) which will generate error in the middle of the scraping process, hence we will want to avoid this situation by skipping the unsupported format text.

## Demonstration

We will take the Facebook page “NTU Confession” as an example of how the scraping of text is done.



We first copied the endpoint of the webpage as shown below:



This endpoint will be the target page input for our program. We will run our program and input the endpoint “NTUConfess” into our program. For demonstration purposes, we will only scrape one page of data. The inputs are as shown below:

```
In [2]: runfile('D:/next-sentence-predictor/FBScraper.py',
wdir='D:/next-sentence-predictor')

Target facebook page to scrape: NTUConfess

Number of pages: 1
```

The sample output is as shown:

```
Scraping post #0:
"To the guy who stays in Crescent Hall , don't know which room.
Eh bro, if your alarm can't wake you up, turn in off lah! Your alarm sibeh noisy you know? Pls lah."

Submitted Tuesday, 17/11
#NTUConfessions29717
-----Post saved-----

Scraping post #1:
"Responding to #NTUConfessions29729, I am a year 3 film student from ADM and I have also heard rumours about the girl who was bullied from the seniors, who are in the same batch as her.

I just want to say that it is not that easy to ignore rumours in this day and age and we can be easily misguided. I dunno about other schools... but in filmmaking we kinda have to help one another's projects and it is best not to burn bridges. When the seniors asked us to help on their film projects, normally we as juniors will go and help out to gain experience. Many times we hear conflicts that happen between classmates, and we tend to listen to the majority? There are also times when I realised that group of seniors are really toxic, but there's nothing I can do about it because I'm helping them out for their project and they are my seniors, who have an authority over school facilities and equipments... As selfish as it sounds, most of us just want to turn it into a blind eye and act blur lol

Another thing that I am trying to say is that because of the manipulative clique of seniors, maybe many other people were misguided by the impression of the victim. I also thought the victim was what that clique of students said, and because I don't know the victim personally, I assumed that she was like this and went along with it. I think a lot of people were in the same situation as I am, and soon it escalated because nobody addressed the rumours and confronted the clique (maybe too scared to talk to them).

I really dunno much about what happened but this is just my view, I could be wrong :/"

Submitted Friday, 20/11
#NTUConfessions29741
-----Post saved-----

Successfully saved as scrapingData_NTUConfess_FULL.xlsx
Total rows: 1
```

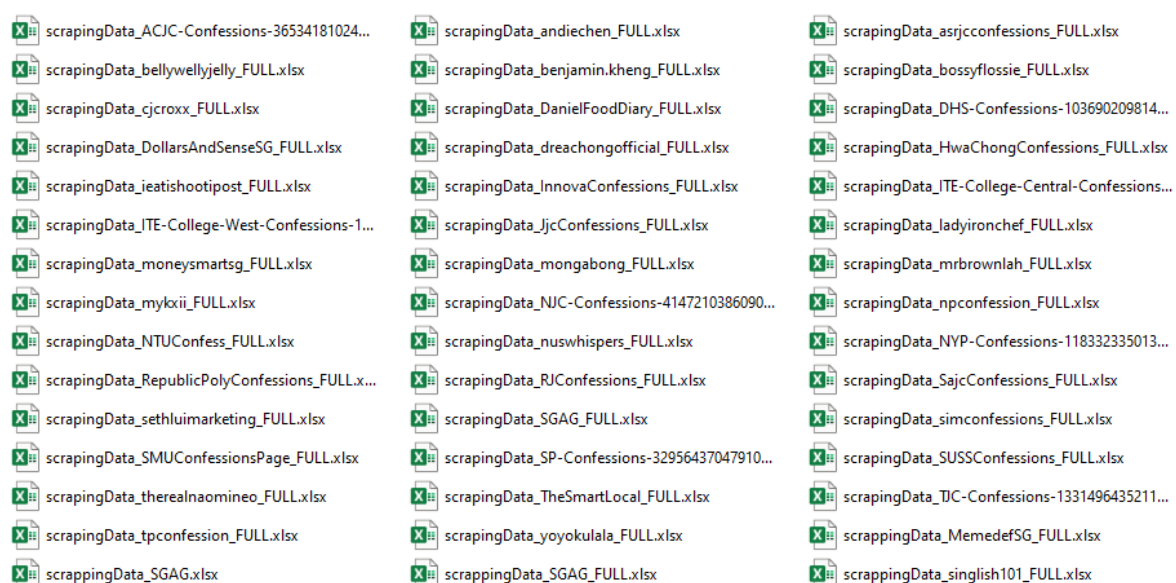
And the .xlsx file data is stored as shown below (for one page):

	"To the guy who stays in Crescent Hall , don't know which room. Eh bro, if your alarm can't wake you up, turn in off lah! Your alarm sibe noisy you know? Pls lah."
	Submitted Tuesday, 17/11
1	#NTUConfessions29717
	"Responding to #NTUConfessions29729, I am a year 3 film student from ADM and I have also heard rumours about the girl who was bullied from the seniors, who are in the same batch as her.
	I just want to say that it is not that easy to ignore rumours in this day and age and we can be easily misguided. I dunno about other schools... but in filmmaking we kinda have to help one another's projects and it is best not to burn bridges. When the seniors asked us to help on their film projects, normally we as juniors will go and help out to gain experience. Many times we hear conflicts that happen between classmates, and we tend to listen to the majority? There are also times when I realised that group of seniors are really toxic, but there's nothing I can do about it because I'm helping them out for their project and they are my seniors, who have an authority over school facilities and equipments... As selfish as it sounds, most of us just want to turn it into a blind eye and act blur lol
	Another thing that I am trying to say is that because of the manipulative clique of seniors, maybe many other people were misguided by the impression of the victim. I also thought the victim was what that clique of students said, and because I don't know the victim personally, I assumed that she was like this and went along with it. I think a lot of people were in the same situation as I am, and soon it escalated because nobody addressed the rumours and confronted the clique (maybe too scared to talk to them).
	I really dunno much about what happened but this is just my view, I could be wrong :/"

Typically, we scraped around 10000 to 15000 pages per Facebook page.

## Data scraped in this project

The data (.xlsx files) that were scraped and used to build our corpus are as shown below in this file directory:



## **Challenges faced in this phase**

1. The data we scraped may contain unwanted data such as hashtags, dates, emojis etc., which may affect our text generator. One solution will be to use regular expressions (regex) to remove all the unwanted text which will be demonstrated in the data cleaning/pre-processing phase in Phase 2.
2. Although the data scraped has the 'informal' way of writing by the various authors, there are very few Singlish texts seen in the data scraped as compared to non-Singlish words, as such this may badly affect the text generator to generate texts that contain Singlish words in it. One solution is just to generate the text by using the scraped text corpus, then do a POS tagging on the text generated and then hand engineer the output by inserting the Singlish words according to the POS tags. This will be demonstrated in the POS tagging and insertion of English words phase (Phase 5).

## Phase 2: Data Cleaning/Pre-processing

### The need to do Data Cleaning

The data we scraped are very 'dirty'. It has a mixture of wanted and unwanted texts. As such, we will have to do some data cleaning in order to sift out the data we need to train our model. An example of 'dirty' data and unwanted texts are as shown below:

#### The presence of text emoji

Why don't they let the crashers play orientation games this year? Suddenly so strict :(

#### The presence of unsupported text format

Anyone knows that sassy indian girl from Neytiri? She is so cute ❤️❤️

#### The presence of numbers and links

To the dude who said that the probability of you passing SBJ being  $3.78 \times 10^{-4}$  being not bad, you do realise that you have to jump approximately 2.8 thousand times to have a chance of passing SBJ

Sassy Admin:

<http://i.qkme.me/3tgcs8.jpg>

I.QKME.ME

<http://i.qkme.me/3tgcs8.jpg>

Hence, we will need to remove all the unwanted texts as this will affect the generated texts.

