소프트웨어융합개론 최종 보고서

나경아, 문지원, 오유솔

0) 주제

자연어처리를 활용한 웹소설 생성

1) 프로젝트 목표 및 내용

1. 프로젝트 최종 목표

   본 프로젝트의 최종 목표는 웹소설을 스스로 작성하는 AI를 구현하는 것이다. AI 글쓰기는 자연어 처리와 딥러닝 분야의 주요 연구분야로, 언어모델과 데이터 학습을 통해 AI가 스스로 글을 작성할 수 있도록 하는 것이다. AI의 창의성과 문장구성능력에 대해서는 많은 이견이 있지만, AI가 글을 쓰는 것 자체는 매우 현실적인 이야기가 되었다. 실제로 로봇 저널리즘 분야에서는 Ai를 통해 기사를 작성하고 있으며, 로봇이 작성한 기사와 인간이 작성한 기사를 특별히 구분하기 어렵다고 하다. OPEN AI사가 개발한 GPT-3 모델의 경우 생성해내는 본문의 질이 매우 높아 인간이 작성한 본문과 구별해내기 어렵고, 이로 인해 여러 유해성과 위해성까지 언급되었다.

2. 주제에 대한 설명

   "웹소설 쓰는 AI" 프로젝트를 진행하며 다음과 같은 세가지 사항을 중점적으로 살펴보고 웹소설을 랜덤으로 작성하는 AI를 구현해 보고자 한다.

   a) 대량의 텍스트 데이터 수집 및 전처리
   b) 언어와 관련된 사전학습모델(Pre-training)을 사용하고 성능을 개선시키는 법
   c) 전이 학습(Transfer-learning)을 수행하는 법

랜덤으로 글을 쓰는 AI를 구현하기 위해서는 사전학습모델(Pre-training)을 활용하여 전이학습(Transfer-learning)을 수행할 수 있도록 해야 한다. 전이 학습이란, 먼저 대량의 데이터를 사전 훈련하여 사전학습모델을 만들고, 자신의 데이터를 통해 다시 학습하는 방식이다. 이러한 방식을 이용하면 데이터의 크기가 작아도 높은 정확도를 얻을 수 있다는 장점이 있다. 언어와 관련된 여러 사전학습모델 중 구글의 BERT, SKT의 koGPT2와 같은 사전학습모델을 비교, 분석하여 더 나은 성능을 가진 사전학습모델을 사용하고자 했다. 또한 대량의 텍스트 데이터를 추가 수집하여 사전학습모델의 성능을 개선시킬 수 있는 방법을 탐구하고자 했다. 이후 웹소설 데이터를 수집, 전처리하여 전이학습을 수행하고, 개연성을 가진 웹소설을 작성하는 AI를 구현하고자 했다.

그러나 데이터 수집 후, 사전학습 모델을 이용하려는 과정에서 문제가 발생하였다. OpenAI API 사이트에 접속하여 GPT-3 이용을 신청하였지만, GPT-3를 활용했을 때 가짜뉴스 생성 등 여러 사회적 문제가 발생 할 수 있기 때문에 보수적으로 운영하여 승인을 거의 해주지 않음을 알게 되었다. 따라서 다른 학습모델들을 찾아보게 되었고, LSTM 모델을 이용하여 본 프로젝트를 진행하기로 결정하였다.

2) 주제 선정 이유

인공지능 (AI)란 컴퓨터에서 인간과 같이 사고하고 생각하고 학습하고 판단하는 논리적인 방식을 사용하는 인간 지능을 본 딴 고급 컴퓨터 프로그램을 말한다. 요즘 인공지능은 정말 많은 분야에서 다양한 기능을 하고 있다. 고객 개인 맞춤형 상담, 환자의 임상 증상과 생체 신호를 통한 질병 예측, 골프 워치를 통한 실시간 지형 특성 알림, 베이비 모니터링, 기사 작성, 수필 작성 등 생활 속 여러 곳에서 자리 잡고 있다. 이런 인공 지능의 다양한 분야로의 발전과 적용 사례들을 보니, '창의적'인 부분에서 인간의 능력을 얼마나 따라올 수 있을지 궁금해졌다. 많은 사

람들의 취미 생활로 자리 잡힌 '모바일 웹 소설'과 같은 창작물을 구현해 낼 수 있을지 의문이 생겼다. 따라서, 우리는 '웹소설 생성 AI'를 구현해보기로 했다. 웹소설 중 가장 인기가 많은 장르인 '로맨스 판타지'에 집중하여 프로젝트를 진행하기로 했다.

