

케라스로 배우는 자연어처리

허진경 지음

케라스로 배우는 자연어처리

발 행 | 2021년 12월 30일

저 자 | 허진경

펴낸이 | 한건희

펴낸곳 | 주식회사 부크크

출판사등록 | 2014.07.15.(제2014-16호)

주 소 | 서울특별시 금천구 가산디지털1로 119 SK트윈타워 A동 305호

전 화 | 1670-8316

이메일 | info@bookk.co.kr

ISBN | 979-11-372-0000-0

www.bookk.co.kr

© 허진경 2021

본 책은 저작자의 지적 재산으로서 무단 전재와 복제를 금합니다.

CONTENT

1장. 인공지능과 자연어처리	1
1절. 인공지능	2
1.1. 인공지능의 시작	2
1.2. 인공지능의 역사	3
2절. 자연어처리	4
2.1. 자연어와 자연어처리 분야	4
2.2. 자연어처리 역사와 영역	5
2.3. 자연어처리 활용 사례	6
1) 인공지능 번역기	6
2) 인공지능 챗봇	6
3) 인공지능 스피커	6
4) 인공지능 리걸 테크(Legal Tech)	6
5) 인공지능 저널리즘(Journalism)	7
6) 인공지능 컨택서비스(Contact Service)	7
3절. 자연어처리를 위한 텍스트 전처리	8
3.1. 텍스트 전처리	8
3.2. 형태소 분석	9
1) NLTK	9
2) KoNLPY	9
3.3. 임베딩	10
1) 워드 임베딩	10
2) 워드 임베딩의 종류	10
3) 언어 모델	11
4절. 워드 임베딩	12

4.1. TF	12
4.2. TDM	14
4.3. TF-IDF	15
4.4. Word2Vec	17
1) 윈도우와 슬라이딩 윈도우	18
2) CBOW	19
3) Skip-gram	20
4) CBOW와 Skip-Gram의 차이점	21
5) gensim 패키지	22
6) Pre-trained 모델	23
2장. 순환신경망	25
1절. 순환신경망	26
1.1. 순환신경망 개요	26
1.2. RNN의 종류	27
1.3. Seq2Seq	28
1.4. Bidirectional RNN	30
2절. LSTM과 GRU	31
2.1. LSTM	31
2.2. GRU	33
2.3. GRU Stacking	35
3절. 어텐션	36
3.1. Seq2Seq의 문제점	36
3.2. 어텐션	37
3.3. 어텐션 적용 과정	39
1) 어텐션 가중치 계산	39
2) 어텐션 분포 계산	39
3) 어텐션 벡터 계산	40

4) 연결	41
4절. 트랜스포머	42
4.1. 트랜스포머 개요	42
1) 트랜스포머란?	42
2) 트랜스포머 구조	43
3) 트랜스포머의 인코더와 디코더 구조	43
4) 트랜스포머의 임베딩과 Positional 인코딩	44
5) 트랜스포머의 Self-Attention	44
4.2. Multi-head Self Attention	46
1) Multi-head Self Attention	46
2) Multi-head Attention Value Matrix	47
3) Position-wise FFN	48
4) 잔차 연결	48
5) 정규화	49
5절. 인공지능 언어 모델	50
5.1. 언어 모델 개요	50
1) 언어 모델	50
2) 최신 언어 모델	51
3) 언어 모델의 분류	51
5.2. 언어 모델 성능 평가	52
1) 스쿼드(SQuAD)	52
2) KorQuAD 1.0	52
3) KorQuAD 2.0	52
5.3. 언어 모델 전이 학습	53
1) 전이 학습이란?	53
2) Pre-trained 모델의 Fine-tuning	54
3) 전이 학습 전략의 선택	55
5.4. 최신 인공지능 언어 모델	55
1) BERT	55
2) RoBERTa	56
3) ALBERT	57

4) T5	57
5) KoBERT	58
3장. 자연어처리 실습	59
1절. 개요	60
1.1. 학습 환경	60
1) 텐서플로우와 케라스	60
2) 구글 코랩	61
3) 구글 캐글	61
1.2. 자연어처리의 몇 가지 예	62
1) 자동 이미지 캡션처리	62
2) 구글의 이미지 캡션처리	63
3) 페이스북의 가짜 뉴스 판별	64
4) 오피니언 마이닝(Opinion Mining)	64
2절. RNN으로 영화평 분류하기	66
2.1. IMDB 영화평 데이터셋	66
2.2. 구글 코랩에서 영화평 분석하기	67
3절. 로이터 기사 분류하기	83
3.1. 로이터 데이터셋	83
3.2. RNN으로 로이터 기사 자동 분류하기	85
4절. 영-한 번역기	96
4.1. 번역기	96
1) 번역기의 한계	96
4.2. 영-한 번역기	96
1) 사전 만들기	97
2) 단어 변환	98
4.3. 코드 구현	100

5절. BBC 기사 분류하기	113
5.1. BBC 기사 데이터셋	113
5.2. 데이터 전처리하기	115
5.3. RNN 모델	121
6절. 음성 인식하기	126
6.1. librosa 패키지	127
1) 샘플링 비율	128
6.2. 모델의 구조	130
1) 모델 구조	130
6.3. RNN으로 음성 인식하기 구현 코드	131
1) 데이터 처리	131
2) RNN 모델링 코드	135

주요 학습 내용

- 순환신경망 이론
- 영어-한글 번역하기
- Word Embedding
- RNN, LSTM, GRU
- IMDB(Internet Movie DataBase) 영화평 구분하기
- 로이터(Reuter) 기사 분류하기
- 영-한 번역기
- BBC 기사 분류하기
- 음성 인식하기
- 워드 임베딩 개념과 토픽 기반 예측
- 단어단위 언어모델, 문장단위 언어모델의 차이점 및 종류
- 인공지능 자연어처리를 위한 다양한 언어 모델의 종류와 특징

인공지능 비서, 챗봇 및 실시간 통번역 등에 활용되는 범용 인공지능 구현에 있어 가장 해결하기 어려운 문제인 인공지능 딥러닝 자연어처리에 대해 최근 도출된 다양한 성과를 바탕으로 실제 프로젝트 사례 및 이론을 살펴봄으로써 기본 학습부터 심화학습, 실무 적용 능력, AI 텍스트 분석 분야의 트렌드 및 최신 기술에 대한 학습을 제공합니다.

학습 목적 : 실제 산업에서 가장 많이 활용되고 있는 분야 중 하나인 인공지능 자연어처리의 해결하기 어려운 문제와 다양한 팁을 제공하여 실무 적용 능력을 기릅니다.

이 책은 아래의 수정 이력이 있습니다.

- 초안 작성일 : 2021년 12월 30일

“

1장. 인공지능과 자연어처리

”

1절. 인공지능

1.1. 인공지능의 시작

인공지능(Artificial Intelligence)은 1950년대 컴퓨터의 등장과 함께 탄생한 개념입니다. 인지, 학습 등 인간의 지적 능력의 부분 또는 전체를 컴퓨터를 이용해 구현하는 기술 분야입니다.

미국의 전산학자이자 인지과학자인 존 매카시(John McCarthy)가 1956년 다트머스 학회(Dartmouth Conference)에서 인공지능이라는 용어를 처음으로 사용했습니다.

영국의 암호학자이자 컴퓨터 과학의 선구자인 앨런 튜링(Alan Turing)은 『Computing Machinery and Intelligence』(1950) 논문을 통해 컴퓨터가 인간처럼 사고하는 것이 가능하다고 했습니다. 그래서 만들어진 말이 튜링 테스트(Turing Test)입니다. 이것은 기계의 사고력을 증명하기 위한 것으로 이미테이션 게임(Imitation Game)이라고도 불립니다. 훗날 인공지능 연구의 초석이 됩니다.

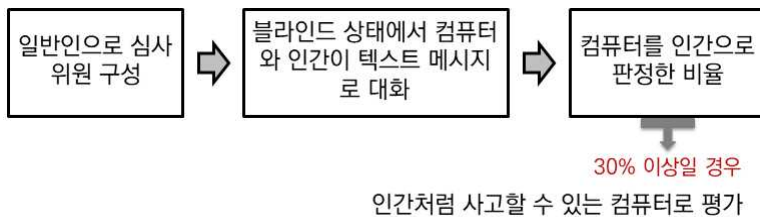


그림 1. 튜링 테스트

튜링 테스트에서 컴퓨터를 인간으로 판정하는 비율이 30% 이상일 경우 그 컴퓨터는 인간처럼 사고할 수 있는 컴퓨터로 평가됩니다.

유진 구스트만(Eugene Goostman)은 최초로 튜링 테스트를 통과한 컴퓨터 프로그램입니다. 영국의 레딩대학교가 개발했으며, 13세 우크라이나 소년으로 설정되어 있습니다. 이 컴퓨터의 튜링 테스트에서 25명의 심사위원 중 33%가 인간이라고 판단했습니다. 그런데 문맥과 맞지 않게 대답한 경우가 많아서 종합적인 사고능력이 있다고 판단하기에는 무리가 있다고 평가되어 튜링 테스트의 실효성 논란이 있기도 했습니다.

1.2. 인공지능의 역사

인공지능의 역사는 1950년대부터 시작합니다. 1956년 존 메카시가 처음 인공지능 용어를 사용하고 인공지능의 개념이 정립된 이후 인공지능은 암흑기와 증흥기를 거칩니다.

딥러닝에 관한 연구가 활발해 지면서 2010년대에 이르러서야 인공지능의 황금기가 도래합니다. 2011년 IBM의 왓슨은 제퍼디 퀴즈쇼에서 역대 최강 챔피언들을 상대로 압도적인 승리를 거둡니다.

알파고 대 이세돌 대결로 알려진 딥마인드 챌린지 매치(Google Deepmind Challenge match)는 2016년 3월 9일부터 15일까지, 하루 한 차례의 대국으로 총 5회에 걸쳐 서울의 포-시즌스(Four Seasons) 호텔에서 진행된 이세돌과 알파고(AlphaGo) 간의 바둑 대결입니다. 최고의 바둑 인공지능 프로그램과 바둑의 최고 중 최고 인간 실력자의 대결로 주목을 받았으며, 최종 결과는 알파고가 4승 1패로 이세돌에게 승리하였다. 이로써 일반인들의 인공지능에 관한 관심이 폭발적으로 증가하게 되었습니다.

2절. 자연어처리

2.1. 자연어와 자연어처리 분야

자연어는 인간이 상대방과 의사소통을 위해서 사용하는 언어입니다. 한국어, 영어, 중국어 등이 자연어에 해당합니다. 자연어처리(NLP; Natural Language Processing)는 사람의 자연어를 컴퓨터가 이해하고 처리할 수 있도록 하는 인공지능의 한 분야입니다.

자연어처리는 자연어 생성(NLG; Natural Language Generation) 분야와 자연어 이해(NLU; Natural Language Understanding) 분야로 나뉩니다. 자연어 생성은 컴퓨터 스스로 자연어를 만들어내는 것을 의미하며, 자동완성 및 생성형 언어 모델 등에 활용합니다. 자연어 이해는 컴퓨터가 사람의 자연어를 이해하는 것을 의미하며, 감정분석 및 기계 독해 등에 합니다. 자연어 생성과 자연어 이해는 챗봇(Chatbot) 엔진이 핵심 구성요소이기도 합니다.

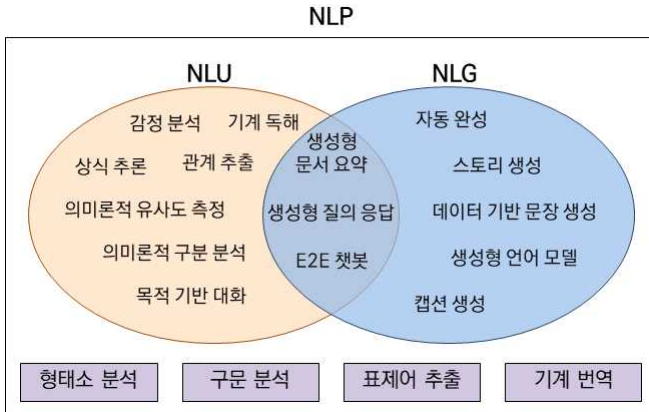


그림 2. 자연어처리 분야

2.2. 자연어처리 역사와 영역

자연어처리의 역사는 1940년대까지 올라갑니다. 1940년대에는 실용적인 관점에서 이중어 사전(Bilingual Dictionary)을 이용한 기계 번역을 시도했으며, 1950년대에는 언어를 작은 의미소 단위로 분석, 형태소 추출 후 문장을 재구성하는 구조주의 방법론을 적용합니다.