### The Data Cleaning/Pre-processing code: Phase2-data-pre-processing.ipynb

We will first pre-process the data we have scraped in Phase 1. Due to resource limitations in Google Colaboratory, we have to combine our data into two .xlsx files first (instead of just one) then we do the cleaning of data. We only pre-processed one .xlsx file in one runtime. Afterwards, we combined the two files and formed the final text corpus.



Firstly, we appended the file name into a list so as for the next step of combining the data as shown below:

```
tempList = os.listdir('scrapedData')
# display individual .xlsx names in the ScrapedData sub-folder
fileList = list()
for file in tempList:
    fileList += [BASE + file]

fileList

['./scrapedData/scrapingData_ACJC-Confessions-365341810240423_FULL.xlsx',
 './scrapedData/scrapingData_andiechen_FULL.xlsx',
 './scrapedData/scrapingData_asrjcconfessions_FULL.xlsx',
 './scrapedData/scrapingData_bellywellyjelly_FULL.xlsx',
 './scrapedData/scrapingData_benjamin.kheng_FULL.xlsx',
```

To combine our data, we wrote a function name 'excelCombiner' to take in the lowest and the highest index of the list so as to combine the .xlsx files together as shown below:

```
# Combining excel files in bigger batches
def excelCombiner(idxLow,idxHigh):

    #Create a dataframe
    df = pd.DataFrame()

    # read them in
    excels = [pd.ExcelFile(item) for item in fileList[idxLow:idxHigh]]

    # turn them into dataframes
    frames = [x.parse(x.sheet_names[0], header=None,index_col=None) for x in excels]

    # delete the first row for all frames except the first
    # i.e. remove the header row -- assumes it's the first
    frames[1:] = [df[1:] for df in frames[1:]]

    # concatenate them..
    combined = pd.concat(frames)

    return combined
```

We called the 'excelCombiner' function and saved the combined data into two bigger batches of data in .xlsx format as shown below.

```
[ ] # Batch 1
    excelBatch1 = excelCombiner(0,34)
    excelBatch1.to_excel(TEMP+"excelBatch1.xlsx", header=False, index=False)

[ ] # Batch 2
    excelBatch2 = excelCombiner(34,len(fileList)+1)
    excelBatch2.to_excel(TEMP+"excelBatch2.xlsx", header=False, index=False)
```

To load the larger batch files and also to get the information of the files another function named 'xlsxBatchInfo' is initialised and it took in the sub-folder and the batch file name as parameters and it returned the file itself and the number of rows each batch file has as shown below.

```
# details of individual batches of .xlsx
def xlsxBatchInfo(subFolder,xlsxBatchFile):
    file = subFolder + xlsxBatchFile
    workbook = xlrd.open_workbook(file)
    sheet = workbook.sheet_by_index(0)
    # check file directory
    #print(file)
    # check number of rows in the scraped text corpus in this batch
    #print(sheet.nrows)
    return file, sheet.nrows

[ ] # call the details of batch1 excel file
    batch1, noOfRows1 = xlsxBatchInfo(TEMP,"excelBatch1.xlsx")
    print(batch1)
    print(noOfRows1)

    ./tempData/excelBatch1.xlsx
    64003

[ ] # call the details of batch2 excel file
    batch2, noOfRows2 = xlsxBatchInfo(TEMP,"excelBatch2.xlsx")
    print(batch2)
    print(noOfRows2)

    ./tempData/excelBatch2.xlsx
    18923
```

After loading the batch files, we proceeded to do our data cleaning on both the batch files, which is defined with the function named 'cleanedData'. We used regular expressions (regex) to clean our data. These were the aspects that we took into account when cleaning the data:

1. Remove chinese characters and emojis
2. Remove hashtags
3. Remove websites
4. Remove tags (surrounded by squared brackets [ ])
5. Remove angular brackets < >
6. Replace \n with full stop
7. Remove date and time
8. Remove month
9. Remove parentheses ( )
10. Remove mentions @
11. Remove equal signs =
12. Replace trailing punctuations with just one
13. Replace forward slash with a space
14. Remove underscore
15. Remove emoticons like :) and <3 etc
16. Replace contractions with proper sentence word e.g. don't with do not
17. Remove quotation marks
18. Replace trailing spaces with just one

Some examples of the texts after it was being cleaned are as shown below:

```
---After encoding/decoding at row 18631
every. single. time.

---After encoding/decoding at row 18632
tampines mrt station users will know this feel

---After encoding/decoding at row 18633
buy this car, they said.you will look so cool. they said.
```

We then saved the texts in .txt format after cleaning for each of the batches, using a function named 'saveTxt' as shown below:

```
[ ] ### Save batch data as .txt file
def saveTxt(inputFile,outputFile,destFolder):
    with open(destFolder+outputFile, 'w') as f:
        for item in inputFile:
            f.write("%s\n" % item)
```

Lastly, we merged the two batch files and formed our final corpus named 'SgCorpus.txt' as shown below:

```
[ ] import glob

read_files = glob.glob("./tempData/*.txt")

with open("./finalData/SgCorpus.txt", "wb") as outfile:
    for f in read_files:
        with open(f, "rb") as infile:
            outfile.write(infile.read())
```

### Challenges faced in this phase

1. Some unwanted data might not be able to use regex to remove it due to the formatting issues of the different texts being scraped.
2. There are too many unwanted data (cases) to be sifted out, hence there might still be a chance that you will see some unwanted texts.

## Phase 3: Basic Exploratory Data Analysis

### EDA Code: Phase3-EDA.ipynb

In this phase, we will do some simple exploratory data analysis (EDA) to get some insights of the data that was scraped and cleaned in Phase 1 and Phase 2. We will perform two types of analysis here, they are the WordCloud and the term frequency–inverse document frequency (tfidf). But in order to do so, we will need to further process our data by removing punctuations and to split the corpus into individual word tokens. This can be illustrated in the code snippet as shown below:

```
# turn a doc into clean tokens
def clean_doc(doc):
    # replace '--' with a space ' '
    doc = doc.replace('--', ' ')
    # remove punctuation from each token
    table = str.maketrans('', '', string.punctuation)
    # split into tokens by white space
    tokens = doc.split()

    tokens = [w.translate(table) for w in tokens]
    # remove remaining tokens that are not alphabetic
    tokens = [word for word in tokens if word.isalpha()]
    # make lower case
    tokens = [word.lower() for word in tokens]
    return tokens

[ ] # clean document
tokens = clean_doc(doc)
print(tokens[:50])
```

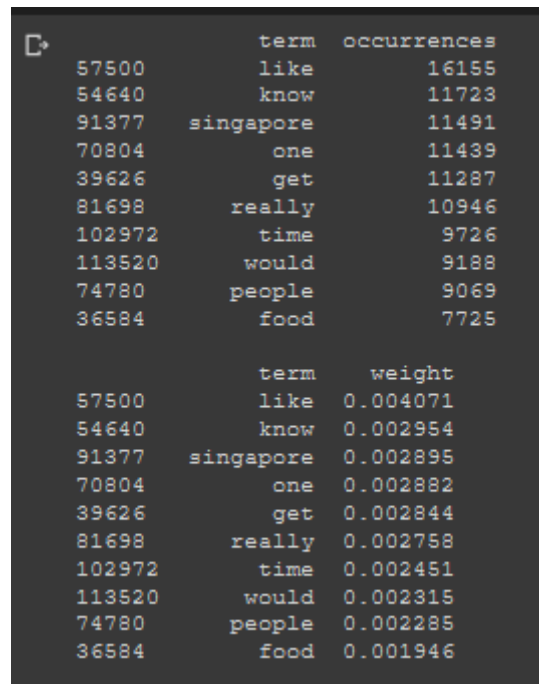
We also need to remove the stopwords in our corpus in order to get the word cloud that is representative of the text we scraped. For this, we made use of the nltk library to track the stopwords and stored them into a list variable named 'stopwords' so that it can be used for the word cloud and TF-IDF analysis. A code snippet of the operation is as shown below:

```
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
print(stopwords.words('english'))

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Unzipping corpora/stopwords.zip.
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",
<
[ ] stopwords = stopwords.words('english')
```



