Korean Journal of English Language and Linguistics
Vol. 20, March 2020, 42−63
10.15738/kjell.20..202003.42

딥러닝을 활용한 감정 분석 과정에서 필요한 데이터 전처리 및 형태 변형

서혜진 (동국대학교)
신정아 (동국대학교)*

Seo, Hye−Jin and Jeong−Ah Shin. 2020. Data preprocessing and transformation in the sentiment analysis using a deep learning technique. *Korean Journal of English Language and Linguistics* 20, 42−63. This study examined how to preprocess and transform data efficiently in order to use deep learning techniques in analyzing linguistic data. Researchers' interests in deep learning techniques have explosively increased worldwide; however, it is not easy for them to link linguistics to deep learning techniques or algorithms because linguists do not know how and where to begin in using them. Thus, this study provides the general procedure to train data using deep learning algorithms in practice. In particular, for instance, we focused on how to preprocess and transform Tweet data for a sentiment analysis by using deep learning techniques. In addition, we introduced the latest deep learning algorithm, so−called BERT, in the data preprocessing and transformation procedure. The data preprocessing is particularly important because the result from deep learning can significantly vary depending on it. Even though the data preprocessing procedure can differ according to the aim of research, this study tries to introduce the general way that advanced researchers frequently use for deep learning algorithms. This study is expected to lower the barriers in applying deep learning techniques to linguistic data and make it easier for researchers to conduct deep learning research related to linguistics.

Keywords: data preprocessing, transformation, sentiment analysis, deep learning

1. 서론

최근 10년 동안의 기술 발달은 인공 지능(AI, Artificial Intelligence) 기법을 근간으로 하고 있다. 특히, 구글 딥마인드의 딥러닝(Deep Learning) 기법을 이용한 바둑 인공지능 알파고(AlphaGo)와 최근 게임 인공지능 알파스타(AlphaStar)는 인공지능과 딥러닝 기법에 대한 대중적 관심을 폭발적으로 증대시켰다. 인공지능은 사람처럼 컴퓨터가 스스로 의사결정을 할 수 있도록 데이터를 컴퓨터에 학습시켜서 구현한다. 학습시키는 과정을

* 제1저자: 서혜진, 교신저자: 신정아

기계학습(Machine Learning)이라고 하며, 딥러닝 기법은 기계학습 중 하나의 기술이라고 할 수 있다. 예를 들어, 개와 고양이를 컴퓨터가 구별할 수 있게 하기 위해서 컴퓨터를 학습시킬 때, 기존의 전통적인 기계학습 방식은 컴퓨터가 이미지를 구별 할 수 있도록 개와 고양이의 특징(feature)을 사람이 부여해주며 컴퓨터를 학습시키는 방법인 반면에, 딥러닝은 컴퓨터가 특징을 스스로 찾아서 학습하는 방법이다. 스스로 학습하면서 성장해 나가는 딥러닝 기법은 전 세계 모든 분야에 인공지능 연구 개발의 필요성을 보여주게 되었다.

딥러닝 기술이 발전함에 따라서 언어학의 중요성이 부각되고 있다. Linzen(2019)에서는 현재 컴퓨터 성능의 발달 및 풍부한 데이터 자원으로 인하여 굉장히 빠른 속도로 딥러닝 기술이 발전하고 있지만 딥러닝 기술이 어느 정도까지 사람처럼 판단을 하는지에 대한 척도가 부족하다고 하고, 언어학 연구가 딥러닝 기술이 언어를 어느 정도 학습하였는지 알아볼 수 있는 중요한 역할을 할 수 있다고 주장하였다. 예를 들면, Linzen, Dupoux와 Goldberg(2016)은 딥러닝 기술 중 하나인 순환신경망(Recurrent Neural Network; RNN)이 과연 계층 통사형태론적 의존(a hierarchical morphosyntactic dependency) 현상을 학습하는지를 알아보기 위해서 ‘The key to the cabinets’ 주어에는 ‘were’가 아닌 ‘was’ 동사가 뒤따라야 한다는 주어-동사 일치(Subject-verb agreement) 현상을 활용하여 딥러닝 기술을 테스트하였다. 순환신경망이 ‘The key to the cabinets were..’ 라는 문장이 나오면 비문법적인 문장으로 판단하고, ‘The key to the cabinets was..’ 라는 문장이 나오면 문법적인 문장으로 옳게 판단한 결과를 토대로, 딥러닝 학습 과정에서 주어-동사 일치 현상을 학습했다고 주장하였다.

한편, McCoy, Frank와 Linzen(2018)에서는 순환신경망 모델 중 하나인 시퀀스-투-시퀀스 순환신경망(sequence-to-sequence RNN) 모델이 영어 예-아니오 질문을 옳은 형식으로 만들 수 있는지 살펴보았다. 평서문 ‘My walrus that will eat can giggle’은 ‘Can my walrus that will eat giggle?’과 같은 의문문 형식을 갖는다. 시퀀스-투-시퀀스 순환신경망 모델이 의문문을 형성할 때, 본동사의 조동사를 이동시킨다는 구조 가설(the structural hypothesis)을 따른다면 ‘Can my walrus that will eat giggle?’과 같은 의문문 형식을 출력값으로 도출할 것이고, 가장 왼쪽에 있는 조동사를 이동시킨다는 선형 가설(the linear hypothesis)을 따른다면 ‘Will my walrus that eat can giggle?’과 같은 의문문 형식으로 잘못된 출력값을 도출할 것이다. 그 모델 학습 결과, 시퀀스-투-시퀀스 순환신경망 모델이 마치 구조 문법을 학습한 것처럼 의문문을 옳은 형식으로 도출하였다. 딥러닝 기술 발전이 굉장히 빠른 속도로 이루어지고 있지만, 아직까지 언어학 연구를 딥러닝 기술에 적용시킨 연구는 앞에 제시한 두 연구뿐이다. 앞으로 영어학에서 다루는 다양한 언어 현상을 토대로 딥러닝 기술의 성능을 판단한다면 굉장히 객관적인 척도가 될 수 있을 것이다.

또한, 다양한 분야에서 딥러닝과 관련한 연구가 활발히 진행되고 있지만, 언어학

연구자들이 딥러닝을 어떻게 접근해야하고, 언어학 분야에 어떻게 활용해야 하는지에 대한 논의는 많지 않다. 언어학에 딥러닝을 어떻게 활용할지에 대한 관심은 많지만, 손쉽게 접근할 수 있는 가이드라인은 아직 언어학 분야에서 찾기는 쉽지 않다. 그래서 본 연구에서는 주어진 문장이 문법적인지 비문법적인지 또는 긍정적인지 부정적인지와 같이 두 개의 항목으로 문장을 판단하는 이진 분류(Binary classification) 딥러닝 학습을 위해서 필요한 기본적인 전처리 과정과 데이터 형태 변형에 대해서 알아보고 쉽게 따라 활용할 수 있는 방법을 공유하고자 한다.