3) 데이터 획득

학습시키고자 하는 데이터는 크롤링이나 다른 방법을 통해 접근할 수 있어야 했고, 특정 장르에 집중할 필요가 있었다. 따라서, 기존에 존재하는 웹소설 사이트를 selenium을 이용해 크롤링하여 데이터를 수집하고자 했다.

우선 목표로 정한 '로맨스판타지' 장르를 다루는 곳은 조아라, 네이버 웹소설, 리디북스, 카카오페이지 등 이었다. 그중에서도 아마추어 작가들이 글을 자유롭게 올리고, 글 생산량이 많은 '조아라' 사이트가 목표에 부합하다고 생각했다. 따라서, 투데이 베스트를 기준으로 2018년도, 2019년도, 2020년도의 소설을 크롤링하여 약 36mb의 데이터를 수집하였다. 소설의 특정 화의 주소에 접근하게 되면, 그 뒤로는 화면을 클릭하여 다음 화로 넘어 갈 수 있도록 코드를 구성하여 데이터를 수집했다.

크롤링 한 후의 데이터 저장 방식은 각각 csv와 txt였으며, 앞으로의 프로젝트 진행에 있어 전처리와 모델 적용에 더 접근이 편리한 형식의 데이터를 사용할 예정이었으며, 실제 데이터 전처리 과정에서는 txt 파일을 이용하여 진행하였다.

데이터의 전처리 과정은 주로 각 화에 종종 들어있는 '작가의 말'이나 소설의 주요 내용과는 관계없는 특수 문자나 이모티콘, 장면을 구분하는 문자들을 중점적으로 제거하였다.

In [ ]:

```
#해야할 것 - 텍스트 가져오고 싶은 작품 1화 주소 붙여넣기. 회수에 맞게 숫자 조정하기. 실행하기.
from urllib.request import urlopen
import requests
import time
from selenium import webdriver
import random

totalText = []
driver = webdriver.Chrome('C:/Users/크롬드라이버위치/chromedriver.exe')

driver.get("작품 1화 주소(모바일 버전)")

time.sleep(random.uniform(3,5))

#range()안에는 최종 화수-1.
for i in range(21):
    if i==0:
        while True:
            # 다음에 보기
            try:
                driver.find_element_by_xpath("""/html/body/div/div/div[1]/div/div[3]/a""").clic
            except:
                pass
            # x 버튼
            try:
                driver.find_element_by_xpath("""//*[@id="root"]/div/div[1]/div[3]/div[1]/a/img"
                break
            except:
                pass
        while True:
            try:
                text = driver.find_element_by_xpath("""//*[@id="root"]/div/div/div/div[1]/div[2
                totalText.append(text)
                break
            except:
                pass

        a = random.uniform(30,50)
        print(a)
        time.sleep(a)
        print("시간 지연 테스트")
        print(1,"화")

    while True:
        try:
            driver.find_element_by_xpath("""//*[@id="root"]/div/div/div/div[1]/a[2]/div""").cli
            #다음 화 버튼
            driver.find_element_by_xpath("""//*[@id="react-confirm-alert"]/div/div/div/div[3]/b
        except:
            pass

        try:
            driver.find_element_by_xpath("""/html/body/div/div/div[1]/div/div[3]/a""").click()
            break
        except:
            pass

    b = random.uniform(20,40)
    print(b)
```

```
    time.sleep(b)
    print("시간 지연 테스트")
    print(i+2,"화")
    while True:
        try:
            text = driver.find_element_by_xpath("""//*[@id="root"]/div/div/div[2]/div[1]/div[2]
            totalText.append(text)
            break
        except:
            pass

    #최종회수 -2
    #if i == 20:
        #break
    #[작품 소개]와 다음 화를 분리하기 위한 장치
    totalText.append('^^^')
print('텍스트 추출 성공')
driver.quit()

#제목에 ? 들어가면 안 되는 거 주의!
title = "[장르] 제목(-최종화수)"

#txt 버전(가독성용)
afterText = []
mainText = []

for i in range(len(totalText)):
    beforeText = totalText[i]
    afterText = beforeText.split("₩n")
    mainText.extend(afterText)
print(mainText)

f = open(title+".txt",'w', encoding='UTF8')
for i in range(len(mainText)):
    data = mainText[i]
    f.write(data)
    f.write("₩n")
    f.write("₩n")
f.close()

#csv 버전
afterText = []
mainText = []

for i in range(len(totalText)):
    beforeText = totalText[i]
    afterText = beforeText.split("₩n")
    mainText.extend(afterText)
print(mainText)

import pandas as pd
from pandas import DataFrame

df = DataFrame(mainText)
df

df.to_csv('csv용/'+title+'.csv',encoding='utf-8-sig')

#txt 버전(데이터용)
afterText = []
mainText = []
```

```
121
122 for i in range(len(totalText)):
123     beforeText = totalText[i]
124     afterText = beforeText.split("₩n")
125     mainText.extend(afterText)
126 print(mainText)
127
128 f = open("데이터용/"+title+".txt",'w', encoding='UTF8')
129 for i in range(len(mainText)):
130     data = mainText[i]
131     f.write(data)
132 f.close()
```