1960년대에는 SAD-SAM, BASEBALL, STUDENT, ELIZA 등 다양한 NLP 시스템이 개발되기 시작했고, 1970년대에는 구문분석과 의미 분석, 어휘 사전을 활용하여 특정 분야에 특화된 NLP 연구가 활발하게 진행되었습니다. 이후 1980년대에는 변형을 최소화한 통합기반 문법(Unification-based Grammar)이 나왔으며, 1990년대에는 말뭉치를 대규모로 구축하여 통계적으로 처리하는 NLP 방식이 발전되었습니다. 2000년대 이후에는 NLP 분야에 기계학습(Machine Learning)과 딥러닝(Deep Learning)을 본격적으로 적용하기 시작했습니다.

지금의 자연어처리는 머신러닝과 딥러닝 일부에도 포함되어 있으면서 AI 내에서 별도의 영역에도 해당합니다.

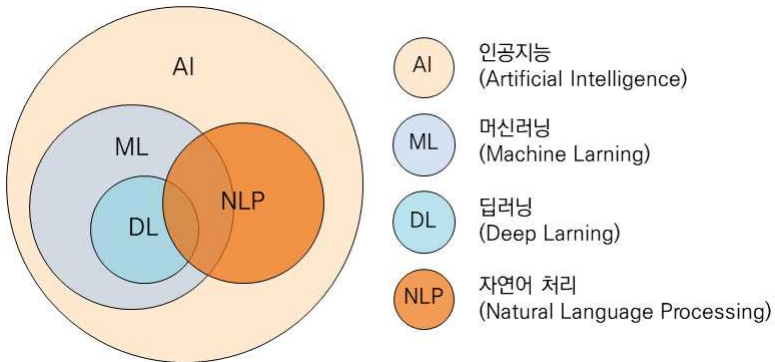


그림 3. AI vs. ML vs. DL vs. NLP

2.3. 자연어처리 활용 사례

1) 인공지능 번역기

딥러닝을 이용해 국가 간 언어를 번역하는 서비스입니다. 기존 번역 서비스보다 정확도가 한층 높아졌으며, 전후 문맥을 파악하여 자연스러운 번역이 가능해집니다. 최근에는 딥러닝 음성인식 기술과 융합하여 통역 서비스에 활용하기도 합니다.

2) 인공지능 챗봇

챗봇(Chatbot)은 채팅(Chatting)과 로봇(Robot)이 합성어로 인공지능 기반의 커뮤니케이션 소프트웨어입니다. 사람과의 문자 대화를 통해 질문에 알맞은 답변이나 각종 연관 정보를 제공하며, 다양한 비즈니스 도메인 분야에 활용됩니다.

3) 인공지능 스피커

음성인식 기능과 인공지능 자연어처리 기술 등을 활용한 스피커를 의미합니다. 인공지능 스피커는 사용자의 음성 명령을 스피커가 서버로 전송하고, 인공지능 플랫폼이 의미분석 및 서비스를 제공합니다.

4) 인공지능 리걸 테크(Legal Tech)

인공지능 자연어처리, 빅데이터 기술 등을 활용한 법률서비스를 의미합니다. 기존에 판례나 법령, 변호사들의 자문 빅데이터를 활용합니다. 이용자가 기본적인 정보만 입력하면 자동으로 법률 문서가 생성되고 법률 상담 서비스를 제공합니다.

5) 인공지능 저널리즘(Journalism)

인공지능 기술을 활용해 기사를 작성하는 로봇 저널리즘입니다. 머신러닝, 딥러닝 등을 활용한 자연어처리 기술을 활용합니다. 방대한 빅데이터에서 중요한 정보를 편집해서 기사화하는 데이터 저널리즘 등에 활용합니다. 기사 및 정보의 사실 여부를 평가하여 가짜 뉴스 판별에도 활용합니다.

6) 인공지능 컨택서비스(Contact Service)

고도화된 AI 음성 로봇을 활용한 고객 상담센터를 제공할 수 있습니다. 고객들의 상담 대기시간을 획기적으로 줄일 수 있으며, 고객의 문의 사항에 대한 신속한 답변과 안내 사항 등 각종 정보를 제공합니다.

3절. 자연어처리를 위한 텍스트 전처리

3.1. 텍스트 전처리

자연어처리에 사용되는 데이터를 말뭉치(코퍼스, Corpus)라고 부릅니다. 말뭉치는 여러 문장으로 구성된 문서의 집합입니다. 텍스트 전처리 단계는 정제, 불용어 제거, 어간 추출, 토큰화 및 문서 표현 등의 작업을 수행해야 합니다. 텍스트 전처리를 통해 자연어처리에 적합한 말뭉치(코퍼스, Corpus)를 만듭니다.

정제(Cleaning)는 특수문자 등과 같은 불필요한 노이즈 텍스트를 제거하거나 대소문자를 통일시킵니다.

- 특수문자 예시: '!"#\$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
- 대소문자 통일 예시: korea, Korea → KOREA

불용어 제거(Stop word Elimination)는 전치사, 관사 등 문장이나 문서의 특징을 표현하는데 불필요한 단어를 제거합니다.

- 어간 추출 : 단어의 기본 형태를 추출하는 단계
- 어간 추출 예시: stem, stems, stemmed, stemmer → stem

토큰화(Tokenizer)는 말뭉치에서 분리 자(Separator)를 포함하지 않는 연속적인 문자열 단위로 분리합니다. 파이썬에서 영문 토큰화는 nltk를 사용하며, 한글 토큰화는 konlpy를 사용할 수 있습니다.

워드 임베딩(Word Embedding, 문서 표현)은 주어진 문서나 문장을 하나의 벡터로 표현합니다. 단어들을 인덱싱(Indexing)하고 주어진 문서에 존재하는 단어의 빈도수를 사용하여 문서를 표현합니다. 워드 임베딩은 One-hot 인코딩, TF-IDF, Word2Vec 등이 있습니다.

3.2. 형태소 분석

1) NLTK

NLTK(Natural Language Toolkit, <https://www.nltk.org/>)는 인간의 언어 데이터를 파이썬 언어로 분석할 수 있도록 만들어진 패키지입니다. 이 패키지는 분류, 토큰화, 형태소 분석, 품사 태깅, 구문 분석 및 의미론적 추론을 위한 텍스트 처리 라이브러리와 WordNet과 같은 50개가 넘는 말뭉치를 제공합니다.

2) KoNLPY

KoNLPy(코엔엘파이)는 한국어 정보처리를 위한 파이썬 패키지입니다. KoNLPy는 공개 소스 소프트웨어이며 [GPL v3 이상] 라이선스로 배포됩니다. 공식 사이트는 다음과 같습니다.

- <http://konlpy.org/> 또는 <https://github.com/konlpy/konlpy>

다음은 KoNLPy 패키지가 지원하는 한국어 형태소 분석기입니다.

- Hannanum(한나눔)
 - Kkma(꼬꼬마)
 - Komoran(코모란)
 - Mecab(미캡)
 - Okt(Open Korean Text, Twitter가 0.5.0부터 Okt로 바뀜)
- <https://konlpy.org/ko/latest/morph/>에서 KoNLPy 내부 품사 태깅 클래스들의 성능 비교를 볼 수 있습니다.

형태소 분석에 대한 더 자세한 내용은 [파이썬 데이터 전처리 및 탐색 라이브러리, 출판사 부크크, 저자 허진경] 책을 참고하세요.

3.3. 임베딩

임베딩(Embedding)은 범주형 자료를 연속형 벡터 형태로 변환시키는 것입니다.

1) 워드 임베딩

자연어처리를 하려면 인간이 이해하고 사용하는 언어를 컴퓨터가 이해할 수 있도록 숫자 형태로 변환해야 합니다. 그래서 워드 임베딩(Word Embedding)은 문자열을 숫자로 변환하여 벡터(Vector) 공간에 표현하는 방법입니다.

워드 임베딩을 통해 단어와 단어, 문장(문서)과 문장(문서) 간의 유사도를 계산할 수 있습니다. 그리고 벡터 간 연산을 통해 의미적 관계를 도출할 수 있습니다.

2) 워드 임베딩의 종류

워드 임베딩은 빈도(Frequency)기반, 토픽(Topic) 기반, 예측(Prediction) 기반으로 나뉩니다.

빈도(Frequency) 기반은 문서에 등장하는 각 단어의 빈도를 계산해서 행렬로 표현하거나 가중치를 부여합니다. 즉, 빈도를 이용해서 단어의 중요도나 문서 간 유사도를 측정하려는 방법이며, TF(Term Frequency), DTM(Document Term Matrix), TF-IDF(Term Frequency-Inverse Document Frequency)가 있습니다.

토픽(Topic) 기반은 주어진 문서에 잠재된 주제(Latent Topic)를 추론(Inference)하기 위한 임베딩입니다. 토픽 기반으로 LSA(Latent

Semantic Analysis, 잠재 의미분석) 인코딩이 있습니다.

예측(Prediction) 기반은 주어진 문장이나 단어의 다음 단어 예측, 주변 단어에 대한 예측, 마스킹(Masking) 된 단어의 예측 등을 위한 임베딩입니다. Word2Vec, FastText, BERT(Bidirectional Encoder Representations from Transformers), ELMo(Embeddings from Language Model), GPT(Generative Pre-trained Transformer) 등이 있습니다.

3) 언어 모델

워드 임베딩은 단어를 기반으로 임베딩을 수행하는 것입니다. 이것은 문맥을 고려하지 않은 상태에서 임베딩 합니다. 워드 임베딩은 서로 다른 문맥의 동음이의어가 같게 임베딩 되는 단점이 있습니다.

문맥을 고려하여 문장(Sentence)을 기반으로 임베딩을 수행하는 것을 문장 기반 워드 임베딩이라고 하며 다른 말로 언어 모델(Language Model)이라고 부릅니다.

4절. 워드 임베딩

워드 임베딩(Word Embedding)은 단어를 숫자에 대응시킵니다. 워드 임베딩을 해 놓으면 의미론적, 구문론적 컨텍스트 또는 단어를 인지하는 데 도움이 되며 기사, 블로그 등에서 다른 단어와 얼마나 유사하거나 유사하지 않은지 이해하는 데 도움을 줍니다.

인기 있는 워드 임베딩은 다음과 같습니다.

- 이진 인코딩(Binary Encoding) 또는 원-핫 인코딩
- TF(Term Frequency)
- TDM(Term Document Matrix)
- TF-IDF(Term Frequency-Inverse Document Frequency)
- LSA(Latent Semantic Analysis)
- Word2Vec

4.1. TF

TF는 Term Frequency의 약어로, 문서에서 용어가 나타나는 총 회수로 정의됩니다.

예를 들면 다음 두 문장이 있다고 가정하겠습니다.

- John likes to watch movies. Mary likes movies too.
- Mary also likes to watch football games.