We also performed the TF-IDF analysis on our dataset and the result is as shown below:

A terminal window with a dark background and light gray text. It displays two tables of TF-IDF analysis results. The first table has columns 'term' and 'occurrences', and the second has columns 'term' and 'weight'. Both tables list the same ten terms, sorted by their respective values in descending order.

	term	occurrences
57500	like	16155
54640	know	11723
91377	singapore	11491
70804	one	11439
39626	get	11287
81698	really	10946
102972	time	9726
113520	would	9188
74780	people	9069
36584	food	7725

	term	weight
57500	like	0.004071
54640	know	0.002954
91377	singapore	0.002895
70804	one	0.002882
39626	get	0.002844
81698	really	0.002758
102972	time	0.002451
113520	would	0.002315
74780	people	0.002285
36584	food	0.001946

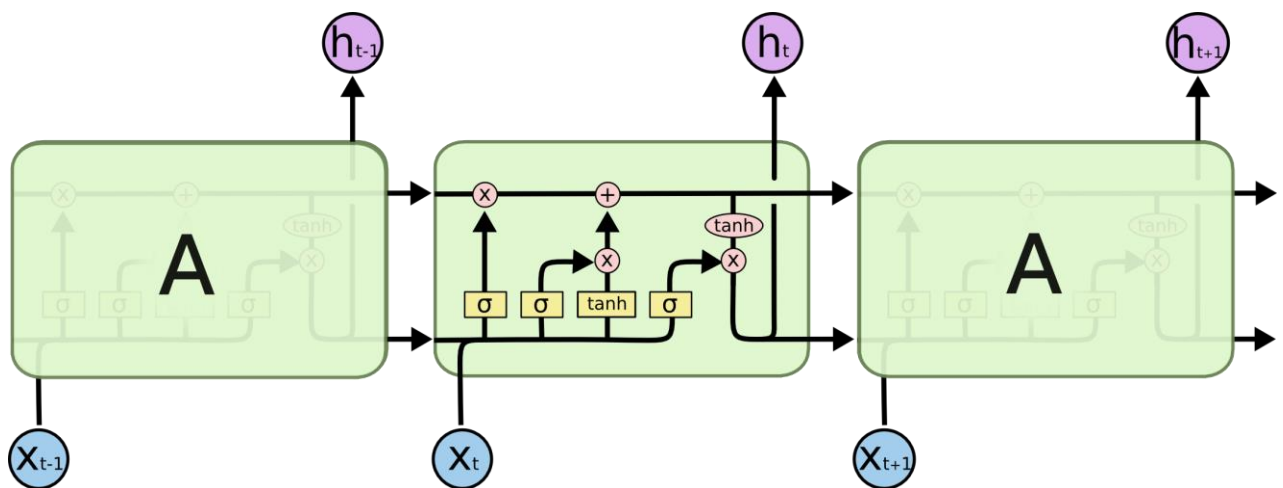
From the result shown above, the occurrences of 'like', 'know', 'singapore', 'one' and 'get' appeared very frequently in the corpus.

## Phase 4a: Training of Model (LSTM & GRU)

In this phase, we will be training various models with our corpus to see which model can generate the most coherent sentences. The model will then be saved and loaded again in phase 5 where we do the insertion of Singlish words according to the POS tagging of the generated sentences. This will be the most crucial phase as we aim to generate the sentences that are as coherent as possible. As such, the selection of models, the optimizers used, and the hyperparameter-tuning of the model played a part for this NLG task. Mainly, the models that we will use in this phase are the Recurrent Neural Network models (RNN models) such as Long-Short Term Memory (LSTM) and Gated Recurrent Unit (GRU).

### LSTM

LSTM are a special kind of RNN that is designed to learn long-term dependencies. Remembering information for long periods of time is its default behaviour. LSTM has a chain like structure, instead of having a single neural network layer, there are four, interacting in a very special way as shown in the diagram below:

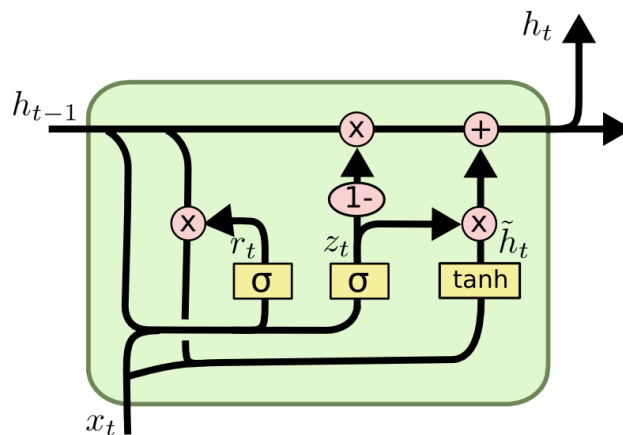


As our sentence generation will be long and also we may want to take into account the long input sentences, LSTM will be a favourable architecture for this NLG task.



## GRU

A simpler implementation of the LSTM will be the GRU. It combines the forget and the input gates into a single 'update gate'. It also merges the cell and the hidden state, hence having lesser parameters to train as compared to the LSTM. A sample of the GRU is as shown below:



## Models trained in the experiments

The models that we have considered in this phase are:

1. Stacked LSTM + Adam + 1 Epoch + Batch Size 128 (Testing Purposes)
2. Stacked LSTM + Adam + 20 Epoch + Batch Size 128 (Testing Purposes)
3. Stacked LSTM + SGD + Momentum + 20 Epoch + Batch Size 128 (Testing Purposes)
4. Stacked LSTM + Adam + 100 Epoch + Batch Size 128
5. Stacked LSTM + RMSprop + 100 Epoch + Batch Size 128
6. Stacked GRU + Adam + 100 Epoch + Batch Size 128
7. Stacked GRU + RMSprop + 100 Epoch + Batch Size 128
8. Stacked LSTM + Adam + 100 Epoch + Batch Size 64
9. Stacked LSTM + RMSprop + 100 Epoch + Batch Size 64
10. Stacked GRU + Adam + 100 Epoch + Batch Size 64
11. Stacked GRU + RMSprop + 100 Epoch + Batch Size 64

## Why stacked LSTM or GRU architecture?

Stacked LSTM/GRU hidden layers makes the model deeper and more accurate. Addition of layers adds levels of abstraction of input observations over time.

## Helper functions used in this phase

To ensure code readability and prevention of similar codes, several helper functions are written:

### ***load\_doc()***

- To load the corpus used to train the model (SgCorpus.txt).

### ***clean\_txt()***

- To do further cleaning of corpus by removing the punctuations, make all words lowercase and to make the corpus into word tokens.

### ***fit\_model()***

- To build and train the model by passing in the variations of the models.
- To save the weights and the training loss values as .h5 and .csv files respectively.

### ***load\_pretrain\_model()***

- To load the .h5 files that were previously trained and saved.

### ***generate\_text\_seq()***

- To generate the text given a starting word, a phrase or a sentence

### ***show\_generated\_text()***

- To print out the input and the generated text.

### ***load\_saved\_csv()***

- To load the CSV files so as to plot the training loss against the epochs.