4) 구현 내용 설명

1. 텍스트 전처리

크롤링을 통해 수집한 텍스트 데이터에서 웹소설의 내용을 학습하고 작성하는데 방해가 될 것으로 예상되는 부분을 삭제하였다. 예를 들어 [작가후기]와 같은 부분은 웹소설의 내용과 관련이 없고 소설과 다른 느낌의 구어체를 가지고 있다. 더하여, 작가들마다 다른 형식을 취하고 있는 부분을 통일해주었따. 예를 들어 어떤 작가는 대화체를 " "(쌍따옴표)로 표시하는 반면, 어떤 작가는 [ ] (중괄호)로 표시하고 있다.

중점적으로 처리한 단락과 기호는 다음과 같다.

처리항목	처리방법
작가의 사담	[작품 후기]로 시작되는 단락 삭제
대화체	"", [], 「」, '' 로 시작되는 문장 검토, 삭제
2회 이상 느낌표와 물음표 사용	(???), ??, (?!!!!!!!!) 등의 표현 삭제
2회 이상의 줄바꿈 사용	두 번의 줄바꿈으로 통일, 단락 구분
4회 이상의 온점 사용	세 번의 온점으로 통일
2회 이상의 space 사용	한 번의 space로 통일
이모티콘 사용	:), :(등의 이모티콘 삭제
작품의 제목 및 소제목 표시	문장이 아닌 단어만으로 이루어진 제목 삭제
댓글 목록	삭제
단어 중간에 사용된 -와 —	대에-박, 쾅- 등의 표현을 대박, 쾅으로 대체

2. 맞춤법 수정

웹소설의 특징 상 인터넷 용어들, 맞춤법에 어긋난 표현들 등의 문장이 많았다. 작가들 마다 틀리는 맞춤법 유형이 달랐기 때문에 AI를 정확하게 학습시키기 위해서는 이를 정확하게 통일시키는 과정이 필요하다고 생각했. 따라서, 앞서 전처리 했던 텍스트들을 일괄적으로 맞춤법 검사를 진행해 데이터를 정제하기로 하였다.

맞춤법 검사에 사용한 것은 selenium을 통해 '네이버 맞춤법 검사기'에 접근하는 것이다. 한 번에 교정할 수 있는 글자가 500자로 한정이 되어 있었기 때문에, 텍스트들을 500자씩 나누어 준 후 검사를 시행했다. 500자씩 나누는 과정에서, 단어와 단어 중간에서 끊기면 맞춤법 검사에 차질이 생기기 때문에 이를 고려하여 코드를 작성하였다. ex) '게임을 시작했읍니다."가 '게임을 시작했'과 '읍니다.'로 나뉘는 경우, 정확한 수정이 불가능하다. 따라서 '게임을', '시작했읍니다.'로 나뉠 수 있도록 유의하였다.