다음은 위 두 문장을 토큰화하여 단어 목록으로 만들고 빈도수를 측정하여 각 단어와 고유 단어별 빈도수를 갖는 딕셔너리를 만드는 코드입니다.

```
1 text = "John likes to watch movies. Mary likes movies too.\n2 Mary also likes to watch football games."
```

문서에서 구두점을 제외하고 공백으로 분리합니다.

```
1 words = text.replace('.', '').split()\n2 print(words)\n\n['John', 'likes', 'to', 'watch', 'movies', 'Mary', 'likes',\n'movies', 'too.', 'Mary', 'also', 'likes', 'to', 'watch',\n'football', 'games']
```

다음은 리스트에서 유일한 값의 개수를 셉니다.

```
1 import numpy as np\n2 word_count = np.unique(words, return_counts=True)\n3 print(word_count)\n\n(array(['John', 'Mary', 'also', 'football', 'games', 'likes',\n'movies', 'to', 'too.', 'watch'], dtype='<U8'), array([1, 2, 1, 1,\n1, 3, 2, 2, 1, 2]))
```

단어-개수 딕셔너리를 만듭니다.

```
1 word_to_cnt = {}\n2 for word, cnt in zip(*word_count):\n3     word_to_cnt[word] = cnt\n4 print(word_to_cnt)\n\n{'John': 1, 'Mary': 2, 'also': 1, 'football': 1, 'games': 1,\n'likes': 3, 'movies': 2, 'to': 2, 'too.': 1, 'watch': 2}
```

만일 여러 개 문서가 있다면 문서별로 빈도수를 계산해서 2차원 행렬로 만들어 놓을 수 있습니다.

4.2. TDM

TDM(Term Document Matrix)은 문서별로 단어의 빈도수를 계산해서 행렬로 만들어 놓은 것입니다. 만일 앞의 예제 텍스트가 두 개의 문장이라면 문장마다 단어의 빈도수를 계산해서 표현할 수 있습니다.

```
1 corpus = [  
2     "John likes to watch movies. Mary likes movies too.",  
3     "Mary also likes to watch football games."  
4 ]
```

싸이킷런을 이용하면 TDM 문서를 쉽게 만들 수 있습니다.

```
1 from sklearn.feature_extraction.text import CountVectorizer  
2 vector = CountVectorizer()  
3 tdm_array = vector.fit_transform(corpus).toarray()  
4 tf_dic = vector.vocabulary_  
5 print(tdm_array)  
6 print(tf_dic)
```

```
[[0 0 0 1 2 1 2 1 1 1]  
 [1 1 1 0 1 1 0 1 0 1]]  
{'john': 3, 'likes': 4, 'to': 7, 'watch': 9, 'movies': 6, 'mary':  
5, 'too': 8, 'also': 0, 'football': 1, 'games': 2}
```

다음은 TDM 행렬을 데이터프레임으로 만듭니다.

```
1 import pandas as pd  
2 tf_dic_sorted = dict(sorted(tf_dic.items(),  
                             key=lambda item: item[1]))  
3 tdm = pd.DataFrame(tdm_array, columns=tf_dic_sorted.keys())  
4 print(tdm)
```

	also	football	games	john	likes	mary	movies	to	too	watch
0	0	0	0	1	2	1	2	1	1	1
1	1	1	1	1	0	1	1	0	1	1

4.3. TF-IDF

TF-IDF(Term Frequency-Inverse Document Frequency)는 TF와 IDF의 두 부분으로 나눌 수 있습니다.

TF는 문서에서 해당 용어가 발생한 횟수를 총 용어 수로 나누어 계산합니다. 방법은 상당히 직관적입니다. 문서에서 용어가 더 많이 나올수록 이 용어는 문서에 대해 더 많은 의미가 있습니다. 그러나 문서의 내용에 따라 항상 그렇지는 않습니다. 우리가 계산하는 용어 빈도에서 "a", "of", "the"와 같은 용어가 가장 큰 값을 가짐을 알 수 있습니다. "the"라는 단어가 너무 일반적이어서 모든 기사에 이 단어가 포함되어 있으므로 이러한 단어는 분석에 큰 도움이 되지 않습니다. 그리고 이러한 문법적 목적어를 갖는 것은 결국 우리의 분석 결과에 영향을 미칠 가능성이 매우 큼니다. 이 문제를 처리하는 한 가지 방법은 단순히 제거하는 것입니다. "the" 등의 단어들은 불용어 처리해서 문제를 해결할 수 있습니다. 그러면 실제 의미를 지닌 단어들만 남을 것입니다. 그런데 그것만으로는 모든 문제가 해결되지 않습니다.

대한민국에서 고양이에 대한 뉴스를 분석하고 빈도라는 용어를 계산했다고 가정합니다. 이 기사 중 하나에서 "대한민국", "동물", "음식", "고양이"와 같은 단어가 같은 빈도를 갖는다고 해서 이 단어들이 기사에 대해 같은 양의 중요성이 있다는 것을 의미하진 않을 것입니다. 이 문제를 어떻게 해결하기 위해 IDF를 사용합니다. 이렇게 TF는 때때로 잘못된 용어를 강조하는 경향이 있으므로 IDF는 용어 가중치의 균형을 맞추기 위해 도입되었습니다.

IDF는 전체 문서에서 용어가 얼마나 자주 발생하는지 정의합니다. 전체 문서 세트에서 발생하는 용어의 가중치 균형을 맞추는 데 사용됩니다. 즉, IDF는 자주 발생하는 항의 가중치를 줄이고 드물게 발생하는 항의 가중치를 증가시킵니다. 다음은 TF-IDF를 계산하기 위한 공

식입니다. w 는 단어를 의미하며 IDF 계산 시 분모에 1을 더하는 이유는 0으로 나뉘는 것을 피하기 위함입니다.

$$TF(w) = \text{문서에서 } w \text{의 빈도} / \text{총 단어 수}$$

$$IDF(w) = \log(\text{용어를 포함하는 문서 수}) / (\text{총 문서 수} + 1)$$

IDF는 로그를 취하기 전에 해당 용어가 포함된 문서 수를 전체 문서 수로 나누어 계산합니다. 용어가 전체 문서 세트에 자주 나타나는 경우 IDF 결과는 0에 가까울 것입니다. 그렇지 않으면 양의 무한대로 증가합니다. 최종 TF-IDF 점수를 얻으려면 TF와 IDF의 결과를 곱해야 합니다.

$$TF-IDF(w) = TF(w) * IDF(w)$$

TF-IDF 점수가 클수록 해당 용어가 문서에서 더 관련성이 높습니다. 그 결과 TF-IDF는 기사에서 단어가 등장한 횟수에 비례하고, 이 단어가 기사 전체 영역에서 등장한 횟수에 반비례함을 알 수 있습니다.

다음 코드는 TF-IDF를 이용한 TDM을 만듭니다.

```
1 from sklearn.feature_extraction.text import TfidfVectorizer
2
3 tfidf_vec = TfidfVectorizer()
4 tfidf_array = tfidf_vec.fit_transform(corpus).toarray()
5 tfidf_dic = tfidf_vec.vocabulary_
6 tfidf_dic_sorted = dict(sorted(tfidf_dic.items(),
                                key=lambda item: item[1]))
7 tfidf_tdm = pd.DataFrame(tfidf_array,
                           columns=tfidf_dic.keys())
8 print(tfidf_tdm)
```

	also	football	games	john	likes	mary	movies	to	too	watch
0	0.000000	0.000000	0.000000	0.323699	0.460629	0.230315	0.647398	0.230315	0.323699	0.230315
1	0.446101	0.446101	0.446101	0.000000	0.317404	0.317404	0.000000	0.317404	0.000000	0.317404

4.4. Word2Vec

Word2Vec는 단어를 벡터 형식으로 변환하는 도구입니다.

Word2vec은 워드 임베딩 모델을 생성하는 데 사용되는 얇은 2계층 신경망입니다. 많은 어휘 기반 데이터 세트를 입력으로 수집하고 사전의 각 단어가 고유한 벡터에 대응되도록 벡터 공간을 출력하는 방식으로 작동합니다. 이를 통해 단어 간의 관계를 나타낼 수 있습니다.

Word2Vec은 CBOW(Continuous Bag of Words)와 Skip-Gram의 두 가지 모델이 있습니다. CBOW는 주변 단어를 임베딩하여 중심 단어를 예측하고, Skip-gram은 중심 단어를 임베딩하여 주변 단어를 예측합니다.

CBOW와 Skip-gram은 서로 대칭적 구조입니다.

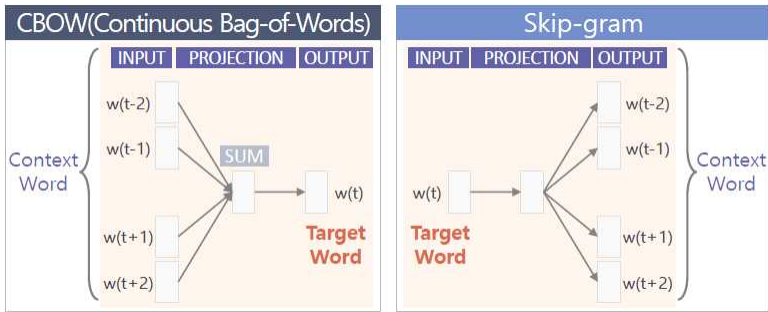


그림 4. CBOW와 Skip-gram 구조

Word2Vec에서 CBOW와 Skip-gram을 설명하기 전에 윈도우(Window)의 개념과 슬라이딩 윈도우(Sliding Window)의 개념을 이해해야 합니다.

1) 윈도우와 슬라이딩 윈도우

윈도우(Window)는 중심 단어(Target Word)를 기준으로 해서 참조하고자 하는 주변 단어(Context Word)의 범위를 말합니다. 만일 윈도우 크기가 2인 경우 “quality is more important than quantity”에서 중심 단어가 ‘quality’인 경우 주변 단어는 ‘is’와 ‘more’이며, 중심 단어가 ‘more’인 경우 앞의 두 단어 ‘quality’와 ‘is’, 뒤의 두 단어 ‘important’와 ‘than’ 이렇게 4개 단어가 주변 단어가 됩니다.

슬라이딩 윈도우(Sliding Window)는 전체 학습 문장에 대해 윈도우를 이동하며 학습하기 위해 중심 단어와 주변 단어 데이터셋을 추출하는 과정입니다.

다음 그림은 “quality is more important than quantity” 문장에서 윈도우 크기가 2인 경우의 슬라이딩 윈도우를 보여줍니다.



그림 5. 슬라이딩 윈도우

다음은 각 단어를 원-핫 인코딩으로 표현한 후 슬라이딩 윈도우를 통해 추출된 중심 단어와 주변 단어 데이터셋입니다.

Sliding	Target Word	Context Word
1	[1, 0, 0, 0, 0, 0]	[0, 1, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0]
2	[0, 1, 0, 0, 0, 0]	[1, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0], [0, 0, 0, 1, 0, 0]
3	[0, 0, 1, 0, 0, 0]	[1, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 1, 0]
4	[0, 0, 0, 1, 0, 0]	[0, 1, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 1]
5	[0, 0, 0, 0, 1, 0]	[0, 0, 1, 0, 0, 0], [0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 1]
6	[1, 0, 0, 0, 0, 1]	[0, 0, 0, 0, 1, 0], [0, 0, 0, 1, 0, 0]

그림 6. 슬라이딩 윈도우와 주변 단어 데이터셋

2) CBOW

CBOW(Continuous Bag-of-Words)는 주변 단어(Context Word)를 임베딩하여 중심 단어(Target Word)를 예측하기 위한 Word2Vec 방법입니다.

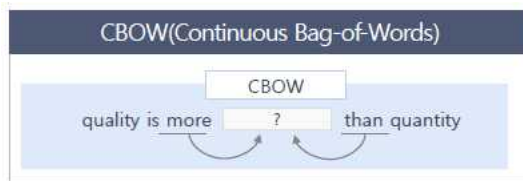


그림 7. CBOW

CBOW는 소규모 데이터베이스 작업에 적합한 모델이며 RAM 메모리 요구사항도 필요하지 않습니다. CBOW에서는 문맥에 따라 단어를 예측합니다. 전체 어휘는 다음 작업을 진행하기 전에 "Bag of Words"

모델을 만드는 데 사용됩니다. Bag of Words 모델은 문법을 무시하고 단어를 단순히 표현한 것입니다.