## Implementation: Phase4a-next-sentence-generator.ipynb

### Preparing the data

We first declared the constants used in this phase as shown below:

```
[4] # FIXED CONSTANTS
WORD_LIMIT = 500000
LENGTH = 50 + 1 # input seed (50) + predicted output (1)
NO_OF_OUTPUT_WORDS = 30 # number of text predicted/generated

# file paths of the dataset
DATA = ["SgCorpus"]
COLAB_FILEPATH = './drive/My Drive/next-sentence-predictor/finalData/'
WEIGHTS_DIR = './drive/My Drive/next-sentence-predictor/saved_weights/'
```

Note that the constant 'WORD\_LIMIT' is the number of word tokens that will be taken into consideration in building the model. In total, there are approximately 4 million word tokens in our corpus and due to resource constraint, we can only get the first 500000 word tokens from our corpus.

Then, we will proceed to load and preprocess the data by calling the function 'clean\_txt' as shown below:

```
# DECLARE THE FINAL FILEPATH TO LOAD THE DATA
filename = COLAB_FILEPATH + DATA[0] + '.txt'
# CALL FUNCTION TO LOAD RAW DATA
data = load_doc(filename)
# PREVIEW SOME TEXT FROM THE RAW DATA
print(data[250:650])

ay. she's really sweet and cute, especially when she laughs does s
why do not they let the crashers play orientation games this year?
anyone knows that sassy indian girl from neytiri? she is so cute
does anyone know the guy from acs in atom?? he's 1.7m plus and is

[ ] # FUNCTION TO FURTHER CLEAN THE SCRAPED DATASET
def clean_txt(doc):
    tokens = doc.split()
    table = str.maketrans('', '', string.punctuation)
    tokens = [w.translate(table) for w in tokens]
    tokens = [word for word in tokens if word.isalpha()]
    tokens = [word.lower() for word in tokens]
    return tokens

[ ] # PASS THE RAW DATA INTO THE FUNCTION
tokens = clean_txt(data)
print(tokens[50:100])

['especially', 'when', 'she', 'laughs', 'does', 'anyone', 'know',
```

We then got the word sequences of the corpus. We skipped a word for each word sequence so that we can have more computational space to let our model process more word tokens. Each word sequence consists of 51 words, first 50 words as the input seed and the last word as the predicted output word. The code snippet of how we generate our word sequences is shown below.

```
# GET WORD SEQUENCES
lines = list()
for i in range(LENGTH, len(tokens), 2): # skip a word for each word sequences
    seq = tokens[i-LENGTH:i]
    line = ' '.join(seq)
    lines.append(line)
    # resource constraint (colab RAM),
    # take only the first (number of words = WORD_LIMIT) it encounters
    if i > WORD_LIMIT:
        break

print(len(lines))
```

249976

As such, there are approximately 250,000-word sequences of 51 words each. Next, we will prepare the sentence input and the word output by tokenizing the text sequence, converting it to a numpy array and then one-hot the output word as shown below:

```
# TOKENIZE TEXT SEQUENCE
tokenizer = Tokenizer()
tokenizer.fit_on_texts(lines)
sequences = tokenizer.texts_to_sequences(lines)

# convert to numpy array
sequences = np.array(sequences)

# assign X and y
X, y = sequences[:, :-1], sequences[:, -1]

# looking at the word embedding
#print(X[111])

# output
#print(y[111])

# SIZE OF THE VOCAB
vocab_size = len(tokenizer.word_index) + 1
#vocab_size

# one-hot the output y
y = to_categorical(y, num_classes=vocab_size, dtype='int8')

# GET THE SEQUENCE LENGTH
seq_length = X.shape[1]
#seq_length

# OUTPUT DIMENSION OF THE EMBEDDING LAYER
EM_OUTPUT_LENGTH = 50
```

We are now ready to build our model with the processed data.

## **Building and training the model**

We built the model with the 'fit\_model()' helper function. The function consists of model building, selection of optimizer, model compilation as well as training of the model. We have three variants in building the models, they are:

1. Stacked LSTM or Stacked GRU?
2. Adam or RMSprop?
3. Batch size 64 or 128?

In total, we will train a total of 8 models with 100 epochs. Initially, we wanted to include models that are trained with Stochastic Gradient Descent (SGD), but for many variations of the models, the training loss did not converge well even after some epochs for the training. In our training, the term "stacked" refers to two layers (e.g. Stacked LSTM is referring to two-layer LSTM). We also fixed some of the hyperparameters to the commonly used hyperparameters by researchers or other practitioners for simplicity of this project. These hyperparameters are:

1. Number of hidden layer neurons per layer: 128
2. Number of neurons in the last Dense layer: 64
3. Learning rate of all optimizers: 0.001

We have also placed 3 more test models (with lesser epochs trained) to compare the quality of the word generation with the actual models that were trained with 100 epochs.

### **Performance of each models**

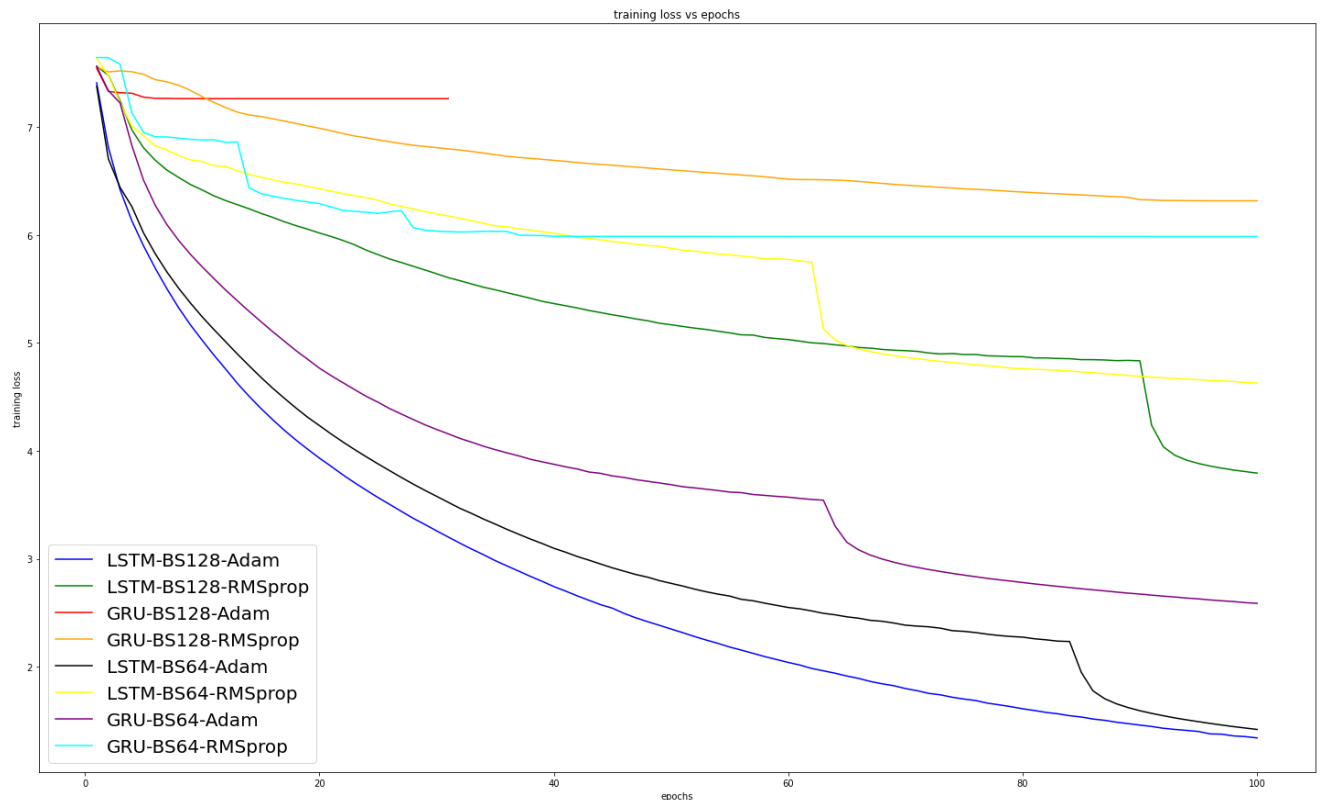
Here are the performances of the 11 models trained (3 test models and 8 actual models):

