Word Prediction

Word Prediction System
Based on Deep learning

Team	1	**Major**	소프트웨어학과
Team member	최지원	**ID**	201420997
	김상호		201620977
	최혜지		201620933

Contents

1. 프로젝트 소개

1.1. 프로젝트 시작 동기 및 목적

자연어(natural language)란 우리가 일상 생활 속에서 사용하는 언어를 말한다. 또한 자연어 처리(natural language processing)란 이러한 자연어를 의미를 분석하여 컴퓨터가 처리할 수 있도록 하는 일을 말한다. 자연어 처리는 음성 인식, 내용 요약, 번역, 사용자의 감정 분석, 텍스트 분류 작업(스팸 메일 분류, 뉴스 기사 카테고리 분류), 질의 응답 시스템, 챗봇과 같은 곳에서 사용되는 연구 분야이다. 최근 딥 러닝이 주목을 받으면서, 인공지능이 제 4차 산업혁명의 주요 키워드로 떠오르고 있다. 자연어 처리는 기계에게 인간의 언어를 이해시킨다는 점에서 인공지능에 있어서 가장 중요한 연구 분야이면서도, 아직도 정복되어야할 산이 많은 분야이다.

그 중에서도 자동완성 기능은 언어모델과 관련이 있는데, 자연어 생성의 기반이 되는 언어모델은 문장의 확률을 계산하거나 이전 단어들이 주어졌을 경우 다음 단어가 나올 확률을 계산하는 것이다. 즉, 언어 모델은 단어들의 조합에 따라 이들이 얼마나 적절한 지, 그리고 해당 문장이 문단에 얼마나 적합한 지 따위의 일을 한다. 이번 프로젝트에서는 딥러닝 기반의 자동완성기능을 텐서플로우를 이용한 프로그램(https://github.com/Kyubyong/word_prediction 참고)을 통해 정리하고, 기존 성능을 개선하며 유지/보수할 수 있는 방안에 대해 모색해 보는 시간을 가졌다.

1.2. 팀 업무 분담 및 역할

본 프로젝트에서는 오픈소스를 기반으로 진행했다. 각 파트를 조원들과 함께 분담하여 프로젝트를 진행하였고, 역할은 다음 표와 같다.

	최혜지	김상호	최지원
Installation	O	O	O
Code Analysis	O	O	O
Code Modifying			O
Debugging		O	
Data Research	O		

2. Word Prediction(Open Source)

2.1. 프로그램 소개

해당 프로그램의 정식 명칭은 word_prediction 으로 GitHub에 올라와 있는 오픈소스이다. 이 프로그램은 Convolutional Neural Network를 사용하여 iPhone의 키보드 프로그램보다 단어 예측을 더 잘 할 수 있는지 시험하기 위해 만들어졌다. 언어 모델링 작업에서 보다 널리 사용되는 RNN은 short term memory에 약한 성능을 보이기에 CNN의 깊은 레이어가 이를 해결할 수 있다고 여겨 CNN을 사용하였다. 또한 Char-to-char 모델은 자기 회귀적 가정에 의존한다는 한계가 있어 Character-to-word 모델을 사용한다. 구체적으로 말하자면, 앞의 50 글자를 보고 현재 또는 다음 단어를 예측한다. 단어 예측을 위한 데이터로는 Wikinews dumps의 English news를 사용하였다.

2.2. Requirements.txt

numpy >= 1.11.1
sugartensor >= 0.0.2.4
lxml >= 3.6.4.
nltk >= 3.2.1.
regex

2.3. Glossary / Metric

$$KSR = \frac{FK-RK}{FK}$$

<그림 2.3.a KSR의 계산 공식>

2.3.1. Full Keystrokes(FK)

사용자가 Word Prediction 기능을 비활성화 했다고 가정하였을 때, 사용자가 입력한 문자(공백 포함)의 개수이다.

2.3.2. Responsive Keystroke(RK)

사용자가 의도한 단어가 제시될 때 사용자가 항상 선택할거라고 가정하는 경우, 그 단어가 제시되기 전 까지의 사용자가 입력한 문자의 개수이다.
RK를 줄이는 것이 곧 기능을 높이는 것이다.

2.3.3. Keystroke Saving Rate(KSR)

예측 모델에 의한 입력 문자의 절약 비율이다. 계산식은 위 그림(2.3.a)과 같다.

2.4. Results

약 2~3일간의 학습훈련을 바탕으로 한 pretrained 파일에 따르면 본 프로그램의 성능 아이폰7의 자동완성 프로그램 기능과 비교해 보았을 때 다음과 같다.