딥러닝 모델을 구축할 때, 데이터 전처리(data preprocessing)는 반드시 거쳐야 하는 과정이며 가장 많은 시간을 할애해야하는 부분이다. 포보스 2016년 3월 23일 기사[1]에 따르면, 데이터 과학자들은 일을 하는 동안 80% 정도는 데이터를 수집(collecting)하고 정제(cleaning)하는데 시간을 보낸다는 설문 조사 결과를 보고하였다. 데이터 분석의 기본 전제는 '쓰레기를 넣으면 쓰레기가 나온다(Garbage in, garbage out)'이다. 어떻게 데이터 전처리를 하는지에 따라서 딥러닝 분석 모델의 결과도 확연하게 차이나기 때문에 정확한 데이터 분석 결과를 얻기 위해서는 잘 정제된 데이터를 입력해야한다. 하지만 딥러닝 모델을 구축하는 방법에 대해서 알리는데 주안점을 두고 있는 인터넷상의 튜토리얼 및 출판된 책을 살펴보면 데이터 전처리 과정을 간략하게 설명하고 넘어가며 주로 이미 정제된 데이터를 가지고 딥러닝 모델을 학습시킨다. 실제로 딥러닝 관련 연구를 진행하고자 했을 때는 정제된 데이터보다는, 미가공 데이터(raw data)를 가지고 연구를 진행해야 되는 경우가 대다수이고 연구 목적에 따라서 전처리 과정이 상이해지기 때문에 데이터 전처리 과정에서 상당한 어려움을 겪게 된다.

또한 딥러닝 학습을 위한 자연어 처리에서는 사람이 구사하는 자연어를 컴퓨터가 이해 할 수 있도록 자연어를 숫자로 구성된 벡터 값으로 변환하는 임베딩(Embedding) 과정이 있다. 데이터 속의 각 단어를 어떻게 표현하는지에 따라서 자연어 처리 결과가 상당히 달라질 수 있기 때문에 임베딩 과정 전에 컴퓨터가 좀 더 정확하고 빠르게 자연어 처리를 할 수 있도록 데이터를 전처리하여 핵심 단어들로 데이터를 구성하는 과정이 필요하다. 예를 들어, 영어의 대문자로 구성된 단어('BOY')와 소문자로 구성된 단어('boy')가 있다면 사람은 형태만 다른 동일한 의미의 단어라고 생각하지만 컴퓨터는 완전히 다른 두 단어로 인식을 한다. 이는 컴퓨터가 처리해야 할 단어의 가짓수가 더 많아진다는 것을 의미한다. 이렇게 컴퓨터가 처리해야 할 단어의 가짓수가 많아지게 된다면 임베딩 해야하는 단어들이 많아지게 되어 결국 딥러닝 자연어 처리 결과에 안 좋은 영향을 끼칠 수 있다. 그러므로 형태가 다른 같은 의미의 단어들을 정제하여 핵심어들로 데이터를 구성해주는 과정이 필요하다.

[1] https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/#2fdbae4f6f63

본 연구에서는 데이터 수집부터 정제하기, 그리고 이진 분류 딥러닝 학습을 위한 데이터 형태 변형 과정까지 상세하게 다루면서 연구자들에게 언어학 연구에 딥러닝을 활용 할 수 있도록 전반적인 데이터 처리 과정을 다루고자 한다. 본 논문의 구성은 다음과 같다. 먼저 2장에서는 데이터 특성 파악 및 전처리 과정을 기술한다. 3장에서는 딥러닝 학습을 위한 데이터 형태 변형에 관해서 다룬다. 이후 4장에서는 연구의 의의를 살펴보고자 한다.

2. 데이터 특성 파악 및 전처리[2]

2.1 딥러닝 모델 학습 및 검증 데이터 특성 파악

본 연구에서는 영어 데이터 전처리 과정에서 일반적으로 고려하는 전처리 과정을 살펴보기 위해서 다양한 데이터 정제 과정이 필요한 트위터 데이터를 이용하여 감정 분석(Sentiment analysis)을 이용하고자 한다. 본 연구에서 소개되는 전처리 과정은 트위터 데이터 이외의 영어 데이터 전처리 과정에도 응용할 수 있다. 트위터 데이터를 이용한 감정 분석(Sentiment analysis)은 딥러닝 모델로 빈번하게 구현되고(예, 서혜진, 이종현, 신정아 2019, Severyn and Moschitti 2015), 자연어 처리(Natural Language Processing; NLP) 분야에서 연구가 많이 되고 있다. 감정 분석은 정형화되지 않은 텍스트 내의 주관적인 감정, 태도, 성향 등과 같은 개인의 의견들을 파악하여 객관적이고 체계적으로 감정을 수치화하는 분석법으로써, 일반적으로 주어진 텍스트가 긍정적인지 아니면 부정적인지에 관한 텍스트의 편향성(polarity)을 파악한다. 최근에는 다양한 소셜 네트워크 서비스(Social Networking Service; SNS)의 발달로 개인의 의견을 공개된 소셜 네트워크 서비스에 적극적으로 표현하기 때문에 여론 조사, 기업 마케팅 전략, 서비스 개선의 방향성 등 다양한 분야에 감정 분석이 활용되고 있다(TTA정보통신용어사전 2012).

본 연구에서는 온라인에 구축 되어있는 감정 분석을 위한 훈련/검증 데이터[3]를 활용하여, 데이터 전처리 과정과 데이터 형태 변형에 대해서 다루고자 한다. 훈련 데이터는 딥러닝 모델을 학습 시킬 때 사용하는 훈련 데이터이며, 검증 데이터는 딥러닝 모델 학습이 끝났을 때 구축된 딥러닝 모델이 어느 정도 실효성 있는지 확인할 때 사용하는 데이터이다. 일반적으로 전체 데이터에서 훈련 데이터와 검증 데이터는 8:2의 비율로 나눈다. 훈련 및 검증 데이터는 동일한 전처리 과정과 형태 변형 과정을 거치며, 딥러닝 모델 구축 바로 직전에 훈련 데이터와 검증 데이터를 분리하므로, 이하 훈련 및 검증 데이터를 통칭하여 학습 데이터라고 부르겠다. 본 연구에서 학습 데이터는 100만 개의 트위터 데이터로

[2] 본 연구와 연관된 데이터 처리 및 변형 코드는 다음의 링크에서 살펴볼 수 있다. http://bit.ly/2SppWqa

[3] https://github.com/akanshajainn/Sentiment-Analysis-Twitter-word2vec-keras

구성하였고[4], 트위터 데이터가 긍정적인 의미 일 때는 1, 부정적인 의미 일 때는 0으로 이진 분류(binary classification)로 구분되어있다. 이진 분류 딥러닝 모델을 구축할 때, 학습 데이터에서 편향성의 데이터 개수를 균등하게 맞춰주어야 하기 때문에 전체 데이터의 50만 개는 긍정적 의미의 트위터 문장이며, 50만 개는 부정적인 의미의 트위터 문장이다.

표 1. 트위터 데이터 예시

편향성	트위터 데이터
0	@terinea that's what I was trying to find! Thanks, but it doesn't seem to work on Windows 7
0	It's so wrong to be in work on a Sunday
1	Taking a break at practice. I love her so much.
1	@PaulaAbdul you must have FUN!dont work too hard today Paula. I just joined your site hehe

다음 절에서는 트위터 언어의 특성에 따른 전처리 방법을 세부적으로 다루며, 다음과 같은 트위터 데이터의 전처리 과정을 단계별로 다루어보고자 한다.