<b>Models</b>	<b>Number of Epochs</b>	<b>Training Accuracy</b>	<b>Training Loss</b>	<b>Testing/ Actual</b>
Stacked LSTM + Adam + Batch Size 128	1	0.0339	7.4446	Testing
Stacked LSTM + Adam + Batch Size 128	20	0.2390	4.1284	Testing
Stacked LSTM + SGD + Batch Size 128	20	0.0316	10.3689	Testing
Stacked LSTM + Adam + Batch Size 128	100	0.6804	1.3403	Actual
Stacked LSTM + RMSprop + Batch Size 128	100	0.4209	3.7939	Actual
Stacked GRU + Adam + Batch Size 128	31 (Early Stopping)	0.0316	7.2625	Actual
Stacked GRU + RMSprop + Batch Size 128	100	0.1279	6.3162	Actual
Stacked LSTM + Adam + Batch Size 64	100	0.6777	1.4187	Actual
Stacked LSTM + RMSprop + Batch Size 64	100	0.3573	4.6284	Actual
Stacked GRU + Adam + Batch Size 64	100	0.4979	2.5871	Actual
Stacked GRU + RMSprop + Batch Size 64	100	0.1700	5.9854	Actual

From the comparison table shown above, we can see that the stacked LSTM model + Adam using a batch size of 128 performs the best in terms of training accuracy and training loss, followed by the stacked LSTM model + Adam using a batch size of 64, followed by the stacked GRU model + Adam using a batch size of 64. From this table,

we can imply that Adam is perhaps the better optimizer for this NLG task and the stacked LSTM model generally performs better than the stacked GRU model.

Here is the plot of the training losses against the epochs for all the actual models to compare their performances:



### **Text generation for a given input sentence**

For the sake of completeness, we will be generating the next 30 words for a given fixed input sentence for all 11 models trained and see its coherency. To generate text, the helper function 'load\_pretrain\_model()' is used to load the trained model that was trained previously, then the helper function 'show\_generated\_text()' will load the model and the input sentence, and generates the next 30 words based on the input sentence.

## Results of the text generation

Here are the generated texts for all the 11 models given an input sentence:

**Input sentence: i came to see**

### Stacked LSTM + Adam + Batch Size 128 + 1 epoch

Result: [i came to see] to the the the the and and the the and and the and and and  
and and and and and and and and and and and and and and and and and Stacked LSTM + Adam  
+ Batch Size 128 + 20 epochs

Result: [i came to see] you guys to be able to get a best for the school in the end of the school was a levels and i am not a crush on my og

### Stacked LSTM + SGD + Batch Size 128 + 20 epochs

Result: [i came to see] the the the the the the the the the the the the the the the the  
the the the the the the the the the the the the

### Stacked LSTM + Adam + Batch Size 128 + 100 epochs

Result: [i came to see] you feel we have chris rich ah oh me selfies with a story never accept me please support my favourite dessert chilli french pork with cute colourful giveaway fastest cafe

## Stacked LSTM + RMSprop + Batch Size 128 + 100 epochs

Result: [i came to see] some activities safe and will go moving to catch the to the page who are truly a what they want to do no more answers beyond just do not forget

### Stacked GRU + Adam + Batch Size 128 + 100 epochs

Result: [i came to see] the the the the the the the the the the the the the the the  
the the the the the the the the the the the the

---

The above results show that the model can generate fluent sentences.

## Stacked GRU + RMSprop + Batch Size 128 + 100 epochs

Result: [i came to see] the new new and the best of the new singapore and the first of the new year is a first of the best and the first of the best and



#### Stacked LSTM + Adam + Batch Size 64 + 100 epochs

Result: [i came to see] yall portal i was given of people from people with making it it thus intelligent of me keep fighting celebration lets make people for aden confessions tutorial there can go

#### Stacked LSTM + RMSprop + Batch Size 64 + 100 epochs

Result: [i came to see] us for this tune ago in the singapore dinner forward from yummy epic and serves with mi thai giveaway is new details in the amount of front of it be

#### Stacked GRU + Adam + Batch Size 64 + 100 epochs

Result: [i came to see] everything staff its given forever kate school together until ur yeo date ceo time together moin staff go ceo setting more dress update koh exclusive artisanal cheng lane near holland

#### Stacked GRU + RMSprop + Batch Size 64 + 100 epochs

Result: [i came to see] you who was a lot of the school and i am not a time to be a great to me i am not a little time to be a great

From the above examples, we could see that models with the lowest training loss might not give the most coherent generated sentence. In fact, the test model Stacked LSTM + Adam + Batch Size 128 + 20 epochs actually generated a sentence that is quite coherent even though it is only trained on 20 epochs and there are quite a few repetitive words in it. And also models like Stacked GRU + Adam + Batch Size 128 + 100 epochs only generate repetitive words due to the model's training loss not being able to converge.

### **Choice of models for POS tagging and insertion of Singlish words in Phase 5**

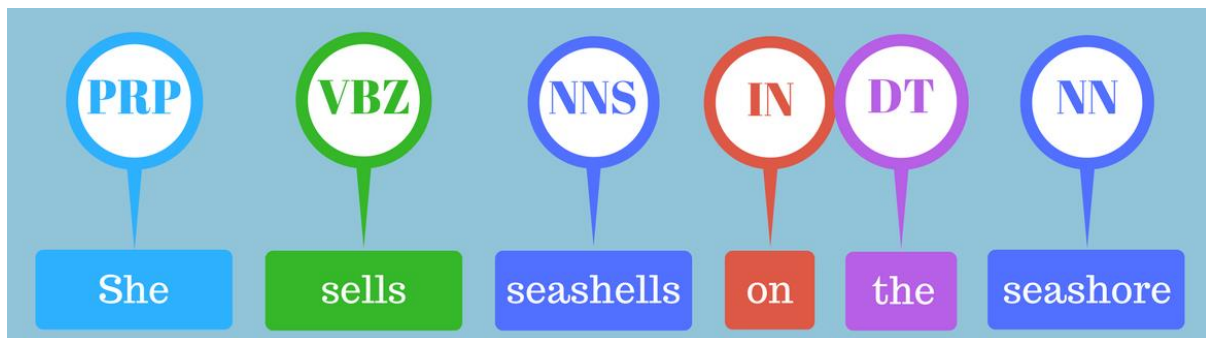
From the above experiment, we select 4 models to do further analysis and insertion of the Singlish words in Phase 5, the 4 models are:

1. Stacked LSTM + Adam + Batch Size 128 + 20 epochs
2. Stacked LSTM + Adam + Batch Size 128 + 100 epochs
3. Stacked LSTM + Adam + Batch Size 64 + 100 epochs
4. Stacked GRU + Adam + Batch Size 64 + 100 epochs

## Phase 5: POS tagging and insertion of Singlish words

### What is POS tagging?

POS tagging is the process of marking up a word in a corpus to a corresponding part of a speech tag, based on its context and definition. This task is not straightforward, as a particular word may have a different part of speech based on the context in which the word is used. To understand the meaning of any sentence or to extract relationships, POS tagging is a very crucial step. An example of POS tagging is as shown below:



In our context, we need to detect the POS tags of the generated words so that we can insert the Singlish words in the correct position to maintain the overall sentence coherency.