CBOW의 중심 단어를 예측할 때 윈도우 크기를 1로 하고 입력으로 'more'와 'than'을 넣는다면 'important'가 출력됩니다.

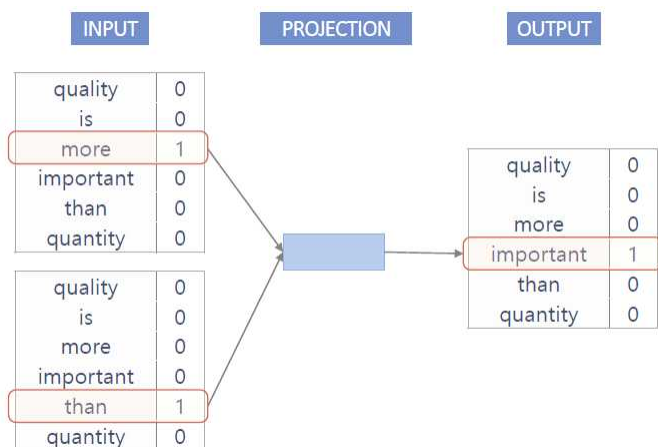


그림 8. CBOW 윈도우 1일 경우 입력과 출력

3) Skip-gram

Skip-gram은 중심 단어(Target Word)를 임베딩 하여 주변 단어(Context Word)를 예측하기 위한 Word2Vec 방법입니다.



그림 9. Skip-gram

Skip-gram에서 윈도우 크기를 1로 하고, 입력을 important를 넣으면 'more'와 'than'이 출력됩니다.

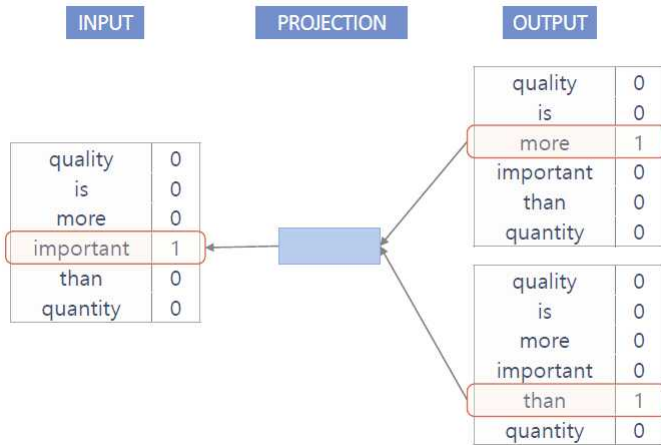


그림 10. Skip-gram 윈도우 1일 경우 입력과 출력

4) CBOW와 Skip-Gram의 차이점

두 모델은 대칭을 보여주지만, 아키텍처와 성능 면에서 다릅니다.

Skip-Gram 모델은 단어의 맥락적 유사성에 의존하여 특정 단어를 둘러싼 단어를 예측합니다. 반면에 CBOW 모델은 주변 단어를 사용하여 중심 단어를 예측합니다.

Skip-Gram 모델은 빈도가 낮은 단어에 대해 더 정확합니다. 더 큰 데이터베이스에 적합하며 작동하려면 더 많은 RAM이 필요합니다. CBOW 모델은 더 빠르고 더 적은 RAM이 필요하며 더 작은 데이터베이스에 적합하지만, 자주 사용되지 않는 단어의 처리를 보장하지 않습니다.

5) gensim 패키지

파이썬에서 CBOW, Skip-gram 등의 Word2Vec 적용을 위한 라이브러리입니다.

gensim.models.Word2Vec(data, sg, size, window, min_count)

- ▶ data : 리스트 형태의 데이터입니다.
- ▶ sg : 0이면 CBOW, 1이면 Skip-gram 방식을 사용합니다.
- ▶ size : 벡터의 크기를 지정합니다.
- ▶ window : 윈도우의 크기입니다. 3이면 앞/뒤 3단어를 포함합니다.
- ▶ min_count : 사용할 단어의 최소 빈도입니다. 3이면 3회 이하 단어는 무시합니다.

다음 코드는 앞에서 사용한 corpus 데이터를 이용해서 Word2Vec 모델을 이용하여 특정 단어와 가장 유사한 단어를 찾거나 두 단어의 유사도를 확인하는 예입니다.

```
1 corpus = ["John likes to watch movies. Mary likes movies too",
2           "Mary also likes to watch football games."]
3
4 word_list = []
5 for word in corpus:
6     word_list.append(word.replace('.', '').split())
7
8 from gensim.models import Word2Vec
9 model = Word2Vec(word_list, sg=0, size=100,
10                  window=3, min_count=1)
11
12 print(model.wv.most_similar('likes'))
13 print(model.wv.similarity('movies', 'games'))
```

```
[('too', 0.07887491583824158), ('games', 0.06314955651760101),
 ('watch', 0.019802890717983246), ('also', 0.004840634763240814),
 ('to', -0.06213974952697754), ('football', -0.1027420312166214),
```

```
('John', -0.12695640325546265), ('movies', -0.1419140100479126),  
('Mary', -0.23345059156417847)]  
0.21119785
```

6) Pre-trained 모델

대용량 코퍼스 데이터를 이용해서 사전에 학습된 모델을 Pre-trained 모델이라고 합니다. 한국어의 미리 학습된 Word2Vec 모델을 사용하여 유사도를 분석할 수 있습니다.

<https://github.com/Kyubyong/wordvectors>에서 화면 아래쪽으로 스크롤 한 후 Pre-trained models 목록에서 Korean (w) 항목을 클릭한 후 ko.bin 파일을 내려받으세요.

Word2Vec.load() 함수를 이용해서 모델을 불러올 수 있습니다. 이후 특정 단어와 유사한 단어들을 확인할 수 있습니다. 예제에서는 '인공지능' 단어와 유사한 단어들을 확인해 봅니다.

```
1 model = Word2Vec.load('ko.bin')  
2 print (model.wv.most_similar('인공지능'))  
[('컴퓨팅', 0.6520194411277771), ('가상현실',  
0.6393702030181885), ('심리학', 0.63037109375), ('모델링',  
0.625065267086029), ('신경망', 0.6200424432754517), ('로봇',  
0.6109743118286133), ('시뮬레이션', 0.6101070642471313), ('지능',  
0.6092983484268188), ('기술', 0.6087720990180969), ('기술인',  
0.5957075953483582)]
```




2장. 순환신경망

1절. 순환신경망

1.1. 순환신경망 개요

순환신경망(RNN; Recurrent Neural Network)은 1986년 데이비드 루멜하트(David Rumelhart)가 개발한 알고리즘입니다. 이 알고리즘은 시계열 데이터를 학습하는 딥러닝 기술입니다. RNN은 기준 시점(t)과 다음 시점($t+1$)에 네트워크를 연결합니다. RNN은 AI 번역, 음성인식, 주가 예측의 대표적 기술입니다.

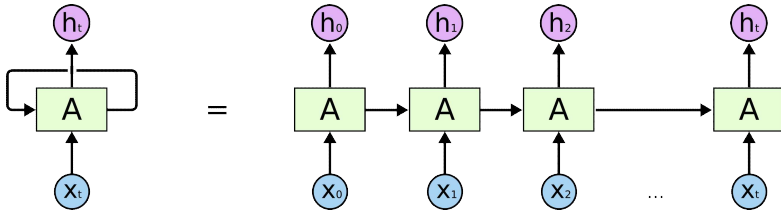


그림 12. RNN

출처: <https://colah.github.io>

인공신경망은 인간의 두뇌를 모방한 것입니다. 위 그림에서 왼쪽은 RNN의 마스터 구조를 표현한 것이며 마스터 구조에서 x 는 입력이며 y 는 출력입니다. RNN 구조는 풀어헤칠 수 있습니다. 풀어헤치면 오른쪽 그림이 됩니다. 물론 입력 데이터만큼 풀어헤치는 것입니다. 단어 5개짜리를 이용해서 분류하려면 5개로 풀어헤칩니다. 풀어헤친 그림에서 가로 화살표는 마스터 구조에서 f 를 의미합니다.

1.2. RNN의 종류

RNN은 다루는 문제에 따라 다릅니다.

- One to Many : 입력이 하나이고 출력이 여러 개 생성
- Many to One : 여러 입력이며, 하나의 출력을 생성
- Many to Many : 여러 입력이며, 출력도 여러 개 생성

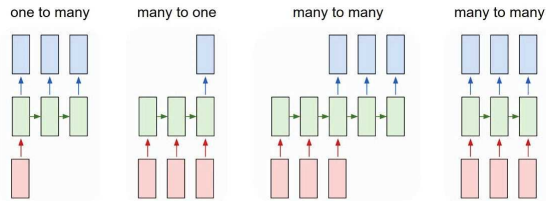


그림 13. RNN 종류

가장 많이 사용하는 것은 가운데 두 개입니다.

One to Many는 영상에 캡션을 달아야 할 때 사용합니다.

영화평을 분류할 경우 Many to One이 사용됩니다. 영화평을 위한 여러 단어가 들어가고 결과는 좋고 나쁨 등의 결과 하나만 필요하기 때문입니다.

위 그림에서 3번째 Many to Many 구조는 번역에 사용됩니다. 입력이 'I love you'라면 출력은 번역된 문장 '나는 너를 사랑해'가 될 것입니다.

1.3. Seq2Seq

번역기 기술은 PBMT(Phrase-based Machine Translation) 기술과 GNMT(Google Neural Machine Translation) 기술로 나뉩니다.

PBMT 기술은 문장을 구절 단위로 나누고 번역합니다. 이러한 방식보다 지금은 GNMT 기술을 사용합니다.

GNMT는 크게 인코더 입력, 디코더 입력, 그리고 디코더 출력으로 나뉘집니다. 그리고 그사이에 RNN 인공신경망이 포함되는 구조입니다. 이것을 이용하면 기존의 PBMT 기반 번역과 비교해서 더 나은 성능을 얻을 수 있습니다.

GNMT는 RNN 기반 시퀀스-투-시퀀스(Sequence to Sequence, Seq2Seq) 방식의 번역을 수행합니다. 시퀀스(Sequence)는 연관된 일련의 데이터를 의미하는데 예를 들면 문장에 시퀀스에 해당합니다.

Seq2Seq는 RNN을 연결하여 하나의 시퀀스 데이터를 다른 시퀀스 데이터로 변환하는 모델입니다. Seq2Seq는 입력 데이터를 처리하기 위한 인코더(Encoder)와 출력을 위한 디코더(Decoder)가 연결되는 구조입니다.

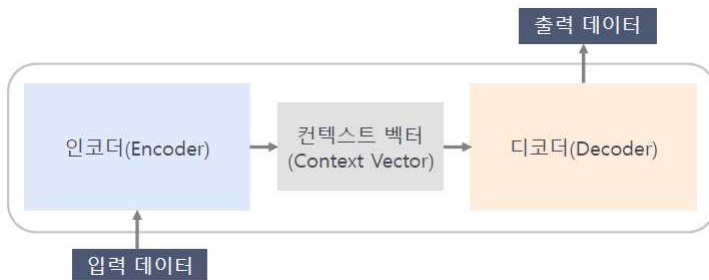


그림 14. Seq2Seq 구조

Seq2Seq는 인코더(Encoder), 컨텍스트 벡터(Context Vector) 그리고 디코더(Decoder)로 구성됩니다.

인코더는 차례로 입력된 문장들의 모든 단어를 압축하여 컨텍스트 벡터를 생성합니다. 컨텍스트 벡터는 차례로 입력된 문장의 모든 단어의 정보를 압축한 벡터이며 잠재 벡터(Latent Vector)라고도 합니다. 디코더는 입력된 컨텍스트 벡터를 이용하여 출력 시퀀스를 생성하고 출력합니다.

번역기에서 인코더는 입력 문장을 학습시키고, 디코더는 출력 문장을 학습시킵니다.