(1) 부정형이 축약형으로 표현된 경우
"@terinea cooooool that's what I was trying to find! THANKS, but it **doesn't** seem to work LOL 😀 on Windows 7 https://official-kmspico.com/windows-7-iso-download/ #Windows7"

(2) 부정형이 구어체로 표현된 경우
"@terinea cooooool that's what i was trying to find! THANKS, but it **doesnt** seem to work lol 😀 on windows 7 https://official-kmspico.com/windows-7-iso-download/ #windows7"

트위터에서 사용하는 언어는 인터넷 및 휴대전화, 컴퓨터 통신에서 사용하는 통신 언어로써 이모티콘, 이모지, 축약형 사용, 생략 등과 같은 다양한 언어학적 현상이 나타나는 언어이다(이정복, 김봉국, 이은경, 하귀녀 2000). (1)과 (2) 같은 문장은 트위터에서 사용하는 다양한 통신언어 및 비전형적인 텍스트로 구성이 되어있기 때문에 트위터 데이터의 특징을 잘 보여주는 데이터이다. 딥러닝 연구를 할 때, 연구하고자 하는 텍스트의

[4] 원본 데이터는 약 160만 개의 트위터 데이터로 구성이 되어있는데, 본 연구에서는 160만 개의 데이터 중에서 무작위로 100만 개를 학습 데이터로 선정하였다.

특징에 따라서 전처리 과정이 달라질 수 있기 때문에 연구 데이터의 특징을 잘 파악하여 어떠한 전처리를 수행할지 판단하여야 한다.

2.2. 데이터 전처리를 위한 단계별 프로세스

2.2.1 소문자화

앞에서 언급했듯이 딥러닝 모델을 구축할 때는 컴퓨터가 사람처럼 자연어를 처리하기 위해서 컴퓨터가 인식할 수 있도록 언어를 변환하는 과정이 필요하다. 연구 목적에 따라서 대문자로 구성된 단어와 소문자로 구성된 단어를 구분 안 할 경우에는 모든 문자열을 소문자화하여 'THANKS'와 'thanks'를 같은 단어로 처리하게 만들어주어야 한다. 이때, 그림1에서와 같이 (1)의 트위터 글을 파이썬(Python) 컴퓨터 언어의 lower() 명령어를 사용하면 텍스트를 쉽게 소문자화 할 수 있다.

```
1 txt = """@terinea cooooool that's what I was trying to find! THANKS, but it doesn't
2 seem to work LOL 😀 on Windows 7 https://official-kmspico.com/windows-7-iso-download/
3 #Windows7"""
```

```
1 print(txt.lower())
```

```
@terinea cooooool that's what i was trying to find! thanks, but it doesn't
seem to work lol 😀 on windows 7 https://official-kmspico.com/windows-7-iso-download/
#windows7
```

그림 1. 텍스트 소문자화

2.2.2 단어 토큰화(Word Tokenization)

데이터 전처리 과정에서는 연구 목적 및 용도에 맞는 토큰화 과정이 필요하다. 토큰(token)의 단위는 주로 의미를 갖는 단어(word)라고 정의한다. 파이썬에서는 자연어를 쉽게 처리 할 수 있도록 제공하는 NLTK(Natural Language Toolkit) 패키지에 다양한 토큰화 라이브러리가 있다. 단어의 축약어(contraction) 및 구두점(punctuation) 등을 개별적인 토큰의 단위로 처리할 것인지 처리하지 않을 것인지에 따라서, 상황에 맞는 토큰화 라이브러리를 사용해야 한다. 일반적으로 쓰는 토큰화 라이브러리는 세 가지 종류가 있다(word_tokenize, WordPunctTokenize, TweetTokenizer). 그림 2에서 살펴 볼 수 있듯이, 라이브러리에 따라서 축약어를 처리하는 방식이 달라지는 것을 확인 할 수 있다.

```
from nltk.tokenize import word_tokenize
print(word_tokenize(txt.lower()))
```

```
['@', 'terinea', 'cooooool', 'that', "'s", 'what', 'i', 'was', 'trying', 'to', 'find', '!',
'thanks', ',', 'but', 'it', 'doesnt', 'seem', 'to', 'work', 'lol', '😀', 'on', 'windows',
'7', 'https', ':', '//official-kmspico.com/windows-7-iso-download/', '#', 'windows7']
```

```
from nltk.tokenize import WordPunctTokenizer
print(WordPunctTokenizer().tokenize(txt.lower()))
```

```
['@', 'terinea', 'cooooool', 'that', "'", 's', 'what', 'i', 'was', 'trying', 'to', 'find',
'!', 'thanks', ',', 'but', 'it', 'doesnt', 'seem', 'to', 'work', 'lol', '😀', 'on', 'window
s', '7', 'https', '://', 'official', '-', 'kmspico', '.', 'com', '/', 'windows', '-', '7', '-
', 'iso', '-', 'download', '/', '#', 'windows7']
```

```
from nltk.tokenize import TweetTokenizer
print(TweetTokenizer().tokenize(txt.lower()))
```

```
['@terinea', 'cooooool', "that's", 'what', 'i', 'was', 'trying', 'to', 'find', '!', 'thanks',
',', 'but', 'it', 'doesnt', 'seem', 'to', 'work', 'lol', '😀', 'on', 'windows', '7', 'http
s://official-kmspico.com/windows-7-iso-download/', '#windows7']
```

그림 2. NLTK 라이브러리에 따른 단어 토큰화

표 2. 라이브러리에 따른 단어 토큰화

라이브러리	that's	doesn't	토큰 개수
word_tokenize	'that', "'s"	'does', "n't"	2
WordPunctTokenize	'that', "'", 's'	'doesn', "'", 't'	3
TweetTokenizer	"that's"	"doesn't"	1

특히, “that's”와 “doesn't”을 어떻게 처리하는지를 살펴보면, 각 라이브러리의 특징을 쉽게 확인할 수 있다. 표 1에서 살펴보면, word_tokenize 라이브러리의 경우 각각 두 개의 토큰으로 나누었다. WordPunctTokenize 라이브러리의 경우, word_tokenize 라이브러리와 달리 어퍼스트로피(apostrophe)를 독립적으로 토큰화하여 각각 세 개의 토큰으로 나누었다. TweetTokenizer 라이브러리의 경우, 문자열 그대로 1개의 토큰으로 처리한 것을 볼 수 있다. 또한 @ 및 http 링크, 해시태그(hashtag; #)는 트위터 데이터의 고유 특성으로써 TweetTokenizer에서는 다른 토큰화 라이브러리와 달리 단일한 토큰으로 처리하는 것을 살펴볼 수 있다(2.2.4 참조). 본 연구에서는 축약어 처리 및 http 주소 처리와 해시태그(hashtag)와 같은 트위터 텍스트를 처리하는데 최적화된 TweetTokenizer 라이브러리를 사용하여 단어 토큰화 처리를 하였다.

2.2.3 축약어(Contractions)

TweetTokenize의 경우 “that's”와 “doesn't” 같은 축약어를 문자열 그대로 1개의 토큰으로 반환하는 것을 살펴보았다. 감정 분석에서 ‘not’은 굉장히 큰 의미를 갖는

단어이기 때문에 TweetTokenize로 토큰화 한 이후에 “that’s”는 ‘that’과 ‘is’[5]로, “don’t”는 ‘do’와 ‘not’으로 더 세부적으로 토큰화 하는 과정을 거쳤다.