### Motivation of this phase

We initially wanted to find Singapore data that has lots of Singlish in it, so that the sentence generator can naturally generate sentences with Singlish without the need to do POS tagging. However, it is very tedious to get datasets which are rich in Singlish words. This is due to the fact that on average, most English sentences only consist of one or at most two Singlish words. Another factor is that the data we scraped are not in the perfect format, there might be spelling errors, incompatible character encoding etc which will affect our generation of text. Therefore, we change our problem formulation. Instead of focusing on the amount of Singlish words in a text, we focus on scraping the data that normal Singaporeans will say in text in a more informal way. We will then hand engineer the generated data by doing a POS tagging on it using the nltk library. Then we will list down a list of commonly used Singlish words and insert it into the generated sentence based on the POS tagging of the individual words.

## Helper functions used in this phase

To ensure code readability and prevention of similar codes, several helper functions are written:

### ***load\_doc()***

- To load the corpus used to train the model (SgCorpus.txt).

### ***clean\_txt()***

- To do further cleaning of corpus by removing the punctuations, make all words lowercase and to make the corpus into word tokens.

### ***load\_pretrain\_model()***

- To load the .h5 files that were previously trained and saved.

### ***generate\_text\_seq()***

- To generate the text given a starting word, a phrase or a sentence.

### ***show\_generated\_text()***

- To print out the input and the generated text.

### ***sentenceTagging()***

- To POS tag the generated sentence.

### ***singlify()***

- To insert the most common Singlish word into the generated sentence according to the POS tag.

### ***insert\_singlish\_word()***

- A composite function that does what ***sentenceTagging()*** and ***singlify()*** does.

## Implementation: Phase5-POS-insertion-Singlish-words.ipynb

### Preparing the data & loading of the model

The preparation of the data and loading of the model is exactly the same as that of Phase 4.

### Insertion of Singlish words into the generated sentence

After the generation of the sentence, we proceeded to do POS tagging on the sentence by calling the helper function named 'sentenceTagging()', which takes in the generated sentence and uses a POS-tagger from nltk to tag each word in the sentence. The function will return the sentence word and its tag. The implementation of the function is as shown below:

```
# Tagging the generated sentence
def sentenceTagging(sentence):
    #Initialize tagged list
    tagged_list = list()

    # Tokenize words
    wordsList = word_tokenize(sentence)
    # removing stop words from wordList
    wordsList = [w for w in wordsList]

    # Using a Tagger. Which is part-of-speech
    # tagger or POS-tagger.
    tagged = nltk.pos_tag(wordsList)
    for item in tagged:
        tagged_list.append(item)
    return tagged_list
```

We will then manually create a Singlish list which contains the commonly used Singlish words as shown below:

```
[ ] SINGLISH_ADJ_LIST = ["lah", "lorh", "leh", "sia", "mah", "meh", "kenna", "liao", "siao"]
```

This list can be easily extendable with more complex Singlish words.

Then, we will define a helper function named 'singlify()', which takes in the tagged sentence and the Singlish list. If it detects the zeroth word to be of some certain word, it will be replaced with the Singlish equivalent. Also, if the next word is an adjective, the Singlish list passed into the function will randomly select a Singlish word and insert it in between the zeroth and the first word, or else the first word will come after the zeroth word. The implementation of the function is as shown below:

```
def singlify(tagged_sentence, singlish_adj_list):
    singlish_sentence = ""
    for pair in tagged_sentence:
        word = pair[0]
        type_of_word = pair[1]
        # Replace words if they are the following (more Singlish way of representing in text)
        if word == 'zero':
            word = 'jilo'
        elif word == 'copy':
            word = 'kope'
        elif word == 'already':
            word = 'orredy'
        elif word == 'very':
            word = 'very the'
        elif word == 'vomit':
            word = 'womit'
        elif word == 'better':
            word = 'more better'

        # Looking for adjectives in the tagged sentence
        if type_of_word == 'JJ':
            singlish_word = random.choice(singlish_adj_list)

            # if following word is adjective, insert the Singlish word after the next word
            singlish_sentence = singlish_sentence + " " + word + " " + singlish_word
        else:
            # else just insert the next word
            singlish_sentence = singlish_sentence + " " + word

    # remove the first spacing
    singlish_sentence_ = singlish_sentence[1:]

    # Return sentence
    return singlish_sentence_
```

## Result of the finalised Singlish sentence

In this phase, we will be experimenting with 3 different text starters for the text generation task. We will generate 30 new words. From Phase 4, we have chosen 4 models for this phase. The 4 models are:

- Stacked LSTM + Adam + Batch Size 128 + 20 epochs
- Stacked LSTM + Adam + Batch Size 128 + 100 epochs
- Stacked LSTM + Adam + Batch Size 64 + 100 epochs
- Stacked GRU + Adam + Batch Size 64 + 100 epochs

The 3 different text starters are:

1. "here i am testing my nlg project.. "
2. "never did i have... "
3. "someday i will..."

### **Text starter 1: here i am testing my nlg project...**

#### Stacked LSTM + Adam + 20 Epochs + Batch Size 128

##### ***Text generated:***

here i am testing my nlg project and i am a crush on my og and i am not really nice i am not a crush on my og and i am not really nice i am

##### ***Singlish sentence:***

here i **lorh** am testing my nlg project and i am a crush on my og and i am not really nice **siao** i am not a crush on my og and i am not really nice **meh** i am

#### Stacked LSTM + Adam + 100 Epochs + Batch Size 128

##### ***Text generated:***

here i am testing my nlg project customers subscribe for the latest food news reviewsfacebook daniels food diaryinstagram food videos singapore deli chef recipes off on this new outlet by yong noodle can check out tunglok new

##### ***Singlish sentence:***

here i **lah** am testing my nlg project customers subscribe for the latest food news reviewsfacebook daniels food diaryinstagram food videos singapore deli **liao** chef recipes off on this new **siao** outlet by yong **meh** noodle can check out tunglok new **sia**

#### Stacked LSTM + Adam + 100 Epochs + Batch Size 64

##### ***Text generated:***

here i am testing my nlg project back craze spend loo speakers blogger was being that delivering job on drama masks for heavy session at the past courses and hasi is supporting me because the rest is

##### ***Singlish sentence:***

here i **kenna** am testing my nlg project back craze **kenna** spend loo speakers blogger was being that delivering job on drama masks for heavy **leh** session at the past **liao** courses and hasi is supporting me because the rest is

## Stacked GRU + Adam + 100 Epochs + Batch Size 64

### ***Text generated:***

here i am testing my nlg project ceo kate forever awesome ceo air on kandie flats ceo closing forever playing foreign workers wearing todays projects lets actually enjoying playing roasted resort date ceo lee roasted dumplings man

### ***Singlish sentence:***

here i **sia** am testing my nlg project ceo kate forever awesome **mah** ceo air on kandie flats ceo closing forever playing foreign **liao** workers wearing todays projects lets actually enjoying playing roasted resort date ceo lee roasted dumplings man

## **Text starter 2: never did i have...**

## Stacked LSTM + Adam + 20 Epochs + Batch Size 128

### ***Text generated:***

never did i have been a crush on my og is a lot of the best for jubilant you be a good to be a good time to be a good thing in the

### ***Singlish sentence:***

never did i have been a crush on my og is a lot of the best for jubilant you be a good **liao** to be a good **sia** time to be a good **leh** thing in the