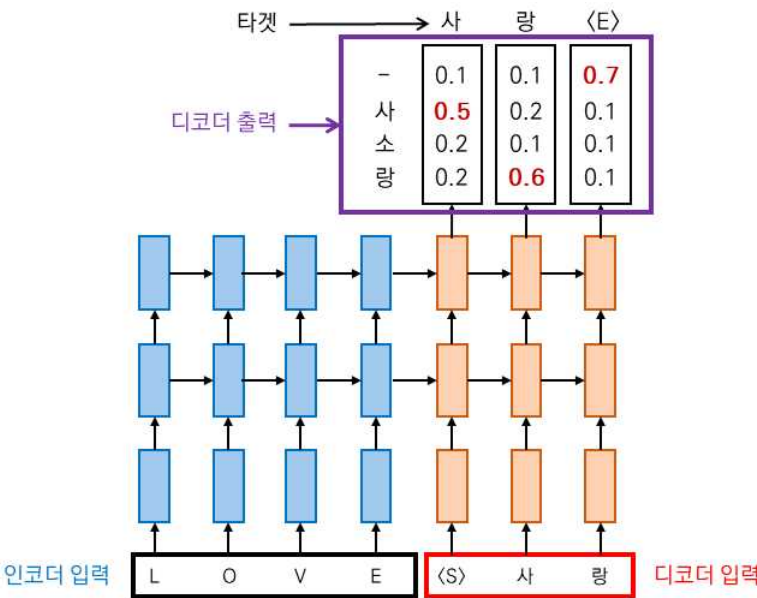


그림 15. GNMT

인코더와 디코더는 여러 개의 LSTM 셀로 구성합니다. 물론 인코더와 디코더의 입력은 단어들이 임베딩되어 전달되어야 하므로 임베딩 층이 있어야 합니다. 앞 그림에서 <S>는 디코더의 시퀀스 시작을 의미하고 <E>는 디코더 출력 시퀀스의 끝을 의미합니다.

1.4. Bidirectional RNN

양방향 RNN은 두 개의 RNN을 하나로 묶습니다. 하나는 순방향으로 가중치를 수정하며, 다른 하나는 역방향으로 가중치를 수정합니다. 이렇게 구성하면 RNN 단어가 길면 과거의 일을 잊는 것을 방지하고 또한 미래의 사실을 과거에 반영할 수 있는 구조입니다.

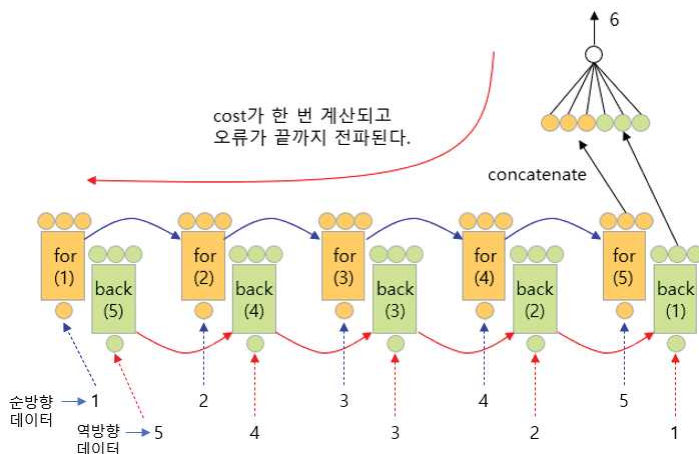


그림 16. Bidirectional RNN

2절. LSTM과 GRU

2.1. LSTM

LSTM(Long-Short Term Memory)은 1997년 셉 하이라이터(Sepp Hochreiter)와 쥘르겐 슈미더(Jurgen Schmidhuber)에 의해 만들어졌습니다. RNN의 단기 기억(Short-Term Memory) 문제를 해결하기 위해 만들어진 것이 LSTM입니다. 단기 기억은 지각시스템에서 수초에서 수분 사이의 일시적인 정보의 보유를 담당하는 기억 체계입니다. RNN에서 최근의 기억은 유지하지만 오래된 기억은 전달되지 않는 문제를 해결하고자 하는 것이 LSTM입니다.

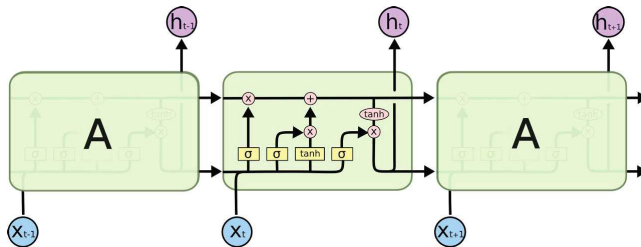


그림 17. LSTM

이 신경망의 특징은 두 가지가 있습니다.

첫 번째 혁신은 기존 RNN은 다음 단으로 전달되는 숫자가 벡터 하나이지만 LSTM 기술은 다음 단으로 전달되는 숫자가 두 개입니다. 기존 RNN에 셀 상태를 전달하는 벡터가 추가된 것입니다.

두 번째 혁신은 LSTM에는 셀의 상태를 저장합니다. 현 단계 정보를 수정해서 다음 단계로 전달할 때 게이트를 활용해서 정보를 더하거나

제거하는 개념입니다. 셀을 상태를 저장하기 위해 4개의 게이트를 사용합니다. 이 4개의 게이트는 망각 게이트, 새 정보 게이트, 셀 상태 게이트 그리고 출력 게이트가 있습니다. 이들 게이트에 대해 알아보겠습니다.

첫 번째 게이트는 망각 게이트입니다. 이 게이트는 셀 상태 라인에 추가/삭제 양을 조절해서 전 단계에서 오는 정보를 얼마나 잊어버릴지 결정합니다. 이때 시그모이드(sigmoid) 함수가 사용되며 값의 범위는 0~1입니다.

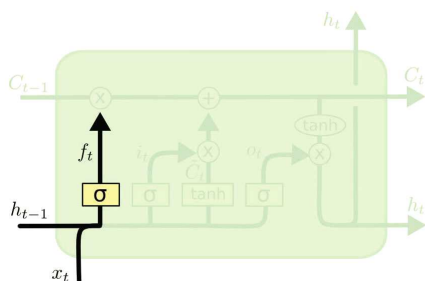


그림 18. 망각 게이트

두 번째 게이트는 새 정보 게이트입니다. 이 게이트는 새로 들어오는 입력값을 얼마나 받아들일지 결정합니다. 이 게이트에는 하이퍼볼릭탄젠트(tanh) 함수가 사용되며 -1~1 범위를 갖습니다.

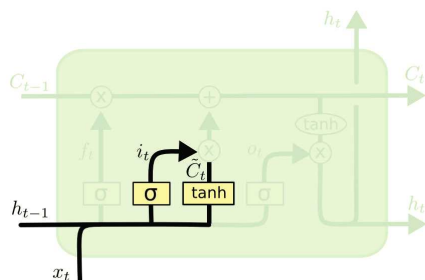


그림 19. 새 정보 게이트

세 번째 게이트는 셀 상태 출력 게이트입니다. 이 게이트는 다음 단계에 전해질 셀 상태의 양을 결정합니다.

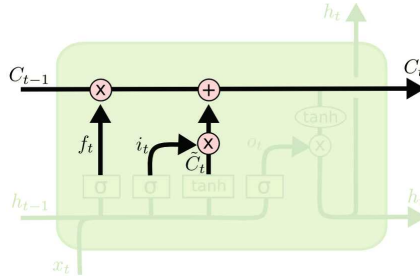


그림 20. 셀 상태 출력 게이트

네 번째 게이트는 출력을 만들기 위해서 사용합니다. 그러기 위해서 이전 상태의 sigmoid 함수를 적용한 결과와 셀 상태에 tanh를 적용한 결과를 결합해서 다음 단으로 출력합니다.

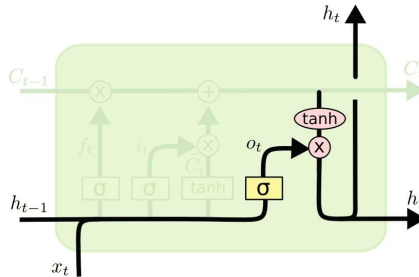


그림 21. 출력 게이트

2.2. GRU

RNN의 Short-Term Memory 문제를 해결하기 위해 만들어진 것이 LSTM입니다. 그런데 LSTM은 게이트의 복잡도 때문에 학습시간이 많이 소요됩니다. LSTM의 게이트를 단순화시켜 속도를 빠르게 개선

한 것이 GRU(Gated Recurrent Unit)입니다.

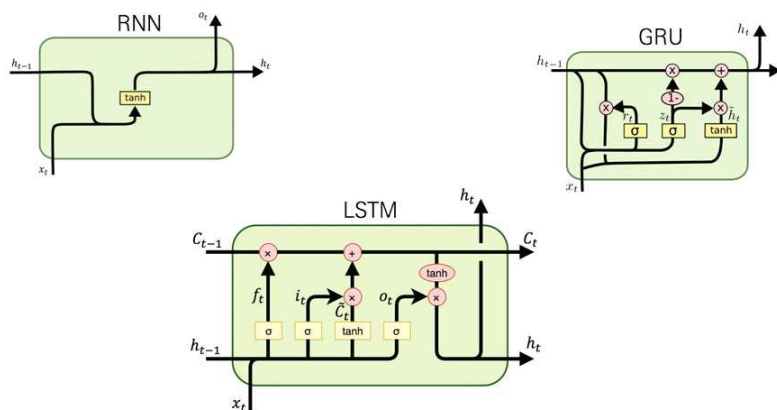


그림 22. RNN, LSTM, GRU

GRU는 LSTM에서 셀 상태(cell state) 라인을 제거했습니다. 그래서 게이트가 4개에서 2개로 줄어들었습니다. 정확도는 LSTM과 비슷하지만, 더 빠르다는 장점이 있습니다.

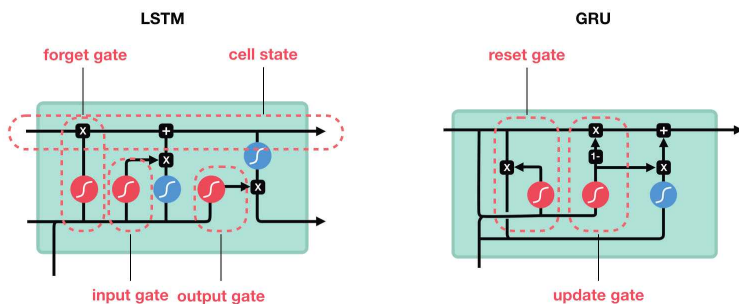


그림 23. LSTM과 GRU 구조

2.3. GRU Stacking

GRU는 쌓을 수 있습니다. 이렇게 하면 시계열 데이터의 학습 효과를 높일 수 있습니다. 다음 그림은 3개의 GRU 층을 쌓아 만들었습니다. 3개의 GRU 층 중에서 3번째 GRU는 `return_sequences=False` 설정을 해야 합니다.

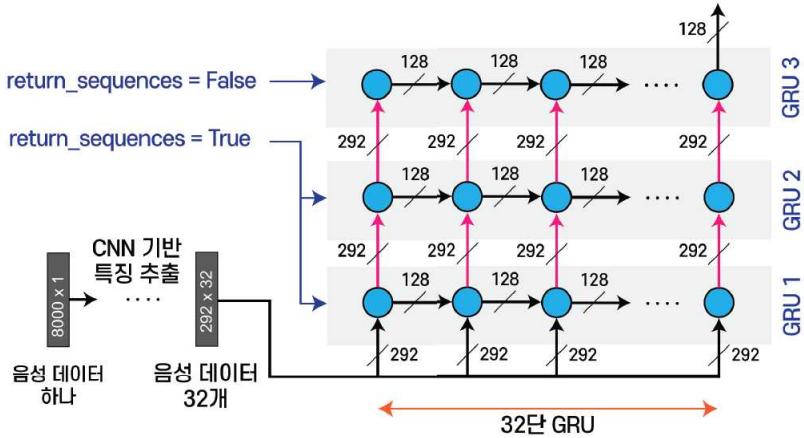


그림 24. GRU Stacking

3절. 어텐션

3.1. Seq2Seq의 문제점

Seq2Seq 모델은 인코더와 디코더로 구성되어 있습니다. 인코더는 입력 시퀀스를 하나의 벡터 표현으로 압축하고 디코더는 이 벡터 표현을 통해서 출력 시퀀스를 생성합니다.