이 과정에서 트위터 데이터가 비형식적인 구어적 요소가 많이 표현된다는 것을 유념해야 하는데, 트위터 데이터의 큰 특징 중의 하나는 “isn’t”과 “don’t”와 같은 표현을 축약어의 ’(apostrophe)를 임의적으로 생략하여 ‘isnt’와 ‘dont’로 쓰는 경우가 굉장히 많기 때문이다. 따라서 ‘isnt’은 ‘is’와 ‘not’의 두 개의 토큰으로, ‘dont’는 ‘do’와 ‘not’의 두 개의 토큰으로 토큰화하여야 하다. 다음과 같이 구어적 표현을 고려하여 더 세부적으로 토큰화하는 과정은 NLTK 라이브러리에 없기 때문에 따로 트위터 데이터 처리에 적합한 dictionary 타입의 contractions.py 파일을 따로 만들어서 처리하였다.

표 3. 세부적 토큰화 예시

입력값	출력값
“is’t”	‘is’, ‘not’
“don’t”	‘do’, ‘not’
“isnt”	‘is’, ‘not’

(2)과 같은 데이터를 TweetTokenizer로 토큰화 처리한 이후에 세부 토큰화 과정을 한 번 더 거치게 되면 그림 4과 같이 최종 출력이 된다.

```
['@terinea', 'coooooool', "that's", 'what', 'i', 'was', 'trying', 'to', 'find', '!', 'thanks',
',', 'but', 'it', 'doesnt', 'seem', 'to', 'work', 'lol', '😬', 'on', 'windows', '7', 'http
s://official-kmspico.com/windows-7-iso-download/', '#windows7']
```

그림 3. TweetTokenizer로 토큰화

```
['@terinea', 'coooooool', 'that', 'is', 'what', 'i', 'was', 'trying', 'to', 'find', '!', 'than
ks', ',', 'but', 'it', 'does', 'not', 'seem', 'to', 'work', 'lol', '😬', 'on', 'windows',
'7', 'https://official-kmspico.com/windows-7-iso-download/', '#windows7']
```

그림 4. TweetTokenizer로 토큰화 후 세부 토큰화

그림 4에서 볼 수 있듯이 입력값으로 “dont”가 입력이 되면 출력값으로 ‘do’와 ‘not’의 두 개의 토큰으로 분절된다.

[5] “that’s”는 ‘that is’ 또는 ‘that has’의 의미를 가질 수 있지만 ‘is’와 ‘has’는 불용어(stopwords)로 처리되어 다음 단계에서 제거되기 때문에 우선은 일괄적으로 ‘that is’로 처리하였다.

2.2.4 트위터 고유 표기 제거

트위터에는 자신의 의견 및 생각을 보다 더 잘 표현하기위한 추가 기능인 리플라이(reply), 해시태그(hashtag), 링크 공유가 있다. 예를 들어, (2)과 같은 트위터 데이터가 있다면, 리플라이(reply) 기능에서는 특정인에게 말을 걸던가 응답을 하는 것으로써 @에 뒤따르는 아이디인 terinea에게 “that's what I was trying to find! Thanks, but it doesn't seem towork on Windows 7”과 같이 트위터 화자가 응답을 한 것이다. 리플라이 기능은 항상 @표시와 함께 동반된다. 해시태그(hashtag; #) 기능은 트위터 내용의 핵심어라고 생각하면 된다. 본인의 트위터 글 중에서 핵심어들을 # 뒤에 작성한다. 마지막으로 링크 공유 기능이 있다. 공유하고 싶은 웹 상의 링크 주소를 트위터 글과 함께 써서 좀 더 확실한 정보 공유를 할 수 있다. 감정 분석을 위해서 리플라이, 해시태그, 링크 공유와 같은 기능은 큰 의미가 없기 때문에 @ 및 #, http와 같은 표현들은 제거하였다(그림 5).

```
['cooooool', 'that', 'is', 'what', 'i', 'was', 'trying', 'to', 'find', '!', 'thanks', ',', 'b
ut', 'it', 'does', 'not', 'seem', 'to', 'work', 'lol', '', 'on', 'windows', '7']
```

그림 5. 트위터 리플라이와 해시태크, 링크 토큰 제거

2.2.5 이모티콘 및 이모지 제거

사람의표정이나 감정을 문자 및 숫자, 기호로 표현한 것은 이모티콘(>_<,ㅠㅠ, ^0^)이며, 일본어 ‘그림(에)’와 ‘문자(모지)’의 합성어로써 이모티콘과 비슷하지만 이미지 자체가 하나의 문자로 취급되는 언어 표현은 이모지이다(예: 😀, 🤣, 😻등이 있음). 본 연구에서는 문자열 텍스트에 더 초점을 맞추어 감정 분석을 하고자하기에, 트위터 데이터에 이모티콘 및 이모지가 포함된 토큰은 제거하였다(그림 6).

```
['cooooool', 'that', 'is', 'what', 'i', 'was', 'trying', 'to', 'find', '!', 'thanks', ',', 'b
ut', 'it', 'does', 'not', 'seem', 'to', 'work', 'on', 'windows', '7']
```

그림 6. 이모티콘과 이모지 토큰 제거

2.2.6 영문자 이외의 문자를 공백으로 변환 후 공백 제거

연구의 목적에 따라서 숫자 및 기호들을 전처리 과정에서 제거할지 안 할지를 결정해야 한다. 본 연구에서는 숫자 및 기호로 된 표현들보다 문자열로 된 토큰에 더 주안점을 두기 위해서 영문자 이외의 문자를 공백으로 변환하였다. 그 이후에 공백으로 토큰화 된 부분을

제거하였다(그림 7).

```
['cooooool', 'that', 'is', 'what', 'i', 'was', 'trying', 'to', 'find', 'thanks', 'but', 'it',
'does', 'not', 'seem', 'to', 'work', 'on', 'windows']
```

그림 7. 영문자 이외의 토큰 제거

2.2.7 무의미하게 반복된 철자 수정

트위터 데이터에서는 감정 표현을 위해서 철자를 반복해서 강조하는 경우가 많다(Yeeeeeeeeeah, Cooooooooooool). 무의미하게 여러 번 반복되는 철자를 최대 2개로 만드는 전처리 과정을 거쳤다(yeeah, cool). 만약 모든 단어에서 같은 철자가 두 번 이상 되는 것을 하나의 철자로 바꾸면 실제 존재하는 'good'과 같은 단어가 'god'으로 바뀔 오류를 범할 수 있다(그림 8 참고).

```
['cool', 'that', 'is', 'what', 'i', 'was', 'trying', 'to', 'find', 'thanks', 'but', 'it', 'do
es', 'not', 'seem', 'to', 'work', 'on', 'windows']
```

그림 8. 여러 번 반복되는 철자 수정한 토큰

2.2.8 표제어 추출(Lemmatization)