## Stacked LSTM + Adam + 100 Epochs + Batch Size 128

### ***Text generated:***

never did i have deserve favours was the schools is chosen thesmartlocalcom away the girls out some ppl hes super terrible bottle of the cable system from stage of aug this was an hat

### ***Singlish sentence:***

never did i have deserve favours was the schools is chosen thesmartlocalcom away the girls out some ppl hes super terrible **liao** bottle of the cable system from stage of aug this was an hat

### Stacked LSTM + Adam + 100 Epochs + Batch Size 64

#### ***Text generated:***

never did i have crossed his results at tedx n was my initials community area there meet fighting pls want a wrong impression i forgot with stuff from music dollarsandsense family last list of

#### ***Singlish sentence:***

never did i have crossed his results at tedx **sia** n was my initials community area there meet fighting pls want a wrong **meh** impression i forgot with stuff from music dollarsandsense family last **mah** list of

### Stacked GRU + Adam + 100 Epochs + Batch Size 64

#### ***Text generated:***

never did i have hit forever on becoming awesome earlier ceo water a wealth ceo ceo date ceo rewards roasted diary new view liang coupons live using the high food highlights the worlds area

#### ***Singlish sentence:***

never did i have hit forever on becoming awesome **leh** earlier ceo **siao** water a wealth ceo ceo date ceo rewards roasted diary liao new **leh** view liang **mah** coupons live using the high **leh** food highlights the worlds area

### **Text starter 3: someday i will...**

### Stacked LSTM + Adam + 20 Epochs + Batch Size 128

#### ***Text generated:***

someday i will be shocked on the past years and i am been a good in the end of the best for yur in the end of the end of the end of

#### ***Singlish sentence:***

someday i will be shocked on the past **liao** years and i am been a good **lah** in the end of the best for yur in the end of the end of the end of



### Stacked LSTM + Adam + 100 Epochs + Batch Size 128

#### ***Text generated:***

someday i will be yesterday for frugal type channel with my family destination careerremember of bubble tea at citylink mall danielfooddiarycomwhere to say about the details to become said to get the boyfriend

#### ***Singlish sentence:***

someday i will be yesterday for frugal lorh type channel with my family destination careerremember of bubble **sia** tea at citylink mall danielfooddiarycomwhere to say about the details to become said to get the boyfriend

### Stacked LSTM + Adam + 100 Epochs + Batch Size 64

#### ***Text generated:***

someday i will get to pull down and honestly i am sure aj simply dance classics next other sp wins when it was such an amazing picture keep actually peace with of bento

#### ***Singlish sentence:***

someday i will get to pull down and honestly i **siao** am sure **sia** aj simply dance classics next other **lorh** sp wins when it was such **siao** an amazing **meh** picture keep actually peace with of bento

### Stacked GRU + Adam + 100 Epochs + Batch Size 64

#### ***Text generated:***

someday i will wanted to work on kim gold ceo bright koh lee wine sheng sea coffeehouse ceo daily bellywellyjelly discount at todays date in kandie chen steamboat arrives code koh pink roasted

#### ***Singlish sentence:***

someday i will wanted to work on kim gold **meh** ceo bright **lah** koh lee wine sheng sea coffeehouse ceo **lorh** daily **leh** bellywellyjelly discount at todays **siao** date in kandie chen steamboat arrives code **liao** koh pink roasted

## Challenges faced in this phase

Every Singlish word has its own unique meaning. However, it will be tough for us to match the meaning of the generated sentence with the correct Singlish word. As such, we adopted the random strategy of inserting the Singlish word based on the POS tag of the generated sentences randomly.

From the above examples, you can see that most sentences generated are incoherent as a whole. This can be due to the fact that the text we scraped may consist of spelling errors, short forms of some actual words or perhaps Singlish words in the corpus itself. All these factors will affect the coherency of the generated sentence.

Another challenge we faced could be the lack of unique word tokens. Due to resource limitations, the training of our model did not take in many unique words into account. Hence, there will only be a limited amount of words generated when we do sentence generation.

## Conclusion of the results

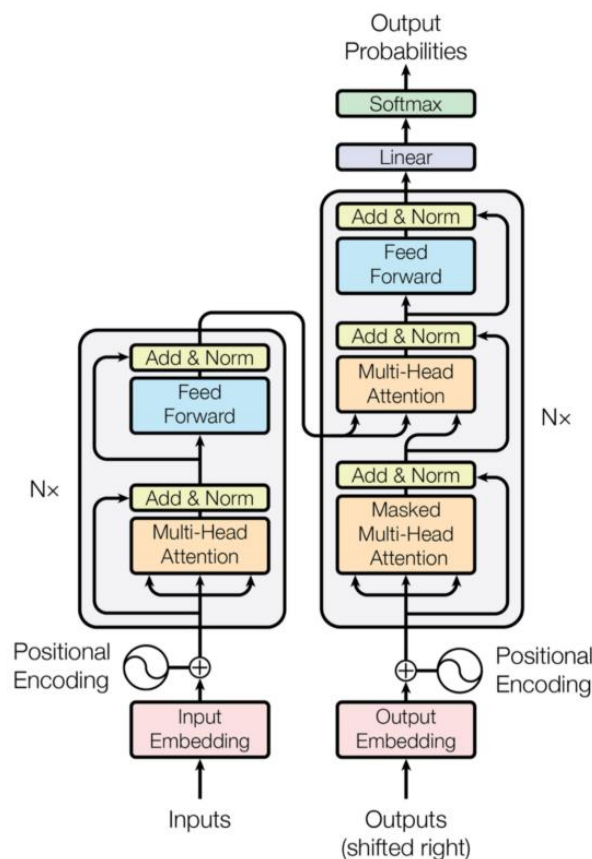
From the 12 examples shown above, it seems that with the 3 starter sentences, the Stacked LSTM + Adam + 20 Epochs + Batch Size 128 seemed to perform the best even though it is only trained with 20 epochs and has repetitive words.

Next, we will take a look at a bonus phase which may generate more coherent sentences which is to finetune a transformer model, GPT-2 on Google Colaboratory.

## BONUS PHASE - Phase 4b: Training of Model (GPT-2)

### What is a transformer?

Transformer is an architecture for transforming one sequence into another one with the help of two parts (Encoder and Decoder), but it differs from the previously described/existing sequence-to-sequence models because it does not imply any Recurrent Networks (GRU, LSTM, etc.). The architecture is as shown below:



In this phase, we will be using GPT-2, one of the transformers which is highly suitable for doing NLG tasks.

### What is GPT-2 (Generative Pretrained Transformer 2)?

GPT-2 is a large transformer-based language model with 1.5 billion parameters, trained on a dataset of 8 million web pages. GPT-2 is trained with a simple objective: predict the next word, given all of the previous words within some text. The diversity of the dataset causes this simple goal to contain naturally occurring demonstrations of many tasks across diverse domains. GPT-2 is a direct scale-up of its predecessor, the GPT, with more than ten times the parameters and trained on more than ten times the amount of data.

## Implementing GPT-2 into our project

In this phase, we will be loading GPT-2 from Google Colaboratory, then fine-tune the raw GPT-2 with our corpus used in Phase 4a, where we trained our model using RNN like LSTM and GRU. We will be loading the smallest GPT-2 model that is available in Google Colaboratory.