In [ ]:

```
#1. total.txt -> 맞춤법 수정된 fixed_total.txt

from selenium import webdriver
from selenium.webdriver.common.keys import Keys
import time
from bs4 import BeautifulSoup

fp = open("total.txt",'r',encoding="utf-8") # total.txt 불러오기
text = fp.read()
fp.close()

#한번 교정할 수 있는 글자 : 500자
# 500자씩 나눠준 후 검색하기
# 중간에서 끊기지 않게 하기 위해 500자로 끊은 후, 가장 빨리 나온 띄어쓰기 기준으로 잘라내기

fix_list = []
while (len(text) > 500) :
    temp_str = text[:500]
    last_space = temp_str.rfind(' ')
    temp_str = text[0:last_space]
    fix_list.append(temp_str)

    text = text[last_space:]

fix_list.append(text)


dv = webdriver.Chrome() #chromedriver 디렉토리 확인하기
dv.get("http://www.naver.com")

elem = dv.find_element_by_name("query") #검색창
elem.send_keys("맞춤법 검사기")         # 검색하기
elem.send_keys(Keys.RETURN)

time.sleep(2)
textarea = dv.find_element_by_class_name("txt_gray")

new_str = ''
for fix in fix_list :
    textarea.send_keys(Keys.CONTROL, "a") #ctrl+a (리셋)
    textarea.send_keys(fix)

    elem = dv.find_element_by_class_name("btn_check")
    elem.click()

    time.sleep(1) #결과소스를 가져오는 시간이 교정되는 시간보다 빨라서

    soup = BeautifulSoup(dv.page_source, 'html.parser') #결과소스코드 가져오기
    st = soup.select("p._result_text.stand_txt")[0].text #교정된 텍스트만 가져오기
    new_str += st

fp = open("fixed.txt",'w',encoding='utf-8')
fp.write(new_str)
fp.close()
```

3. 텍스트 정제 및 하나의 의미 단락으로 구분

일정한 간격과 단락으로 텍스트를 정제하였다. 모델을 학습시키는 과정에서 문단 단위로 진행 할 예정이다.

In [ ]:

```
filename = "/content/drive/Shareddrives/소융개론 텀프로젝트/fixed.txt"
with io.open(filename, encoding = 'utf-8') as f :
    text_f = f.read()

from google.colab import drive
drive.mount('/content/drive')

sentence_list = []
for (idx, sen) in enumerate(text_f.split(".")):
    if (idx == int(len(text_f.split(".")) / 10)):
        break
    sentence_list.append(sen + ".")

for (idx, sen) in enumerate(sentence_list):
    sentence_list[idx] = sentence_list[idx].replace("₩n", "")

def check_sentence(sentence):
    if ((sentence.find("₩"") != -1) | (sentence.find(" “") != -1) | (sentence.find("” ") != -
        return (0)
    elif ((sentence.find("[") != -1) | (sentence.find("]") != -1)):
        return (0)
    elif ((sentence.find("(") != -1) | (sentence.find(")") != -1)):
        return (0)
    elif ((sentence.find("<") != -1) | (sentence.find(">") != -1)):
        return (0)
    elif ((sentence.find("「") != -1) | (sentence.find("」") != -1)):
        return (0)
    elif ((sentence.find("-") != -1) | (sentence.find("_") != -1)):
        return (0)
    elif ((sentence.find("*") != -1) | (sentence.find("~") != -1)):
        return (0)
    elif ((sentence.find("^") != -1) | (sentence.find("#") != -1)):
        return (0)
    elif (sentence.find("  ") != -1):
        return (0)
    elif ((sentence.find("??") != -1) | (sentence.find("!!") != -1)):
        return (0)
    elif (sentence.find(".") == -1):
        return (0)

    else:
        return (1)
temp = ""
num = 0
maxi = 10
for (idx, sentence) in enumerate(sentence_list):
    if (check_sentence(sentence) == 0):
        continue
    else:
        temp += sentence
        num += 1
        if (num == maxi):
            temp += "₩n₩n"
            num = 0
```

4. 토큰화

영어의 경우, 띄어쓰기로 단어가 구분이 되기 때문에 일반적인 토큰화 라이브러리를 사용하여도 괜찮다. 그러나 한글의 경우, 조사, 어미, 접사 등이 있어 띄어쓰기로 정확한 의미 단위로 단어가 구분이 되지 않는다. 따라서 한국어 처리를 위해서는 단순 공백으로 단위를 구분하는 것이 아닌, 형태소 분석을 통해 단어들을 토큰화 해야 했다. 이를 위해 KoNLPY의 한국어 처리 패키지를 이용하여 형태소 분석을 진행하였다.