데이터에서 유의미한 단어들을 최대한 추출해 내기 위해서 사용하는 방법에는 표제어 추출(lemmatization)과 어간 추출(stemming)이 있다. 표제어 추출과 어간 추출은 서로 다른 단어이지만 최대한 하나의 단어로 일반화 시켜서 전체 데이터의 단어의 수를 줄여서 의미 있는 단어들을 찾아내는 과정이다. 딥러닝을 구축 할 때 단어의 빈도수를 기반으로 단어의 특성을 벡터화 하는데 단어의 개수를 최대한 줄여서 데이터에서 좀 더 핵심적인 단어들을 이끌어 내는 작업이다. 표제어 추출은 단어의 기본 사전형 단어를 찾는 과정이다. 예를 들어, 'is, was, were'와 같은 단어들의 기본 사전형 단어는 'be'이다(그림 9). 이와 달리 어간 추출은 단어의 형태소적 형태를 간소화한 것이다(그림 10). 예를 들어, 'going, doing, having'과 같은 단어들의 어미의 '-ing'를 제거하여 'go, do, hav'로 출력한다. 이 작업은 영어의 일반적인 형태소 규칙을 정교하지 않게 제거하는 것이므로 실제로 존재하지 않는 단어가 출력될 수도 있다. 자연어 처리에서는 어간 추출보다는 표제어 추출 방법을 선호하므로, 본 연구에서는 토큰의 표제어 추출을 하면서 데이터 내에서 핵심 단어들을 추출하고자 하였다.

```
['cool', 'that', 'be', 'what', 'i', 'be', 'try', 'to', 'find', 'thanks', 'but', 'it', 'do',
'not', 'seem', 'to', 'work', 'on', 'window']
```

그림 9. 표제어 추출 토큰

```
['cool', 'that', 'is', 'what', 'i', 'was', 'tri', 'to', 'find', 'thank', 'but', 'it', 'doe',
'not', 'seem', 'to', 'work', 'on', 'window']
```

그림 10. 어간 추출 토큰

2.2.9 불용어(stopwords) 제거

불용어란 큰 의미가 없는 단어 토큰으로 대부분이 문법적인 기능을 위해서 쓰이는 기능어(function word)로 구성되어있다. 예를 들어, 영어에서 the, a, I, me, no, not, have, 조사, 접미사와 같은 단어들이다. 이러한 불용어는 문장에서 상당히 많이 사용되지만 명사와 동사와 같은 내용어(content word)에 비해서 상대적으로 문장의 핵심 내용을 내포하고 있지는 않다. 연구에 따라서 연구자는 이러한 불용어를 포함하여 분석할 것인지 포함하지 않고 분석할 것인지 판단하여야 한다. 본 연구에서는 문장 속 내용어를 중점적으로 살펴보며 감정 분석을 하기 위해서 불용어를 제거하였다. 파이썬 NLTK에서는 100여개 이상의 불용어를 정의하고 있는데 'no'와 'not'도 불용어에 포함되어있다. 하지만 본 연구에서는 감정 분석에 중요한 기여를 하는 'no'와 'not'을 제외한 그 이외의 불용어는 제거하였다(그림 11).

```
['cool', 'try', 'find', 'thanks', 'not', 'seem', 'work', 'window']
```

그림 11. 불용어 제거한 토큰

2.2.9 상대적으로 긴 문장 제거

그림 11에서 볼 수 있듯이 해당 문장의 토큰은 총 8개로 구성이 되어있다. 하지만 일부 데이터는 문장의 길이가 너무 길어서 상당히 많은 토큰으로 이루어질 수도 있다. 지나치게 긴 문장은 많은 내용을 포함하고 있기 때문에 해당 문장이 긍정적인지 부정적인지 이진 분류로 판단하는 것은 사람한테도 쉽지 않을 수 있다. 이러한 점을 제외하기 위해서 최대 토큰의 수는 25개로 제한하고, 26개 이상의 토큰으로 구성된 데이터는 학습 데이터에서 제외하였다. 이러한 과정을 다 거친 후, 최종적으로 100만 개의 학습 데이터가 완성되었다.[6]

[6] 2장 데이터 전처리 세부 과정은 본 연구와 연관된 데이터 처리 및 변형 코드(http://bit.ly/2SppWqa) 중에서 'Step 1. Eng_preprocessing.ipynb' 파일을 살펴보면 된다. 해당 파일

3. 딥러닝 모델 구축을 위한 학습 데이터 형태 변형

2절에서 전처리 과정을 마친 데이터를 가지고, 3절에서는 딥러닝 모델에 맞춰서 데이터 형식을 변경하는 방식에 대해서 알아보고자 한다.

3.1 단어 임베딩과 문장 임베딩의 종류 및 차이

임베딩이란 사람이 사용하는 자연어를 컴퓨터가 이해 할 수 있도록 각 단어나 문장을 벡터로 변환하는 것이다. 임베딩 방식에 따라서 각 단어나 문장은 n차원으로 표현한다(수식 1).

수식 1. ‘man’ 워드투벡(Word2Vec) 단어 임베딩
[5.58609292e-02, -7.68227398e-01, 2.66534209e+00, ..., 9.15688753e-01, -6.95801616e-01]

이렇게 단어나 문장을 고유의 벡터로 표현하게 된다면, 단어별 또는 문장별 벡터 연산 과정을 통해서 유사도(similarity)를 계산해서 컴퓨터가 사람이 추론하듯이 의미적인 관계나 문법적인 관계까지도 파악할 수 있다. 구체적으로 단어 벡터에서 첫 번째 단어(‘man’) - 두 번째 단어(‘woman’) + 세 번째 단어(‘king’)를 연산하여 유사도가 가장 높은 단어(‘queen’)를 찾을 수 있게 된다.

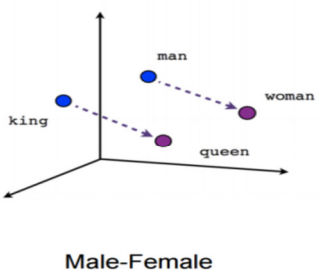


그림 12. 단어 벡터 표현[7]

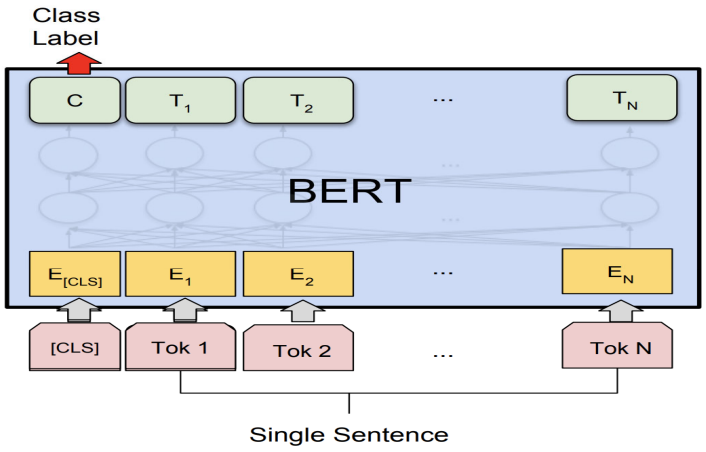
은 2장에서 다룬 전처리 세부 과정을 대량의 데이터에 일괄적으로 적용시키는 과정이 포함되어 있으며, 이러한 과정을 통해서 총 100만 개의 학습 데이터를 완성할 수 있다.