For GPT-2 to work in Google Colaboratory, we will have to backward compatible our tensorflow 2 to tensorflow 1 as seen in the code snippet below:

```
# backward compatibility
!tensorflow_version 1.x

TensorFlow 1.x selected.

[3] import tensorflow as tf

[4] tf.__version__

'1.15.2'
```

Then, we will have to declare the file path as a variable so that we can pass the value into the fine-tuning portion of the GPT-2 model. The declaration of the file path is as shown below:

```
[5] # file paths of the dataset
DATA = ["SgCorpus"]
COLAB_FILEPATH = './drive/My Drive/next-sentence-predictor/finalData/'

DECLARE FINAL FILEPATH

[6] # DECLARE THE FINAL FILEPATH TO LOAD THE DATA
filename = COLAB_FILEPATH + DATA[0] + '.txt'
```

We will then install GPT-2 into Google Colaboratory and we will download the smallest GPT-2 model available, which is the 124M model as shown below.

```
!pip install gpt-2-simple
import gpt_2_simple as gpt2
```

```
[ ] # download the gpt-2 small model
gpt2.download_gpt2(model_name="124M")

Fetching checkpoint: 1.05Mit [00:00, 307Mit/s]
Fetching encoder.json: 1.05Mit [00:00, 123Mit/s]
Fetching hparams.json: 1.05Mit [00:00, 420Mit/s]
Fetching model.ckpt.data-000000-of-000001: 498Mit [00:11, 44.7Mit/s]
Fetching model.ckpt.index: 1.05Mit [00:00, 359Mit/s]
Fetching model.ckpt.meta: 1.05Mit [00:00, 186Mit/s]
Fetching vocab.bpe: 1.05Mit [00:00, 187Mit/s]
```

Finally, we will fine-tune the GPT-2 model by loading our corpus into the training process so that it will take our corpus into account and generate text that is of the Singapore essence. We will first reset any previous sessions, then we will start a new tensorflow session and then fine-tune the model for 100 epochs as shown in the code snippet below.

```
# fine-tuning of gpt-2 model
tf.reset_default_graph()

sess = gpt2.start_tf_sess()

gpt2.finetune(sess,
              dataset=filename,
              model_name='124M',
              steps=100,
              restore_from='fresh',
              run_name='run1',
              print_every=25,
              sample_every=20,
              save_every=10)
```

We then save the checkpoint to Google Drive

```
# save checkpoint to google drive
gpt2.copy_checkpoint_to_gdrive(run_name='run1')
```

## Results from the fine-tuned GPT-2

To observe the generation of text by GPT-2, we will call the generate method by the gpt\_2\_simple library. Same as Phase 4a, we will be generating 30 word tokens and we will generate 3 samples of the generated words given a text starter.

### Text starter 1: here i am testing my nlg project...

```
# once the training is done, do the text generation
gpt2.generate(sess,
               length=30, # number of tokens generated
               temperature=0.7, # randomness in the values
               prefix="here i am testing my nlg project", # the start of the sentence
               nsamples=3, # number of sample sentences produced
               batch_size=1, # get the batch size
               top_k=50, # number of tokens taken into account for the text generation
               run_name='run1')
```

The 3 sentences generated are:

1. here i am testing my nlg project. this is the first time i have tested an object from nlg. the results are encouraging. but i can not believe what i am doing
2. here i am testing my nlg project. i am in a ctu, and i am really into studying nlg. but it is still hard work to get a nlg
3. here i am testing my nlg project from this week and i am so happy to be back in nlg. i am so happy to be back in nlg and thank you

### Text starter 2: never did i have...

```
# once the training is done, do the text generation
gpt2.generate(sess,
               length=30, # number of tokens generated
               temperature=0.7, # randomness in the values
               prefix="never did i have", # the start of the sentence
               nsamples=3, # number of sample sentences produced
               batch_size=1, # get the batch size
               top_k=50, # number of tokens taken into account for the text generation
               run_name='run1')
```

The 3 sentences generated are:

1. never did i have to talk to her. i was worried i would get mad at her and i am glad she is fine. i do not know who the girl is
2. never did i have time to think about it .i am sorry i thought of that but i just can not look at it. if i did, i would not have
3. never did i have to put in the effort to stay in the school.i really feel that i will never get to stay in a school that is so full of people

### Text starter 3: someday i will...

```
# once the training is done, do the text generation
gpt2.generate(sess,
               length=30, # number of tokens generated
               temperature=0.7, # randomness in the values
               prefix="someday i will", # the start of the sentence
               nsamples=3, # number of sample sentences produced
               batch_size=1, # get the batch size
               top_k=50, # number of tokens taken into account for the text generation
               run_name='run1')
```

The 3 sentences generated are:

1. **someday i will** be leaving you and the rest of the world with my own little girl. but i need some help with my feelings for you. you seem to be
2. **someday i will** be back and i have been thinking of you for some time. is there any way to stop seeing you? i am in a similar situation
3. **someday i will** be attending the first day of the new year celebrations at the local library to celebrate my new year. please feel free to come out to see me and

## Conclusion of the results

As we saw from the three examples, the GPT-2 model can generate quite coherent sentences as compared to the RNN model we have trained in Phase 4a. We will take a look at generating longer sentences from this model. A code snippet of it is as shown below.

### Text starter 4: i'm very interested in the field of artificial intelligence, it gives me ample opportunities to explore what i want to know...

```
# once the training is done, do the text generation
gpt2.generate(sess,
               length=500, # number of tokens generated
               temperature=0.7, # randomness in the values
               prefix=""i'm very interested in the field of
                           artificial intelligence, it gives me
                           ample opportunities to explore what
                           i want to know"", # the start of the sentence
               nsamples=3, # number of sample sentences produced
               batch_size=1, # get the batch size
               top_k=50, # number of tokens taken into account for the text generation
               run_name='run1')]
```

The 3 sentences generated are:

1. i'm very interested in the field of artificial intelligence, it gives me ample opportunities to explore what i want to know and what i am not. i am also much more interested in what i am doing and what i am not doing. i have also found that i am not as motivated as i used to be. i have been doing it for years, but the pace of it has slowly changed. i am now on my own and i am being forced to do things that i am not prepared to do. i am not sure if i am really ready yet, but i am feeling a little guilty about it.
2. i'm very interested in the field of artificial intelligence, it gives me ample opportunities to explore what i want to know. i have no idea where i am going to go this year, but i am hoping to find someone who knows me. i have been doing this for a few years now, and i have never been afraid to post this at least twice. it is so good to be able to see my relatives for the first time, and i do not feel bad because i am very happy to have them around. but no matter how much i try, i can not get enough of them. i have
3. i'm very interested in the field of artificial intelligence, it gives me ample opportunities to explore what i want to know. i have been looking forward to this field since i was a kid and i have been thinking about it for years. but since i am not a high school student, i am just not too sure. so i am not going to be a part of a science-based engineering school. i am still not sure if i will be a part of any sciences. i am not sure if i should have chosen some of the sciences myself. i am not sure if i should have chosen the physics or



## **Conclusion for this project**

In conclusion, we look at RNN models like LSTM and GRU and Attention models like GPT-2 to generate our sentences with Singapore context in it. In general, GPT-2 models perform better than RNN models in terms of coherency of the text generated. However, we are unable to extract the generated sentences from the GPT-2 model, hence unable to do a Singlish POS Tagging. There are also other factors to consider for the RNN model such as hyperparameter tuning and also the word embedding size. With all these factors kept on check, we will be able to train and generate better sentences.

## References

<https://www.yseop.com/what-is-nlg>

<https://www.kaggle.com/rtatman/the-national-university-of-singapore-sms-corpus>

<https://www.youtube.com/watch?v=e7dPem4hdQE>

[https://www.youtube.com/watch?v=DNLebQ\\_vYiw](https://www.youtube.com/watch?v=DNLebQ_vYiw)

<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

<https://machinelearningmastery.com/stacked-long-short-term-memory-networks/>

<https://medium.com/analytics-vidhya/pos-tagging-using-conditional-random-fields-92077e5eaa31>

<https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04>

<https://openai.com/blog/better-language-models/>

<https://arxiv.org/abs/1706.03762>