#FK	#RK: Ours	#RK: iPhone 7
40,787	~~24,727 (=0.39 ksr)~~ ->23,753 (=0.42 ksr)	21,535 (=0.47 ksr)

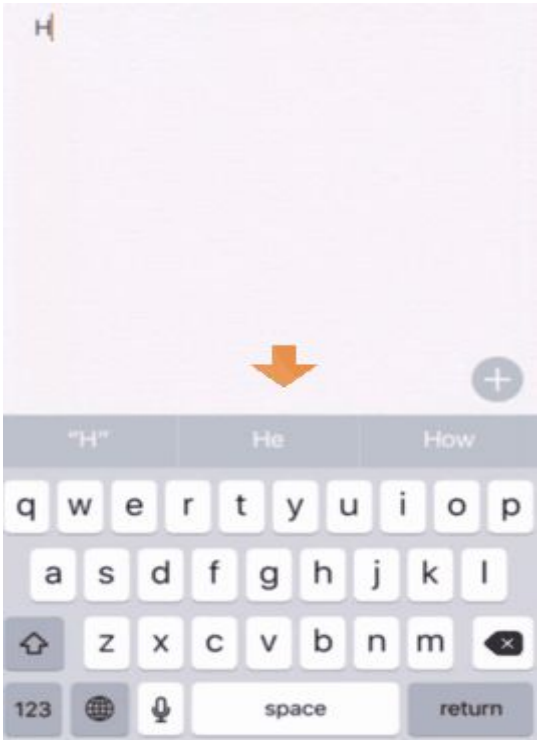
위의 데이터에 의하면 해당 프로그램의 성능은 기존 아이폰 7 자동완성 기능보다 약 5%가량 낮다.

2.5. Conclusions

결과적으로 프로그램 성능이 기존 자동완성 기능보다 낮은 것으로 나타났다. 따라서 우리는 이 점을 토대로 프로그램 성능을 높여보는 것을 최종 목표로 프로젝트 수행계획을 세웠다.

3. 프로젝트의 목표

3.1. 목표의 기준



<그림 3.1.a 자동 완성의 예>

목표를 달성하기 위해서는 여러가지의 기준을 가지고 목표를 달성했는지를 판단한다. 단어 자동 완성에서는 KSR(Keystroke Saving Rating)을 평가 기준으로 사용한다. KSR은 사용자가 단어 자동 완성 기능을 사용한다고 가정하고 단어를 입력할 때, 단어 전체를 입력 할 때보다 얼마나 적게 타자를 치는가를 나타내는 지표로 계산 공식은 <그림 2.3.a>과 같다. FK는 줄일 수 없는 변수로 이 프로젝트에서 실질적으로 줄여야 할 것은 RK로 Responsive Keystroke다.

3.2. 최종 목표

우리 프로젝트의 최종 목표는 현재 가장 사람들이 많이 사용하는 단어 자동 완성인 아이폰의 자동 완성 기능을 뛰어 넘는 것이다. 즉 다음에 올 단어를 좀 더 높은 확률로 예측하는 것이다. 그 최종 목표를 달성하기 위해서는 그 중간 중간의 단기적인 목표를 먼저 달성하는 것을 방법으로 선택했다. 그 단기 목표는 open source인 word prediction의 성능을 기존보다 향상 시키는 것으로 정했다. word prediction의 성능은 KSR = 0.4176으로 이 KSR를 점점더 향상 시키는 것을 단기 목표로 하여 최종적으로는 아이폰의 단어 자동 완성의 KSR인 0.47320을 넘기는 것으로 목표로 한다.

4. 프로젝트 세부사항

4.1. 구현 코드 설명

4.1.1. build_corpus.py

build_corpus.py

```
1   #/usr/bin/python2
2   # coding: utf-8
3   from __future__ import print_function
4   import numpy as np
5   import pickle
6   import codecs
7   import lxml.etree as ET
8   import regex
9   from nltk.tokenize import sent_tokenize
10
11  def clean_text(text):
31
32  def build_corpus():
57
58  if __name__ == '__main__':
```

<그림 4.1.1.a inner code of ‘build_corpus.py’>

자동완성 기능의 패턴을 분석하기 위한 데이터인 enwikinews-20181201-pages-articles-multistream.xml 파일을 en_wikinews.text 파일로 변환하는 부분이다. 프로젝트 실행의 가장 첫 번째 단계에서 진행된다.