[7] 출처: https://medium.com/@jennyamy2531/word2vec-%EC%9C%BC%EB%A1%9C-%EB%8B%A8%EC%96%B4-%EC%9E%84%EB%B2%A0%EB%94%A9%ED%95%98%EA%B8%B0-word2vec-word-embeddings-96725bfb2580

2017년 이전 임베딩 기법은 주로 단어 수준이였으며 대표적인 단어 임베딩은 워드투벡(Word2Vec), 패스트텍스트(FastText), 글로브(Glove)이다. 단어 임베딩의 한계점은 고유의 벡터 값을 통해서 문맥적 단어 의미를 내포할 수 있지만 동음이의어를 분간하기 어렵다는 점이다. 의미가 다를지라도 동일한 형태의 단어라면 동일한 벡터값을 갖는다. 이러한 한계점을 극복하기위해서 2018년 초부터 문장 수준 임베딩 기법이 주목받기 시작하였다. 대표적인 문장 임베딩은 엘모(ELMo), 트랜스포머(Transformer), 버트(BERT)이다. 문장 임베딩은 단어 임베딩과 달리 각 단어별 의미보다는 문장 전체에서 표현된 주변 단어들을 바탕으로 각 단어의 의미를 함축한다(이기창 2019). 따라서 동음이의어 단어도 문장 형태나 그 단어가 사용된 위치에 따라서 다른 임베딩 값을 가질 수 있게 된다. 딥러닝 모델을 어떻게 구축하고자하는 목적에 따라서 임베딩 방식을 결정해야 한다. 본 연구에서는 구글이 개발한 문장 수준 임베딩 기법인 버트 언어모델 구축에 필요한 데이터 형태 변형에 대해서 자세히 다루고자 한다.

3.2 버트 모델의 특성

버트 이전 딥러닝 모델들은 현재까지 주어진 단어의 시퀀스(sequence)를 토대로 그 다음에 뒤따를 단어를 예측하며 학습하는 방식이다. Devlin, Chang, Lee와 Toutanova(2018)에서는 현재까지 주어진 단어 시퀀스 뿐만 아니라 전체적인 문맥까지도 고려할 수 있는 양방향 모델인 버트를 제안하였는데, 2018년 자연어 처리 분야에서 감정 분석과 문맥 이해, 질의응답 등과 같은 11개의 자연어처리 테스크에서 기존 딥러닝 기술 성능을 뛰어넘으며 최고의 성능을 내면서 범용적으로 많이 쓰이게 된 최첨단 딥러닝 모델이다. 버트는 약 33억 개의 단어로 구성된 대량의 텍스트 데이터(위키피디아와 북스코퍼스)로 사전 학습(pre-training) 되어있는 구글의 언어모델로써, 사전 훈련된 모델을 불러온 후 연구 주제에 맞춰서 간단하게 파인튜닝(fine-tuning) 하여 사용 할 수 있다. 기존 모델들은 연구자가 대량의 텍스트를 구축하고 오랜 시간을 거쳐서 딥러닝 모델을 학습 시켜야했기 때문에 개인 연구자들이 쉽게 딥러닝 학습을 시킬 수 없었던 단점들도 보완 하였다. 그림 13에서 볼 수 있듯이 입력층에 하나의 문장이 들어간다면 이 문장은 각각의 토큰으로 세분화되어 버트 층을 통과하게 되고, (이진 분류와 같은 감정 분석은) 처리한 문장이 얼마나 긍정적인지를 확률값으로 산출하여, 확률값이 0.5 이상이면 클래스 라벨은 긍정적인 문장에 가깝다는 의미인 1로 최종 출력이 되고, 0.5 이하이면 부정적인 문장에 가깝다는 의미인 0으로 최종 출력된다.



(b) Single Sentence Classification Tasks: SST-2, CoLA

그림 13. 이진 분류 학습을위한 버트 모델 (Devlin et al. 2018)

3.3 버트 딥러닝 학습을 위한 데이터 형태 변형[8]

2장에서 전처리 과정을 마친 데이터로 파이토치(Pytorch)로 구현 할 수 있는 버트 딥러닝 학습을 위한 데이터 형태 변형에 대해서 다루도록 하겠다.

(3) 전처리 전 트위터 데이터
'is upset that he can't update his Facebook by texting it... and might cry as a result School today also. Blah!'

(4) 트위터 데이터 전처리 후 문자열로 변환
'upset not update facebook texting might cry result school today also blah'

2장의 전처리 과정을 마치면 하나의 문장이 여러 개의 토큰으로 나누어진다. 문장 수준의 임베딩을 활용하는 버트 학습용 데이터로 만들기 위해 여러 개의 토큰을 문자열 데이터로 변환해주어야 한다. (3)과 같은 문장은 (4)과 같은 문자열이 된다.

[8] 3.3 절에서 사용한 코드는 '2020 전산언어학 겨울학교, 고려대학교 언어정보연구소 계산의미론연구실'과 ' https://mccormickml.com/2019/07/22/BERT-fine-tuning'에서 공유된 코드를 바탕으로 작성되었습니다.

3.3.1 식별자 삽입

버트에 입력층으로 문장을 전달할 때, 각 문장의 시작을 선언하는 [CLS] 식별자와 문장의 끝남을 선언하는 [SEP] 식별자가 필요하기 때문에 학습 데이터 처음과 끝에 식별자를 넣어주는 절차가 필요하다.

```
1 sentences = ["[CLS] " + str(sentence) + " [SEP]" for sentence in sentences]
2 sentences[0]
```

```
'[CLS] upset not update facebook texting might cry result school today also blah [SEP]'
```

그림 14. 식별자 삽입

3.3.2 워드피스 토크나이저(Wordpiece tokenizer)

각 문장이 토큰으로 나뉘어져서 버트 층을 통과하는데, 버트는 구글의 워드피스 토크나이저를 사용한다. 자연어 처리를 할 때 데이터 내의 모든 단어들을 벡터화 하는 것은 굉장히 많은 컴퓨터 프로세싱을 요구하며, 훈련 데이터에 존재하지 않는 빈도가 굉장히 낮은 단어들이 검증 데이터에 존재한다면 해당 문장은 학습시 존재하지 않았던 단어(Out of Vocabulary; OOV)로 인하여 처리가 올바르게 되지 않는 문제가 발생한다. 이러한 문제점을 보완하기 위해서 버트에서는 단어를 빈도수에 기반하여 가장 의미있는 최소한의 서브워드 유닛(subword unit)으로 토큰화하는 워드피스 토크나이저 방법을 사용한다. 그림 15와 같이 'texting'과 같은 한 단어가 'text'와 '##ing'[9]인 서브워드 유닛으로 분절된 것을 확인 할 수 있다.

```
1 tokenizer = BertTokenizer.from_pretrained('bert-base-multilingual-cased', do_lower_case=True)
2
3 tokenized_texts = [tokenizer.tokenize(sent) for sent in sentences]
4
5 print (tokenized_texts[0])
```

```
['[CLS]', 'upset', 'not', 'update', 'face', '##book', 'text', '##ing', 'might', 'c', '##ry', 'result',
 'school', 'today', 'also', 'bl', '##ah', '[SEP]']
```

그림 15. 워드피스 토크나이저

[9] 워드피스 토크나이저에서 ## 표시는 앞의 토큰과 이어진다는 의미이다.