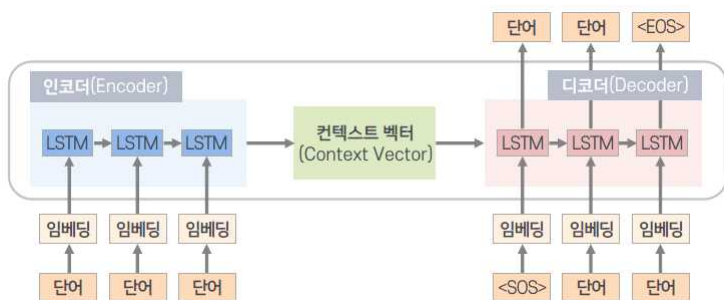


그림 25. Seq2Seq

인코더가 입력 시퀀스를 하나의 벡터로 압축하는 과정에서 입력 시퀀스의 일부 정보가 손실됩니다. 이거는 RNN 구조의 근본적인 문제점입니다. 이로 인해 경사 소실(Vanishing Gradient)이 발생할 가능성이 있고, 입력 데이터의 길이가 길어지면 성능이 저하되는 현상이 발생할 가능성이 있습니다.

입력 데이터의 길이가 길어지더라도 성능이 저하되는 것을 방지하기 위해 어텐션(Attention) 기법이 나왔습니다.

3.2. 어텐션

어텐션(Attention)의 기본 아이디어는 디코더의 출력 결과를 예측하는 때 시점(Timestep)마다 인코더의 은닉 상태(Hidden State)를 입력으로 전달하여 참고하는 것입니다.

어텐션은 전체 입력 문장 모두를 같은 비율로 참고하지 않습니다. 그래서 해당 시점에서 예측해야 할 단어와 연관 있는 입력 단어에 집중(Attention)합니다. 같은 문장 내 모든 단어 쌍 사이의 의미적 문법적 관계를 파악하여 밀접한 관계를 갖는 단어에 높은 어텐션(High Attention)을 부여합니다.

“She is eating a green apple” 문장이 있으면...



그림 26. High Attention과 Low Attention

eating은 apple과 밀접한 관계를 갖는 High Attention입니다.

green은 apple과 밀접한 관계를 갖는 High Attention입니다.

eating은 green과 밀접한 관계를 갖지 않는 Low Attention입니다.

어텐션 함수는 Q(Query), K(Keys), V(Values)를 매개변수로 사용합니다.

Q(Query)는 특정 시점에서의 디코더 셀의 은닉 상태입니다.

K(Key)는 모든 시점에서의 인코더 셀의 Q를 반영하기 전 은닉 상태입니다.

V(Value)는 모든 시점에서의 인코더 셀의 Q 반영 후 은닉 상태입니다.

어텐션 함수는 주어진 질의(Query)에 대해서 모든 키(Key)와 각각의 유사도를 계산합니다. 계산된 유사도를 키와 매핑되어 있는 각각의 값(Value)에 반영합니다. 유사도가 반영된 값(Value)을 모두 어텐션 값(Attention Value)으로 반환합니다.

$$Attention\ Value = Attention(Q, K, V)$$

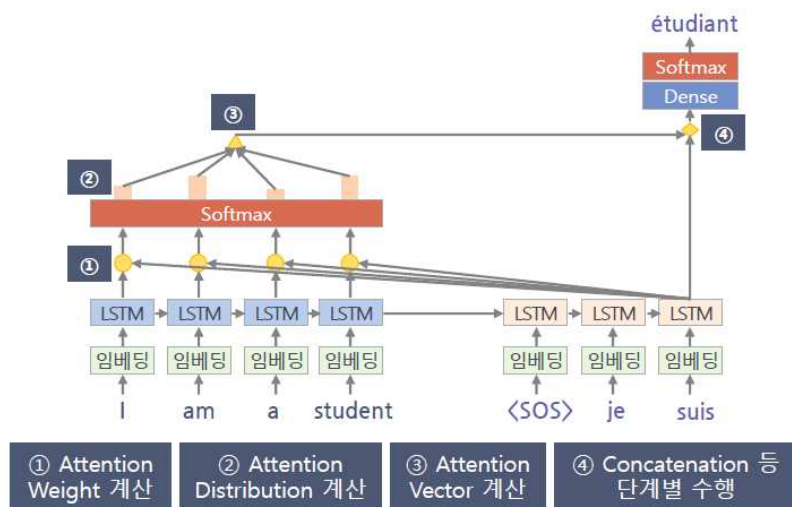


그림 27. Scaled Dot-Production Attention

디코더의 세 번째 LSTM 셀은 출력 단어를 예측하기 위해서 인코더의 모든 입력 단어들의 정보를 다시 한번 참고합니다. 소프트맥스 함수 적용 후 결과값은 입력된 단어들 각각이 출력 단어를 예측할 때 반영할 어텐션의 비율입니다.

3.3. 어텐션 적용 과정

Seq2Seq에 어텐션을 적용하기 위해서는 다음 단계로 수행합니다.

- ① 어텐션 가중치(Attention Weights) 계산
- ② 어텐션 분포(Attention Distribution) 계산
- ③ 어텐션 벡터(Attention Vector) 계산
- ④ 연결(Concatenation)

1) 어텐션 가중치 계산

디코더의 t 시점 Timestep의 은닉 상태와 인코더의 모든 Timestep의 은닉 상태와 행렬 곱셈 연산을 수행합니다. 행렬 곱 연산의 결과로 어텐션 가중치를 계산합니다.

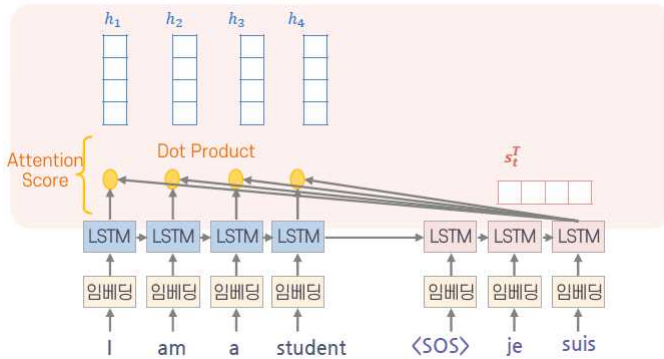


그림 28. 어텐션 가중치

2) 어텐션 분포 계산

인코더의 모든 은닉 상태의 어텐션 가중치의 벡터에 소프트맥스 (softmax) 함수를 적용합니다.

어텐션 분포(Attention Distribution)는 소프트맥스 함수를 적용한 후 각각 어텐션 가중치의 벡터 합이 1이 되는 확률 분포입니다. 각각의 값은 어텐션의 가중치에 해당합니다.

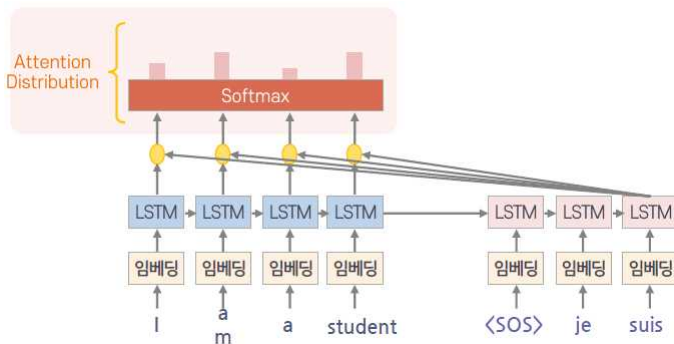


그림 29. 어텐션 분포 계산

3) 어텐션 벡터 계산

인코더의 은닉 상태와 어텐션 가중치를 곱한 후 모두 합하여 어텐션 벡터(Attention Vector)를 계산합니다. 어텐션 벡터는 인코더의 컨텍스트를 포함하는 컨텍스트 벡터(Context Vector)입니다.

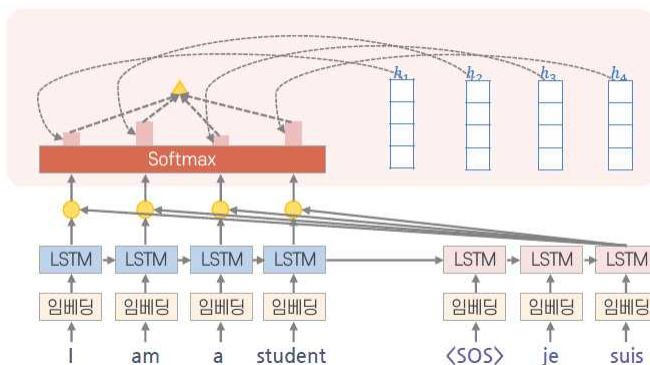


그림 30. 어텐션 벡터 계산

4) 연결

어텐션 벡터와 디코더의 t Timestep의 은닉 상태를 연결 (Concatenation)합니다.

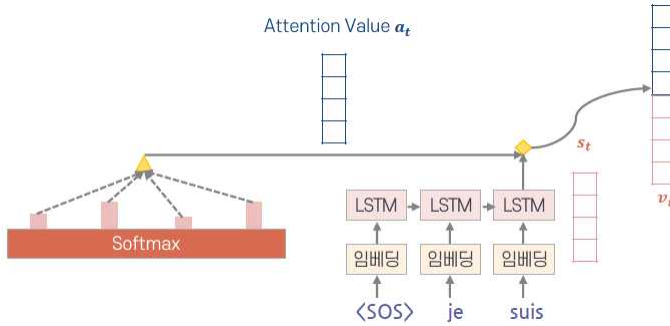
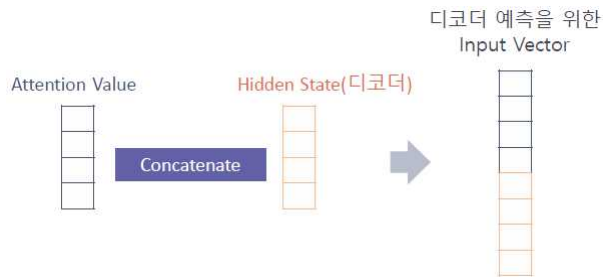


그림 31. 연결

어텐션 벡터와 디코더의 t Timestep의 은닉 상태를 연결한 벡터는 디코더에서 예측을 수행하기 위해 Fully Connected Layer의 입력 및 다음 Timestep의 입력으로 사용됩니다.



4절. 트랜스포머

4.1. 트랜스포머 개요

1) 트랜스포머란?

트랜스포머(Transformer)는 2017년 구글이 “Attention is all you need” 논문에서 발표한 모델입니다. 트랜스포머는 Seq2Seq 구조인 인코더-디코더 구조로 되어있으면서 RNN을 제외하고 어텐션만으로 구현한 모델입니다.

트랜스포머는 학습속도가 개선되었으며 기존 RNN을 사용한 모델보다 우수한 성능을 냅니다.

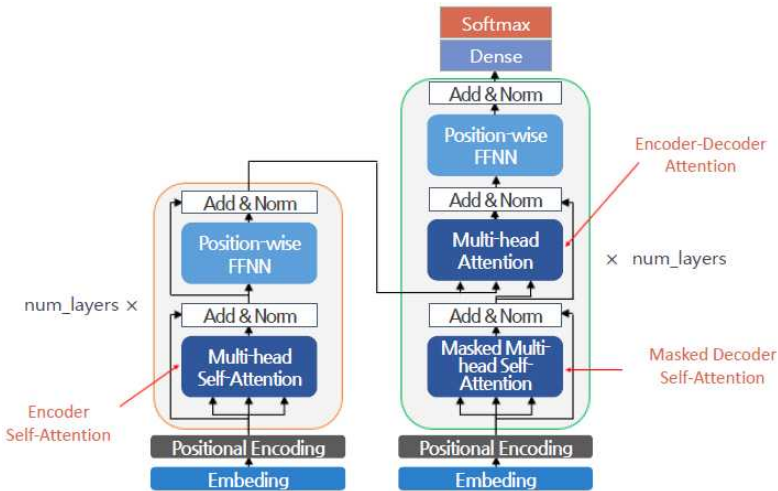


그림 33. 트랜스포머

출처: <https://wikidocs.net/31379>

2) 트랜스포머 구조

입력 문장에 대한 처리 후 출력 문장을 출력합니다. 트랜스포머는 인코더와 디코더 및 인코더와 디코더를 연결하기 위한 Connection을 구성합니다.