In [ ]:

```
pip install konlpy

# 형태소 토큰화

from konlpy.tag import Okt
okt = Okt()
result = okt.morphs(text)
print("단어의 개수: " , len(result))
print(result)
```

5. LSTM 모델 구성 및 학습 진행

LSTM과 RNN 모델

전통적인 neural network는 이전의 사건을 고려하여 이후의 사건을 구성하는 일을 수행하지 못한다는 단점이 있다. 인간의 경우 앞 문장과 뒷 문장, 앞 단어와 뒷 단어의 문맥적 관계를 파악하고 이를 순서대로 나열할 수 있다. 그러나 전통적인 neural network는 일련의 순서가 있는 사건을 분류하지 못한다. 해당 문제를 개선하기 위하여 등장한 모델이 RNN(Recurrent neural network)이다. RNN은 입력과 출력의 과정을 반복하며 이전의 입력과 출력을 통해 얻은 정보가 지속적으로 기억될 수 있도록 한다. 즉, 일련의 시퀀스를 가진 모델이다. 따라서 우리는 RNN 모델에 이전에 나올 단어들과 이후에 나올 단어를 학습시키고 그 과정을 기억하도록 하여 글을 작성하는 AI를 만들 수 있다. 그러나 RNN의 단점은 현재의 위치에서 비교적 가까이에 있는 정보만 기억할 수 있다는 것이다. 더 긴 문장이나 단어를 기억해야 할 경우에는 좋은 결과를 내지 못한다. 따라서 긴 기간의 의존성을 효율적으로 구현하기 위해 LSTM(Long Short-Term Memory) 모델을 사용하였다.

LSTM 모델의 파라미터 : echos, iteration, batch_size의 의미

echos는 전체 데이터를 몇 번 반복하여 학습시킬지 결정하는 변수이다. overfitting과 underfitting의 상황을 막기 위해서는 echos의 횟수를 적절하게 설정하는 것이 중요하다. 보통 1echo를 실행할 때 메모리의 한계와 속도 문제로 인해 데이터를 나누어 학습시키게 된다. 이 때 한번에 학습시키는 데이터의 크리를 batch_size라고 한다. 예를 들어 총 100개의 데이터를 50 echos로 학습시킨다고 하자. 이때 100개의 데이터는 한 번에 학습시키기 어려우무로 10개(batch_size)에 나누어 학습시킬 수 있다. 이러한 과정을 50번 하는 것이 50 echos의 의미이다.

- Reference: https://dgkim5360.tistory.com/entry/understanding-long-short-term-memory-lstm-kr (https://dgkim5360.tistory.com/entry/understanding-long-short-term-memory-lstm-kr) https://wegonnamakeit.tistory.com/7 (https://wegonnamakeit.tistory.com/7) https://omicro03.medium.com/%EC%9E%90%EC%97%B0%EC%96%B4%EC%B2%98%EB%A6%AC-nlp-12%EC%9D%BC%EC%B0%A8-ltsm-81c9751afafb (https://omicro03.medium.com/%EC%9E%90%EC%97%B0%EC%96%B4%EC%B2%98%EB%A6%AC-nlp-12%EC%9D%BC%EC%B0%A8-ltsm-81c9751afafb)

LSTM 모델 사용 및 진행 : keras를 이용하여 LSTM 모델을 구성하였고, batchsize는 128, epoch을 40번으로 설정하여 학습을 진행하였다.

keras의 lstm 모델을 사용하기 위해서 단어 벡터화를 하던 중에 colab의 세션이 계속 종료되는 현상이 발생하였다. 그래서 원인을 생각해보았고, 데이터의 용량이 너무 커서 그런 것일 수 있다는 결론에 도달했다. 따라서 데이터를 나눠서 시도를 해보려했다.

79717개의 문장을 나누어서 문장 10000개가 들어가는 텍스트파일을 각각 만들었으며, 해당 코드는 아래에 있다.

In [ ]:

```
f= open('/content/AI_novel_generator/total.txt', 'r')
lines = f.readlines()
f.close()

cnt = 1
fileName_Num = 1

for line in lines:
    fileName = "total_list_"+str(fileName_Num)+".txt"

    fw = open(fileName, 'a')
    fw.write(line)
    fw.close()

    if cnt == 10000:
        fileName_Num = fileName_Num +1
        cnt = 0

    cnt = cnt +1
```

그러나 여전히 진행과정 중에서 세션이 종료되는 경우는 크게 1) 단어를 numpy 배열로 바꿀 때와, 2) 그걸 이용해서 모델을 학습할 때가 남아있었고, 문제가 발생하는 두 과정을 나누어서 진행해야겠다는 생각을 했다.