3.3.3 버트용 임베딩

버트는 3가지(토큰 임베딩; Token Embedding, 세그먼트 임베딩; Segment Embedding, 위치 임베딩; Position Embedding) 입력 임베딩의 합으로 구성된다(그림 16). 토큰 임베딩은 3.2.2에서 워드피스 토크나이저로 분절된 문장을 토큰 단위로 임베딩을 해준다(그림 17). 세그먼트 임베딩에서는 연구에 따라서 두 개 이상의 문장으로 구성된 텍스트를 문장 단위로 나누어서 앞뒤 문장의 연관성까지도 고려한 임베딩을 할 수도 있다. 예를 들어, 'my dog is cute. he likes playing.' 이라는 두 개의 문장이 있다면 문장의 종료를 알리는 [SEP]를 여러 개 사용하여 '[CLS] my dog is cute [SEP] he likes playing [SEP]'와 같이 표현해주면 된다. 본 연구에서는 트위터 데이터 한 개를 문장 단위로 나누어서 분석하지 않으므로 트위터 데이터 한 개에 [SEP] 식별자 한 개를 사용하였다. 위치 임베딩은 단순하게 토큰의 순서대로 첫 번째 토큰은 0, 그 다음 토큰은 1과 같이 인코딩 된다.

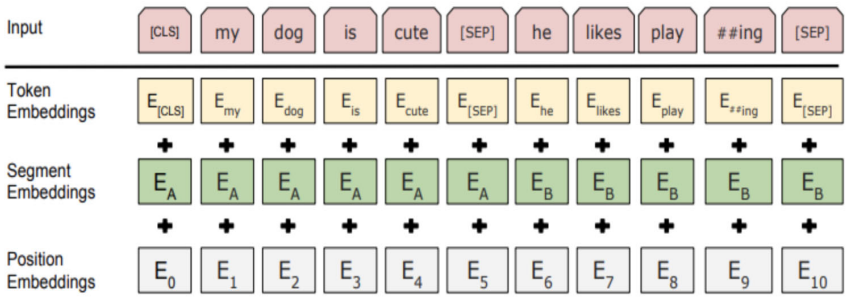


그림 16. 임베딩 (Devlin et al. 2018)

```
input_ids = [tokenizer.convert_tokens_to_ids(x) for x in tokenized_texts]
print(input_ids[0])
```

```
[101, 96213, 10472, 35896, 13295, 24777, 15541, 10230, 20970, 171, 10908, 14493,
11393, 18745, 10379, 21484, 12257, 102]
```

그림 17. 토큰 임베딩

3.3.4 문장의 길이 맞추기 및 패딩(padding)

데이터의 문장은 각각 다른 길이를 가지고 있기 때문에, 모든 데이터를 통일된 길이의 토큰으로 맞추는 작업이 필요하다. 연구 목적에 맞춰서 문장의 최대 길이를 어느 정도로 설정할 것인지 정해야한다. 2.2.9에서 언급했다시피 본 연구에서는 전처리를 마친 데이터는

최대 25개의 토큰을 갖는데, 이 토큰들이 워드피스 토크나이저를 사용하여 더 작은 서브워드 유닛 단위로 토큰화되는 것까지 고려하여 한 문장의 길이를 최대 50개의 토큰으로 설정하였다. 각 문장을 50개의 토큰의 길이로 맞추고, 토큰이 실제로 존재하지 않는 부분은 0으로 채워주는 패딩 작업을 하였다. 패딩 작업은 딥러닝 과정을 거치면서 데이터 형태가 축소되어 일부 데이터가 소실될 가능성을 방지하기위해서 사용하는 방식이다.

```
MAX_LEN = 50
input_ids = pad_sequences(input_ids, maxlen=MAX_LEN, dtype="long", truncating="post", padding="post")
input_ids[0]
```

```
array([  101, 96213, 10472, 35896, 13295, 24777, 15541, 10230, 20970,
         171, 10908, 14493, 11393, 18745, 10379, 21484, 12257,   102,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0])
```

그림 18. 패딩

3.3.5 어텐션 마스크(attention mask)

패딩 작업을 마친 토큰을 살펴보면 0으로 패딩된 부분은 실제로 존재하지 않는 단어들이다. 그렇기 때문에 컴퓨터가 0으로 패딩된 부분까지 굳이 프로세싱을 할 필요가 없기 때문에 이때 어텐션 마스크를 사용하여 버트 모델에서 0으로 패딩된 부분은 수행하지 않기 때문에 학습 속도를 높일 수 있다(그림 19).

```
attention_masks = []

for seq in input_ids:
    seq_mask = [float(i>0) for i in seq]
    attention_masks.append(seq_mask)

print(attention_masks[0])
```

```
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

그림 19. 어텐션 마스크

3.3.6 데이터 분리

학습 데이터가 총 100만 문장이라고 언급하였다. 버트 모델 학습을 위해서는 훈련 데이터셋과 검증 데이터셋으로 분리하는 작업이 필요하다. 명령어에 test_size를 0.2라고 설정하게 된다면 총 100만 개의 학습 데이터를 80개의 훈련 데이터와 20개의 검증 데이터로 분리해준다(그림 20). 이와 같은 방식으로 어텐션 마스크도 훈련 데이터와 검증 데이터로 분리해준다(그림 21).

```
train_inputs, validation_inputs, train_labels, validation_labels = train_test_split(input_ids,
                                                                                    labels,
                                                                                    random_state=2018,
                                                                                    test_size=0.2)
```

그림 20. 학습 데이터 분리

```
train_masks, validation_masks, _, _ = train_test_split(attention_masks,
                                                       input_ids,
                                                       random_state=2018,
                                                       test_size=0.2)
```

그림 21. 어텐션 마스크 분리

3.3.7 데이터 파이토치 형식으로 변환

파이토치는 텐서 연산으로 딥러닝 학습을 하므로, 각각의 데이터를 텐서 자료형 형태로 바꿔준다.

```
train_inputs = torch.tensor(train_inputs)
train_labels = torch.tensor(train_labels)
train_masks = torch.tensor(train_masks)
validation_inputs = torch.tensor(validation_inputs)
validation_labels = torch.tensor(validation_labels)
validation_masks = torch.tensor(validation_masks)
```

```
tensor([  101, 12014, 11424, 10751, 16118, 10751, 53510, 13246, 13961, 11940,
        20181,   102,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0])
```

그림 22. 텐서 형태로 바뀐 데이터

이 과정까지 마치게 된다면 파이토치에서 버트 모델을 사용할 수 있는 데이터 형태 변형까지 마친 것이다.

3.3.8. 버트 모델 결과값 살펴보기

3.3.7 항까지 처리한 데이터를 토대로 버트 모델 학습을 구현한 이후에 확인해 보고자하는 문장을 입력값으로 넣어주면 딥러닝 버트 모델은 해당 문장이 긍정적일 때는 1로 부정적일 때는 0의 결과값을 도출한다.[10] 그림 23과 그림 24에서 볼 수 있듯이, "I'm itchy and miserable!"과 같은 부정적인 문장을 입력하면 결과값으로 0을 반환하고, "downloading apps for my iphone! So much fun."과 같은 긍정적인 문장을 입력하면 결과값으로 1을 반환하는 것을 살펴볼 수 있다.