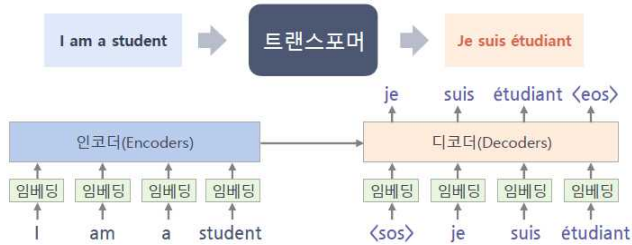


그림 34. 트랜스포머 구조

3) 트랜스포머의 인코더와 디코더 구조

인코딩 컴포넌트는 여러 인코더로 구성되어 있습니다. 디코딩 컴포넌트는 인코딩 컴포넌트와 같은 개수의 디코더로 구성됩니다. 논문에는 6개의 인코더와 디코더로 구성되어 있으나 임의의 개수로 변경 가능합니다.

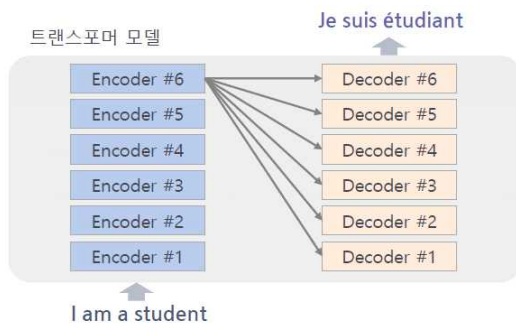


그림 35. 트랜스포머 모델

4) 트랜스포머의 임베딩과 Positional 인코딩

입력 단어들을 임베딩하여 벡터로 변환합니다. 각 단어는 512 크기의 벡터로 임베딩됩니다.

순차적인 특성을 가지는 RNN을 제거함으로써 임베딩 벡터의 각 단어에 대한 위치 파악이 불가능합니다. 그래서 트랜스포머에서 각 단어에 대한 위치 정보를 부여할 필요가 있습니다.

Positional 인코딩은 트랜스포머에서 각 입력 단어의 위치 정보 부여를 위해 각 단어의 임베딩 벡터에 위치 정보들을 추가하여 모델의 입력으로 사용하는 방법입니다.

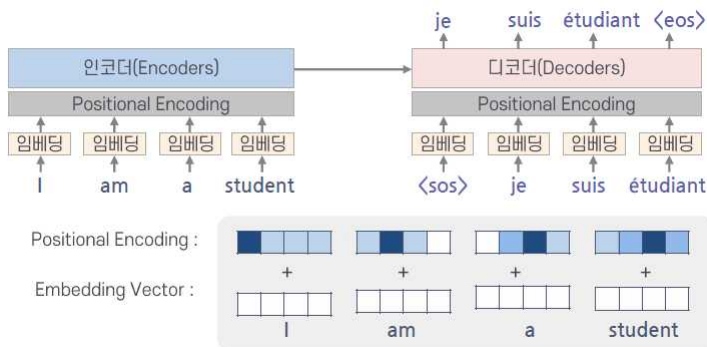


그림 36. Positional Encoding

5) 트랜스포머의 Self-Attention

트랜스포머의 Self-Attention은 현재 처리 중인 단어에 대해 다른 연관 있는 단어들과의 맥락을 파악하는 방법을 제공합니다.

다음 문장이 있다고 가정하세요.

‘The animal didn’t cross the street because it was too tired’

이 문장의 경우 “it”이 가리키는 것은 무엇일까요? animal일까요? 아니면 street일까요?

Self-Attention을 이용하여 animal과 it을 연결할 수 있습니다.

Self-Attention의 첫 단계는 입력 문장에 대해 Query, Key, Value를 계산하는 것입니다. 입력 문장의 512 크기의 벡터와 학습할 가중치(W^Q , W^K , W^V)를 곱하여 64 크기의 Query, Key, Value 벡터를 생성합니다.

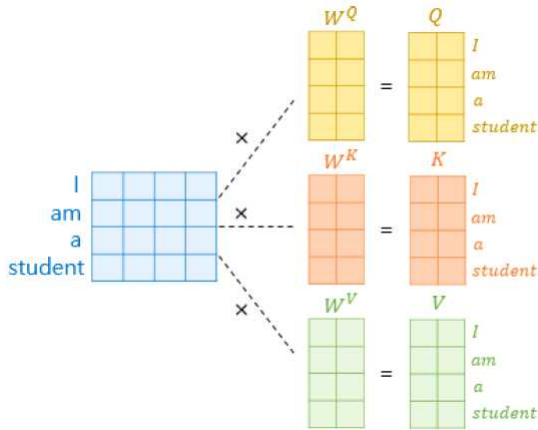


그림 37. Self-Attention의 Query, Key, Value 계산

Query에 Key의 전치(Transpose) 행렬을 곱해서 내적을 계산합니다. 만일 Query와 Key가 특정 문장에서 중요한 역할을 하고 있다면 트랜스포머는 이들 사이의 내적(Dot Product)값을 크게 하는 방향으로 학습합니다. 내적 값이 커지면 해당 Query와 Key가 벡터 공간상 가까이 있을 확률이 높습니다.

Key의 벡터 크기인 64의 제곱근인 8로 나눈 후 소프트맥스 함수를 적용합니다. Value와 내적을 곱하여 어텐션 값인 Z를 계산합니다.

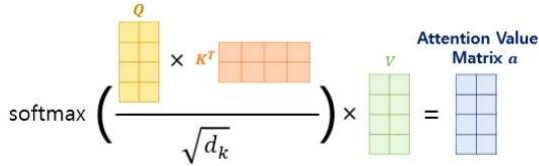


그림 38. 어텐션 값 계산

이를 수식으로 표현하면 다음과 같습니다.

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

위 수식에서 $\sqrt{d_k}$ 로 나눈 이유는 Query와 Key의 내적 행렬의 분산 (Variance)을 축소하고 경사 소실(Gradient Vanishing)을 발생을 방지하기 위해서입니다.

4.2. Multi-head Self Attention

1) Multi-head Self Attention

단일한 Self-Attention을 수행하는 것보다 다수의 Self-Attention을 병렬로 수행하는 것이 효과적입니다. Input 문장에 대해 N개의 Query, Key, Value를 계산하기 위한 N개의 가중치(WQ, WK, WV)를 생성하여 병렬로 Self Attention 수행하고 N개의 Attention Value를 계산합니다. Attention을 병렬로 수행함으로써 다양한 관점에서 단어 간 관계 정보 파악이 가능합니다.

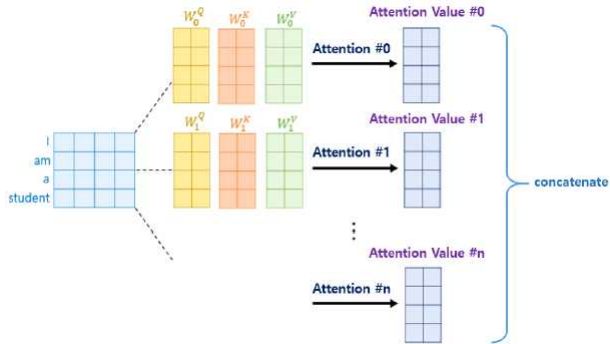


그림 39. Multi-head Self Attention

2) Multi-head Attention Value Matrix

Multi-head Attention Value Matrix 계산은 Multi-head Attention 수행 결과를 연결하고 학습할 가중치를 곱하여 Multi-head Attention Value Matrix를 최종 결과로 도출합니다.

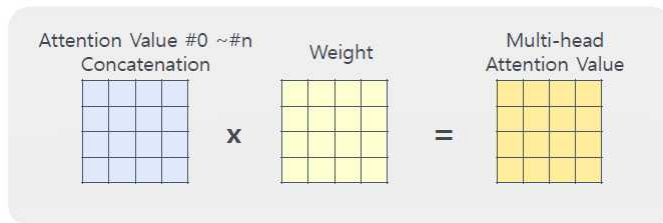


그림 40. Multi-head Attention Value Matrix 계산

3) Position-wise FFN

Position-wise FFN(Feed Forward Neural Network)은 인코더와 디코더에서 공통으로 포함된 Sub-layer입니다.

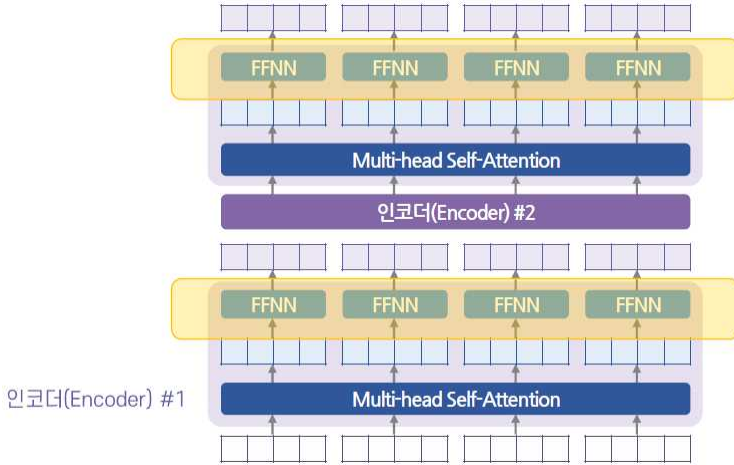


그림 41. Position-wise FFN

4) 잔차 연결

신경망이 깊어지면 경사 소실(Gradient Vanishing) 또는 경사 폭발(Gradient Exploding) 발생으로 인해 학습이 잘 이루어지지 않는 경우가 있습니다. 경사가 줄어드는 문제를 해결하기 위해 Google은 ResNet을 통해 Residual Connection 적용했습니다.

네트워크의 입력과 출력이 더해진 것을 다음 층의 입력으로 사용하는 Skip Connection을 적용했습니다. 스킵 커넥션을 구현하는 것은 덧셈 연산의 추가만으로 가능합니다. 추가적인 연산량이나 파라미터가

불필요합니다. Residual Connection은 역전파(Back Propagation) 수행 과정에서 경사(Gradient)가 이전 계층으로 잘 전달되도록 합니다.

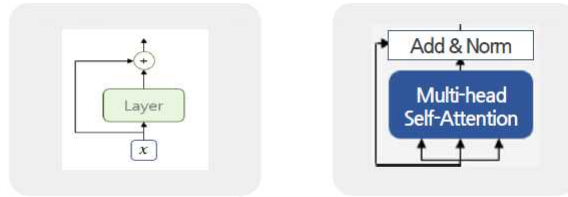


그림 42. Residual Connection(좌), Transformer Residual(우)

5) 정규화

Residual Connection으로 전달된 입력값과 Multi-head Attention Value를 더한 후 정규화(Normalization)를 수행합니다. Position-wise FFNN의 입력값으로 전달하기 전 정규화를 수행함으로써 과적합(Over-fitting)을 방지할 수 있습니다.

5절. 인공지능 언어 모델

5.1. 언어 모델 개요

1) 언어 모델

문맥을 고려하여 문장(Sentence)을 기반으로 임베딩을 수행하는 것을 언어 모델(Language Model)이라고 합니다.

언어 모델에는 ELMo(Embeddings from Language Model), GPT(Generative Pre-trained Transformer), BERT(Bidirectional Encoder Representations from Transformers), GPT2 등의 임베딩 방법이 있습니다.