4.1.2. prepro.py

prepro.py

```
1   #/usr/bin/python2
2   # coding: utf-8
3   '''
4   Preprocessing.
5   Make training/test data and a set of vocabulary.
6   '''
7   from __future__ import print_function
8   import numpy as np
9   import pickle
10  import codecs
11
12  class Hyperparams:  # We will predict the next/current word based on the
17
18  def load_char_vocab():
24
25  def create_word_vocab():
36  def load_word_vocab():
39
40  def create_data():
63
64  def load_train_data():
68
69  def load_test_data():
73
74  if __name__ == '__main__':
78
```

<그림 4.1.2.a inner code of 'preopro.py'>

build_corpus.py의 결과로 생성된 enwikinews.txt 파일을 가지고 training/test와 영어 단어의 집합을 만든다.

4.1.3. train.py

```
train.py
1   # -*- coding: utf-8 -*-
2   '''
3   Training.
4   '''
5   from __future__ import print_function
6   from prepro import *
7   import sugartensor as tf
8   import random
9
10  def q_process(t1, t2):
30
31  def get_batch_data(): # (64, 50) int32, (64, 50) int32, ()
54
55  class ModelGraph():
98
99  def train():
105
106 if __name__ == '__main__':
```

<그림 4.1.3.a inner code of 'train.py'>

prepro.py 에서 생성된 파일을 가지고 학습을 시킨다.

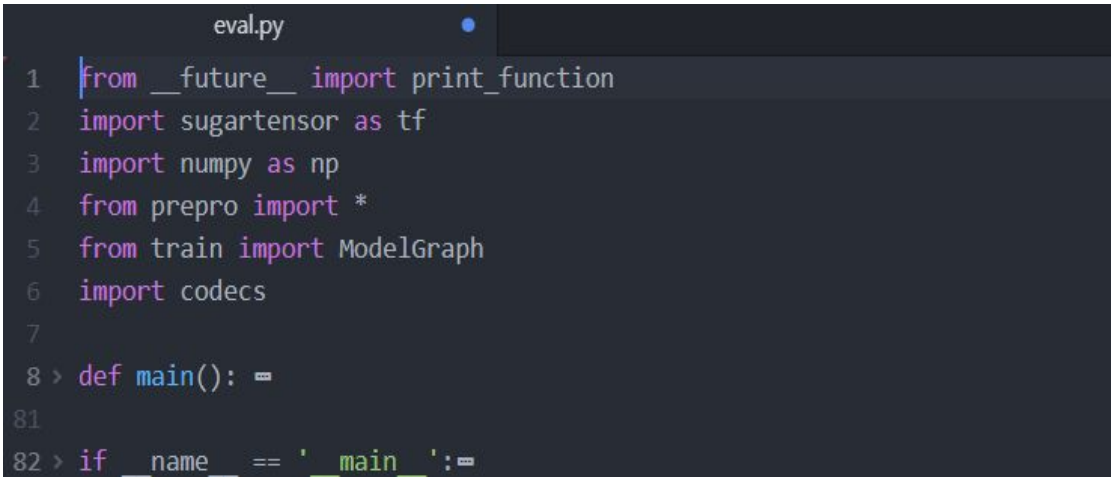
또한 아래의 tf.sg_train()의 parameter 값 중 하나인 lr, 즉 Learning Rate가 성능과 직접적인 관계가 있다고 생각해 값을 바꾸고 다시 evaluation을 했지만 성능 결과값은 변화가 없었다.

```
def train():
    g = ModelGraph()
    print("Graph loaded!")

    tf.sg_train(optim="Adam", lr=0.00001, lr_reset=True, loss=g.reduced_loss, eval_metric=[], max_ep=20000,
                save_dir='asset/train', early_stop=False, ep_size=g.num_batch)
```

<그림 4.1.3.b inner code of 'train.py'>

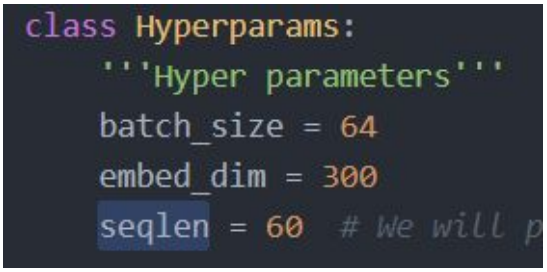
4.1.4. eval.py



<그림 4.1.4.a inner code of 'eval.py'>

eval.py는 train하여 생성된 그래프를 기반으로 하여 얻은 결과를 CSV 파일로 생성하는 코드다. 이 CSV 파일에는 예제로 사용된 단어들과, 시스템의 성능을 알 수 있는 지표인 RK를 보여 준다.