```
1 logits = test_sentences(["I'm itchy and miserable!"])
2 print(np.argmax(logits))
```

```
0
```

그림 23. 부정적인 문장으로 예측한 결과값

```
1 logits = test_sentences(['downloading apps for my iphone! So much fun.'])
2 print(np.argmax(logits))
```

```
1
```

그림 24. 긍정적인 문장으로 예측한 결과값

4. 연구의 의의

본 연구는 딥러닝 학습에 필요한 다양한 고려 사항들을 세부적으로 살펴보았다. 이러한 연구를 통해서 연구자들에게 딥러닝과 관련된 연구의 진입 장벽을 낮출 수 있을 것이라 기대한다. 특히, 최신 언어 모델인 버트를 이용하여 이진 분류 학습을 위한 데이터 전처리 과정 및 형태 변형에 대해서 알아보았다. 연구 데이터를 어떻게 전처리하는지에 따라서 딥러닝 학습 효과가 크게 달라질 수 있기 때문에, 연구자들은 연구 목적에 따라서 어떠한 전처리 과정을 거쳐야 할지 신중하게 판단하여야 한다. 본 연구에서는 감정 분석을

[10] 본 연구에서는 딥러닝 모델 학습 구축보다는 데이터 전처리에 초점을 두고 있기 때문에 딥러닝 모델 구축은 다루지 않는다. 버트 모델 구축은 데이터 처리 및 변형 코드(http://bit.ly/2SppWqa) 중에서 'Step 2. BERT_Model_tutorial.ipynb' 파일을 살펴보면 된다.

다루었지만 좀 더 영어학적인 측면으로 딥러닝 학습을 시키고자 한다면 본 연구에서 소개된 프로세스에서 학습 데이터만 바꾸어주면 된다. 좀 더 구체적으로 설명하자면, 본 연구의 학습데이터는 긍정적인 문장은 1로, 부정적인 문장은 0으로 표기되어 구성되어 있다. 만약 딥러닝 버트 모델이 주어-동사 일치 현상을 잘 포착하는지 살펴보고자 한다면, 'The key to the cabinets was on the table' 문장은 문법적인 문장이라는 의미로써 1로 표기를 하고, 'The key to the cabinets were on the table' 문장은 비문법적인 문장이라는 의미로써 0으로 표기하여 학습 데이터를 구성하면 된다.

최근 딥러닝 모델 기술은 굉장히 빠른 속도로 발전하며 높은 성능을 보여주고 있기 때문에 일반적인 텍스트를 가지고 각 모델들의 학습 효과를 살펴보는 데에는 한계가 있다. 언어학에서 다루는 다양한 언어 현상들을 딥러닝 모델의 학습 효과를 살펴보는 객관적이며 정교한 지표로 활용될 수 있도록 연구가 진행된다면, 이론적인 측면과 실용적인 측면에서 언어학 연구에 공헌할 수 있을 것이다. 또한, 이러한 연구는 딥러닝 모델이 인간과 비슷하게 언어를 처리하고 구사할 수 있게 발전시킬 수 있는 방향성을 제시할 수 있을 것이다.

References

이정복・김봉국・이은경・하귀녀. 2000. 바람직한 통신언어 확립을 위한 기초 연구. 연구보고서, 문화체육관광부.

이정복(Lee, J. B.). 2011. 트위터의 소통 구조와 통신 언어 영역(Communication Structure of the Twitter and its Type of Net-language). 《인문과학연구》(*Journal of Humanity Therapy*) 37, 235-270.

장세은・이경은・박호민・송원문・정해룡・이수상・김재훈(S-E. Jhang, K-E. Lee, H. Park, W-M. Song, H-R. Jung, S-S. Lee and J-H. Kim). 2019. 대화코퍼스를 통한 셰익스피어 비극 작품과 주요 남녀 등장인물 간의 감정분석(Sentiment Analysis of Shakespeare's Tragedy Plays and Their Major Male and Female Characters through Dialogue Corpora). 《언어과학》(*Journalof Language Sciences*) 26(1), 115-147.

허찬・온승엽(C. Heo, S-Y Ohn). 2017. Word2vec와 Label Propagation을 이용한 감성사전 구축방법(A Novel Method for Constructing Sentiment Dictionaries using Word2vec and Label Propagation). 《한국차세대컴퓨팅학회 논문지》(*The Journal of KoreanInstitute of Next Generation Computing*) 13(2), 93-101

허상희・최규수(S. H. Hur, K. S. Choi). 2012. 트위터에서 트윗 (tweet) 의 특징과 유형 연구(A Study on characteristics and types of tweet in twitter). 《한민족어문학》

61, 455-494.

Devlin, J., M. W. Chang, K. Lee and K. Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805.*

Gilbert, E. and C. J. Hutto. 2014. VADER: Aparsimonious rule-based modelfor sentiment analysis of social media text. *Eighth International Conference on Weblogs and Social Media,* 216-225.

Linzen, T. 2019. What can linguistics and deep learning contribute to each other? Response to Pater. *Language,* 95(1), e99-e108.

Linzen, T., E. Dupoux and Y. Goldberg. 2016. Assessing the ability of LSTMs to learn syntax-sensitive dependencies. *Transactions of the Association for Computational Linguistics* 4, 521-535.

Nalisnick, E. T. and H. S. Baird. 2013. Extracting sentiment networks from Shakespeare's plays. *2013 12th International Conference on Document Analysis and Recognition,* 758-762.

Nielsen, F. Å. 2011. A new ANEW: Evaluation of a wordlist for sentiment analysis in microblogs. *arXiv preprintarXiv:1103.2903.*

McCoy, R. T., R. Frank and T. Linzen. 2018. Revisiting the poverty of the stimulus: Hierarchical generalization without a hierarchical bias in recurrent neural networks. *arXiv preprint arXiv:1802.09091.*

Severyn, A. and A. Moschitti. 2015, August. Twitter sentiment analysis with deep convolutional neural networks. *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval,* 959-962.

예시 언어(Examples in): 영어(English)
적용가능 언어(Applicable Language): 영어(English)
적용가능 수준(Applicable Level): 성인(Tertiary)

서혜진(Seo, Hye-Jin), 대학원생(Graduate Student)
동국대학교
영어영문학부
04620 서울특별시 중구 필동로 1길 30(30 Pildong-ro 1 gil, Jung-gu, Seoul 04620)
Tel: 02) 2260-8705
E-mail: seohj0951@gmail.com

신정아(Shin, Jeong-Ah), 교수(Professor)
동국대학교(Dongguk University)
영어영문학부(Division of English Language and Literature)
04620 서울특별시 중구 필동로 1길 30(30 Pildong-ro 1 gil, Jung-gu, Seoul 04620)
Tel: 02) 2260-3167
E-mail: jashin@dgu.ac.kr

논문 접수(Received): 2020년 2월 10일 (February 10, 2020)
논문 수정(Revised): 2020년 3월 10일 (March 10, 2020)
게재 확정(Accepted): 2020년 3월 20일 (March 20, 2020)