그래서 1)에서 .npy 형식으로 데이터를 저장하고, 해당 파일을 이용해 2)를 실행시켜 보았다. 각 과정은 np.save와 np.load를 이용하였다.

In [ ]:

```
x = np.zeros((len(sentences), maxlen, len(chars)), dtype=np.bool)
y = np.zeros((len(sentences), len(chars)), dtype=np.bool)
for i, sentence in enumerate(sentences):
    for t, char in enumerate(sentence):
        x[i, t, char_indices[char]] = 1
    y[i, char_indices[next_chars[i]]] = 1
np.save('/content/list_x1', x)
np.save('/content/list_y1', y)
```

In [ ]:

```
x = np.load('/content/list_x1.npy')
y = np.load('/content/list_y1.npy')
```

데이터를 나누어 학습시키고, 학습과정을 나눠서 진행했던 코드 전문은 다음과 같다.

In [ ]:

```
f= open('/content/AI_novel_generator/total.txt', 'r')
lines = f.readlines()
f.close()

cnt = 1
fileName_Num = 1

for line in lines:
    fileName = "total_list_"+str(fileName_Num)+".txt"

    fw = open(fileName, 'a')
    fw.write(line)
    fw.close()

    if cnt == 10000:
        fileName_Num = fileName_Num +1
        cnt = 0

    cnt = cnt +1

x = np.zeros((len(sentences), maxlen, len(chars)), dtype=np.bool)
y = np.zeros((len(sentences), len(chars)), dtype=np.bool)
for i, sentence in enumerate(sentences):
    for t, char in enumerate(sentence):
        x[i, t, char_indices[char]] = 1
    y[i, char_indices[next_chars[i]]] = 1
np.save('/content/list_x1', x)
np.save('/content/list_y1', y)
```

In [ ]:

```
from tensorflow import keras
from tensorflow.keras import layers

import numpy as np
import random
import io
```

In [ ]:

```
path = '/content/total_list_1.txt'
with io.open(path, encoding="utf-8") as f:
    text = f.read().lower()
text = text.replace("₩n", " ")
print("Corpus length:", len(text))

chars = sorted(list(set(text)))
print("Total chars:", len(chars))
char_indices = dict((c, i) for i, c in enumerate(chars))
indices_char = dict((i, c) for i, c in enumerate(chars))

maxlen = 40
step = 3
sentences = []
next_chars = []
for i in range(0, len(text) - maxlen, step):
    sentences.append(text[i : i + maxlen])
    next_chars.append(text[i + maxlen])
print("Number of sequences:", len(sentences))
```

In [ ]:

```
def sample(preds, temperature=1.0):
    preds = np.asarray(preds).astype("float64")
    preds = np.log(preds) / temperature
    exp_preds = np.exp(preds)
    preds = exp_preds / np.sum(exp_preds)
    probas = np.random.multinomial(1, preds, 1)
    return np.argmax(probas)
```

In [ ]:

```
model = keras.Sequential(
    [
        keras.Input(shape=(maxlen, len(chars))),
        layers.LSTM(128),
        layers.Dense(len(chars), activation="softmax"),
    ]
)
optimizer = keras.optimizers.RMSprop(learning_rate=0.01)
model.compile(loss="categorical_crossentropy", optimizer=optimizer)
```

In [ ]:

```
x = np.load('/content/list_x1.npy')
y = np.load('/content/list_y1.npy')
```

In [ ]:

```
epochs = 40
batch_size = 128

for epoch in range(epochs):
    model.fit(x, y, batch_size=batch_size, epochs=1)
    print()
    print("Generating text after epoch: %d" % epoch)

    start_index = random.randint(0, len(text) - maxlen - 1)
    for diversity in [0.2, 0.5, 1.0, 1.2]:
        print("...Diversity:", diversity)

        generated = ""
        sentence = text[start_index : start_index + maxlen]
        print('...Generating with seed: "' + sentence + '"')

        for i in range(400):
            x_pred = np.zeros((1, maxlen, len(chars)))
            for t, char in enumerate(sentence):
                x_pred[0, t, char_indices[char]] = 1.0
            preds = model.predict(x_pred, verbose=0)[0]
            next_index = sample(preds, diversity)
            next_char = indices_char[next_index]
            sentence = sentence[1:] + next_char
            generated += next_char

        print("...Generated: ", generated)
        print()
```