4.2. 개선



<그림 4.2.a code of 'prepro.py' from line 12 to line 16>

현재 예측할 단어 앞의 char의 개수를 나타내는 seqlen을 60으로 변환하였다. seqlen가 증가하면 전보다 프로그램이 문장의 문맥을 잘 파악하여 prediction accuracy가 상승하지만 대신 연산에 걸리는 시간이 많아진다. 단어 자동 완성은 실시간성이 중요하기 때문에 연산 시간과 정확도에 타협점을 찾아 60으로 설정하였다.

5. 결론

5.1. 구현 완성도

기존 프로그램이 어느정도 완성되어 있던 상태였고 본 프로젝트에서는 이 프로그램을 좀 더 향상시키는 것이 목적이였으나, 시간적 제약으로 인해 코드 분석조차 완벽히 끝내지 못하였다.

또한 본래 목표였던 아이폰7의 자동완성 성능을 앞서는 것은 도달하지 못했지만, 기존 성능보다 2.5만타자에서 17타자를 줄여 약 0.00068%가량 성능을 개선하였다.

5.2. 추가적인 개선 사항

기존 프로그램에서 Model Architecture를 수정하는 것이 가장 좋은 개선 방향이라고 생각한다. 본 프로젝트에서는 주어진 시간이 충분하지 못해 단지 seqlen만 바꿨기에 성능의 큰 변화는 없었다. Learning Rate 값을 수정하는 것도 시도해 볼만 하다. 또한 기존 프로그램이 아이폰을 이기지 못한 이유는 Data set이 너무 컸기에 아이폰 보다 단어 예측의 폭이 넓었고, 그로 인해 사용자가 원하는 단어보다 많은 단어를 표시했고 그 결과, 아이폰의 예측 기능보다 안좋게 나왔다고 생각할 수 있었다. 그리고 기존 Data set이 뉴스 정보에만 치우쳐져 있기에 사용자가 원하는 단어를 찾기에 한계도 분명 존재했을거라 생각하고 따라서 뉴스가 아닌 구어체가 담긴 영화나 드라마의 대본을 Data set으로 해본다면 일반 아이폰 사용자가 좀 더 사용하기에 적합할 것이다.

5.3. 고찰

방대한 학습량과 응용력을 요구하는 수업이었던 만큼 시간 투자를 많이 해야만 진도를 따라가며 내용을 이해할 수 있었던 강의였지만 프로젝트 완료까지 단 1주밖에 주어지지 않았기에 촉박한 일정 속에서 다방면으로 어려움을 겪었다. 그러나 프로젝트를 진행하며 딥러닝을 비롯한 여러 부분들에 대해 얕지만 전체적으로 훑을 수 있는 기회가 있어 뜻깊은 시간이었다.

- 딥러닝 매커니즘의 정의를 이해할 수 있었다.
- Tensorflow 및 Sugartensor 프레임워크를 설치하고 실행할 수 있었다.
- 파이썬 언어를 사용해 자연어 응용프로그램을 수정 및 디버그 할 수 있었다.
- 언어 모델의 정의와 적용범위에 대해 배웠다.
- 자동완성 기능의 기본적인 원리를 이해했다.
- 딥러닝 신경망 매커니즘의 성능을 높이기 위한 다양한 방법들을 실습을 통해 습득할 수 있었다.
- pretrained 된 프로그램을 실행하며 성능을 개선하는 방법을 터득할 수 있었다.

프로젝트를 진행하며 위와 같은 여러가지 부분들을 익힐 수 있었다. 또한 프로젝트를 급하게 마무리 지으면서 미처 다루지 못했던 부분들에 대한 아쉬움 또한 있었다. 그 중 가장 아쉬운 점을 꼽자면 먼저 프로그램의 성능을 뚜렷하게 개선하지 못했다는 점이다. 또한 아이폰의 경우 사람들이 실생활에 쓰는 단어를 예측하는 것이 일반적인데, 이에 반해 data set으로 가져온 wikinews는 뉴스에서 쓰이는 단어로, 실생활보다 더 많은 단어를 가져오기 때문에 단어를 예측하는 데 걸리는 시간이 많을 뿐더러 관련성 또한 떨어진다는 단점이 있었다. 만약 프로젝트를 하는 데 시간이 조금 더 주어진다면, 이 부분을 중점적으로 개선하기 위해 노력하지 않았을까 싶다.

6. 참고문헌

https://www.slideshare.net/HeeWonPark11/ss-80653977 딥러닝 기본원리
https://github.com/songhune/F039-1_AJ_Intensive 프로젝트 기본 설명
https://github.com/Kyubyong/word_prediction 자동완성 프로그램