5) 구현 결과

- AI가 작성한 문단

> Generating text after epoch: 39
> ...Diversity: 0.2
> ...Generating with seed: ", 원단은 왜 사가는 거야?" 레오니드 친구들 중 공부를 잘할 것처럼"
> ...Generated: 일어나는 그에게 한 대 저런 말을 하는 사람은 내가 이런 말이야." "아, 아니 아니. 그렇다고 그 말이 한 번 대회 제 목숨이 있는 거야?" "그런데 나한테 아니라 말하자. 그런 거지? "아, 그런 거고요!" "그래, 이거 나 자, 이 자식이나 비약이야. 그런 거라면, 그런 게 아니겠는가. 그런 이들이 아니겠는가. 그런 게 뭐라고?" "아, 아니 아니. 그런 게 아니겠는 것이 아닌가. 그런데 그 말에 이런 말들을 하는 인물이라면, 나는 그런 이들이 내게 들어오지 않을 그런 기숙사의 비약이야은 없었지만, 나는 그런 도로 내게 된다 하는 그 사람은 기사의 맹세를 맺어줄 수 있는 그 녀석이 그런 이 마법사 내가 이어를 붙잡은 채 대회 소파에 돌던 눈래를 이었다. 그리고는 그러면 그런 아처

6) 결론

데이터를 수정하고 정제하는 여러 단계를 거쳤지만, 프로그램은 기대했던 만큼의 자연스러운 소설을 작성하지는 못했다. 가장 큰 이유는 AI가 자연스러운 문장과 내용을 구현하기에는 데이터가 충분하지 않아서라고 생각한다. LSTM 모델은 주어진 데이터에 대해서만 학습을 하기 때문에, 모델이 주어진 데이터에 과도하게 맞춰져, 새로운 글을 작성하는 데에는 부족한 부분이 있다. 더 많은 데이터를 적용했다면 결과가 더 정확했을지, 웹소설이 아닌

시중에 판매되는 이미 모든 정제가 완료된 소설들을 이용했다면 결과가 달랐을지에 대한 궁금증도 생겼다. AI가 다소 부자연스러운 문단의 소설을 작성하였지만, '주어-목적어-부사어-서술어' 등의 국어식 나열법에 만족하는 부분들을 볼 수 있었다. 다양한 전처리 방법을 공부해보고, 여러 언어 모델에 대해 조사하며 비교하고 적용해보는 과정을 통해 새롭게 알아가는 것들이 많았다. 비슷한 관심분야를 가진 조원들과 함께 진행해 가며, 프로젝트를 잘 마무리할 수 있었다.

- 모든 과정에서 사용된 전체 코드와 데이터, 결과물 등은 다음 링크에서 확인할 수 있다.
  https://drive.google.com/drive/u/1/folders/1k-F4kR5skBmKFTH2r3VvnHJLU02vzpqm
  (https://drive.google.com/drive/u/1/folders/1k-F4kR5skBmKFTH2r3VvnHJLU02vzpqm)

조원 별 역할

- 나경아 :
  주제 변경 과정에서 아이디어 제시, RNN모델과 LSTM 모델 제시, 학습 모델 코드 작성 및 진행 방향성 제시, txt파일의 데이터를 문단화
- 문지원 :
  주제 선정 과정에서 아이디어 제시, 주제 변경 과정에서 (진행하기로 결정된) 새로운 아이디어 제시, 계획서 전문 작성, 데이터 수집 과정의 전 크롤링 코드 작성, LSTM모델을 이용한 학습 과정에서, 최대 얼마정도 학습을 시킬 수 있는지 알아내어 데이터를 활용할 수 있는 방법 찾고자 함.
- 오유솔 :
  주제 변경 과정에서 아이디어 제시, 맞춤법 검사 코드 작성 및 텍스트 수정, 형태소 토큰화 후 인덱싱
- 다같이 진행한 부분 :
  여러 모델들에 대한 공부(장단점 파악, 프로젝트에 적합한 모델 선정), 데이터 크롤링하기, 텍스트 전처리, 중간보고서 작성, 최종 보고서 작성 및 영상 제작