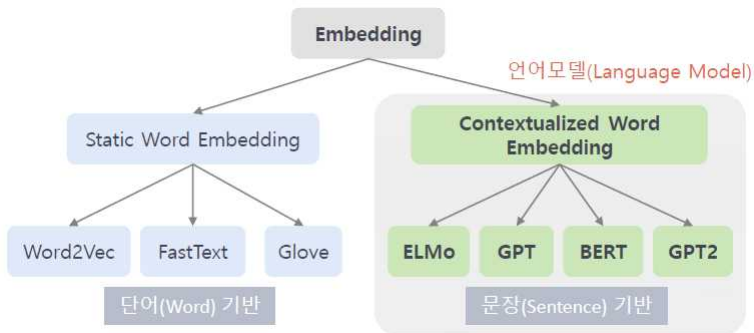


그림 43. 임베딩과 언어 모델

2) 최신 언어 모델

언어 모델의 성능 평가에는 GLUE와 SuperGLUE라는 벤치마크가 주로 활용됩니다. 기존 9가지 레이블링 데이터셋을 통합한 GLUE는 다양한 과제에서 인공지능 언어 모델의 성능 평가의 지표로 활용됩니다.

트랜스포머 등장 이후 2018년~2020년 기간 동안 트랜스포머를 이용한 다양한 언어 모델의 등장 및 인공지능 언어 모델의 폭발적인 성장이 있었습니다.

다음 표는 2018년부터 2020년 사이 발표된 언어모델입니다.

언어모델	발표 시기	모델 발표 기관
ELMo	2018년 2월	Allen AI와 Washington University
GPT-1	2018년 5월	Open AI
BERT	2018년 10월	Google
GPT-2	2019년 2월	Open AI
KoBERT	2019년 6월	ETRI
XLNet	2019년 7월	Carnegie Mellon University와 Google Brain
RoBERTa	2019년 7월	Facebook AI Research
ALBERT	2019년 9월	Google과 TTIC(Toyota Technological at Chicago)
T5	2019년 10월	Google
KoGPT-2	2020년 4월	SKT와 AWS
GPT-3	2020년 6월	Open AI

3) 언어 모델의 분류

언어 모델은 문장의 학습 방향에 따라 순방향, 역방향, 양방향 언어 모델로 나뉩니다. 순방향 언어 모델에는 OpenAI GPT 계열, ELMo가 있으며 역방향 언어 모델은 ELMo, 완전한 양방향 언어 모델은

BERT 계열, XLNet입니다. ELMo는 순방향, 양방향 계층을 각각 독립적으로 학습하기 때문에 진정한 의미의 양방향 모델로 보기 어렵습니다.

5.2. 언어 모델 성능 평가

1) 스쿼드(SQuAD)

스쿼드(SQuAD, Stanford Question Answering Dataset)는 인공지능 언어 모델들이 문서를 이해한 후 질문에 대한 정답을 찾아내는 기계 독해 이해(MRC, Machine Reading Comprehension) 성능 평가를 위해 만든 데이터셋입니다. 순위는 스탠퍼드 대학에서 제공하는 10만 개의 질문과 답으로 정해집니다.

2) KorQuAD 1.0

KorQuAD 1.0은 한국어 기계 독해 이해 성능 평가를 위해 만든 데이터 세트입니다. 이것은 모든 질의에 대한 답변은 해당 위키피디아 기사 문단의 일부 하위 영역으로 구성됩니다.

3) KorQuAD 2.0

KorQuAD¹⁾ 2.0은 KorQuAD 1.0에서 질문 답변 20,000개 이상의 쌍을 포함하여 총 100,000개 이상의 쌍으로 구성된 한국어 기계 독해 이해 데이터셋입니다. KorQuAD 1.0과는 다르게 1~2개 문단이 아닌 위키피디아 기사 전체에서 답을 찾아야 합니다.

1) <https://korquad.github.io/>

긴 문서들이 있으므로 탐색 시간에 대한 고려가 필요하며 표와 리스트도 포함되어 있으므로 HTML 태그를 통한 문서의 구조 이해도 필요합니다.

5.3. 언어 모델 전이 학습

1) 전이 학습이란?

전이 학습(Transfer Learning)은 해결하고자 하는 문제에 적합하도록 적용(Fine-tuning)하여 학습시키는 기법입니다. 대규모 데이터셋으로 사전에 학습이 완료된 언어 모델(Pre-trained Language Model)을 사용할 수 있습니다.

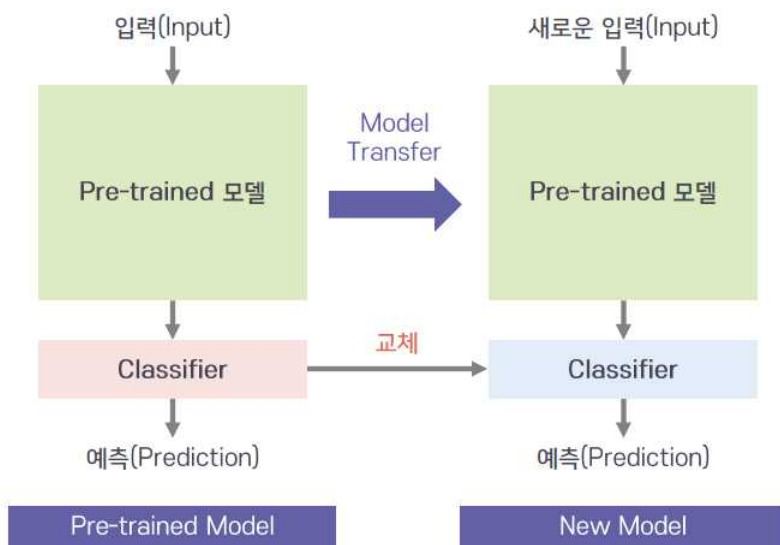


그림 44. 전이 학습

2) Pre-trained 모델의 Fine-tuning

전이 학습을 통해 새롭게 생성된 모델에 대한 최적화된 Train 전략은 3가지가 있습니다.

- Train Entire Model
- Frozen Partial
- Train Classifier Only

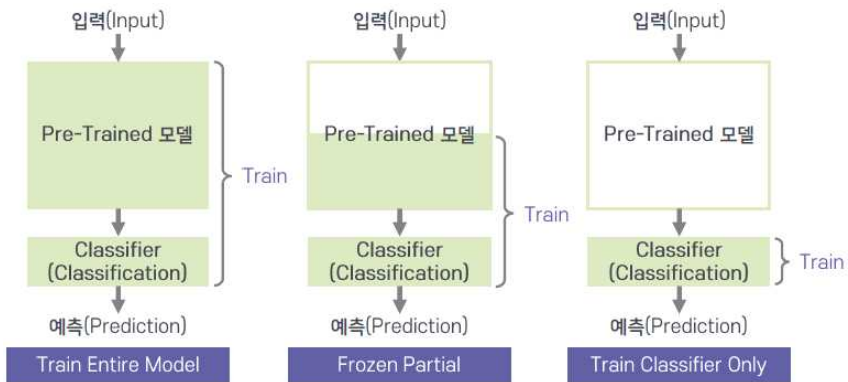


그림 45. 전이 학습 전략

Train Entire Model 전략은 모델 전체를 학습시키는 전략입니다. 모델의 구조만 사용하면서 새로운 모델의 데이터셋에 맞게 모델 전체를 새롭게 학습시키는 전략입니다.

Frozen Partial 전략은 모델의 일부분은 고정된 상태에서 모델의 나머지 부분과 분류기만 학습시키는 전략입니다.

Train Classifier Only 전략은 모델 전체를 고정된 상태에서 새로운 모델의 분류기만 학습시키는 전략입니다.

3) 전이 학습 전략의 선택

전이학습 전략은 데이터의 크기와 유사도에 따라 선택을 다르게 할 필요가 있습니다.

다음 그림은 데이터의 크기와 유사도에 따라 어떤 전략을 선택하는 것이 좋을지를 보여줍니다.

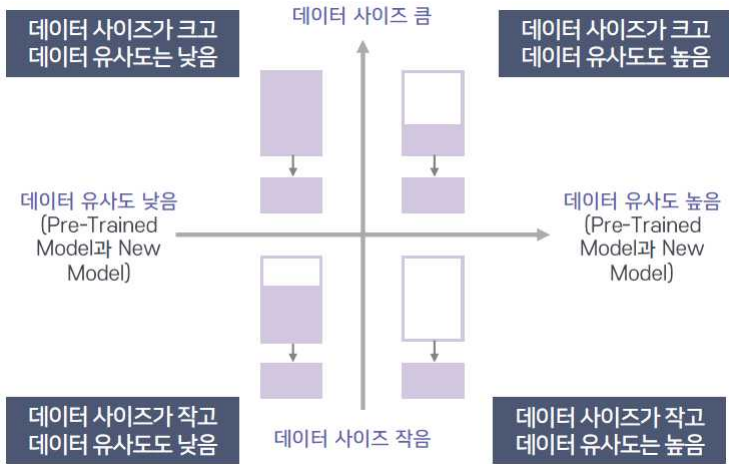


그림 46. 전이 학습 전략이 선택

5.4. 최신 인공지능 언어 모델

1) BERT

BERT(Bidirectional Encoder Representations from Transformers)는 2018년 10월 구글이 발표한 Pre-trained 기반 딥러닝 언어 모델입니다.

대형 코퍼스에서 비지도 학습(Unsupervised Learning)으로 General-Purpose Language Understanding 모델을 구축(Pre-training) 합니다.

지도 학습(Supervised Learning)으로 Fine-Tuning 해서 QA, STS 등의 Downstream NLP 태스크에 적용하는 Semi-Supervised Learning 모델입니다.

트랜스포머로 구현되어 다양한 자연어처리 분야에 활용되며, 트랜스포머의 인코더만 사용하며, BERT Base는 12개의 트랜스포머 블록을 사용하고, Bert Large는 24개의 트랜스포머 블록을 사용합니다.

사전 훈련데이터로 Books Corpus 8억 개 단어, Wikipedia 25억 개 단어를 사용했습니다.

2) RoBERTa

RoBERTa(A Robustly Optimized BERT)는 BERT보다 성능을 한 단계 업그레이드 한 버전입니다. 모델의 크기를 크게 하여 성능을 높이는 방법을 적용했습니다. 훈련데이터의 양은 13GB에서 160GB로 증가 됐고, 학습횟수는 125,000회에서 500,000회로 늘어났습니다. 배치 크기는 256에서 9,192로 커졌고, 사전의 크기 또한 32,000에서 50,000으로 커졌습니다.

BERT의 Next Sentence Prediction은 훈련에서 제외했습니다.

3) ALBERT

ALBERT(A Lite BERT)는 모델 매개변수 수를 줄여 같은 구조의 모델에서의 메모리 사용량을 줄이고 학습속도는 개선했습니다.

각 단어를 저 차원의 임베딩 벡터로 표현합니다. 이를 다시 모델 은닉층의 차원 수만큼 확장합니다. 트랜스포머 인코더 블록 간 매개변수를 공유합니다. 그래서 BERT(Large)보다 매개변수 수는 1/18로 줄어듭니다.

GLUE 성능을 일정 수준 유지하면서(85.2→82.4) 학습속도를 1.7배 개선했습니다. BERT의 Next Sentence Prediction 대신 두 문장의 연관 관계를 예측하는 문장 순서 예측(Sentence Order Prediction, SOP)을 통해 훈련 성능 개선합니다.

4) T5

하나의 모델로 모든 문제를 해결할 수 있도록 모델의 크기를 대폭 확대한 모델입니다. GLUE와 SQuAD 등에 포함된 다양한 자연어 이해(NLU) 데이터셋을 사전 훈련에 사용합니다.

정제 텍스트 데이터 700GB, 모델 매개변수는 110억 개 규모입니다. 이것은 RoBERTa의 4.4배이며, 모델 매개변수는 32배입니다.

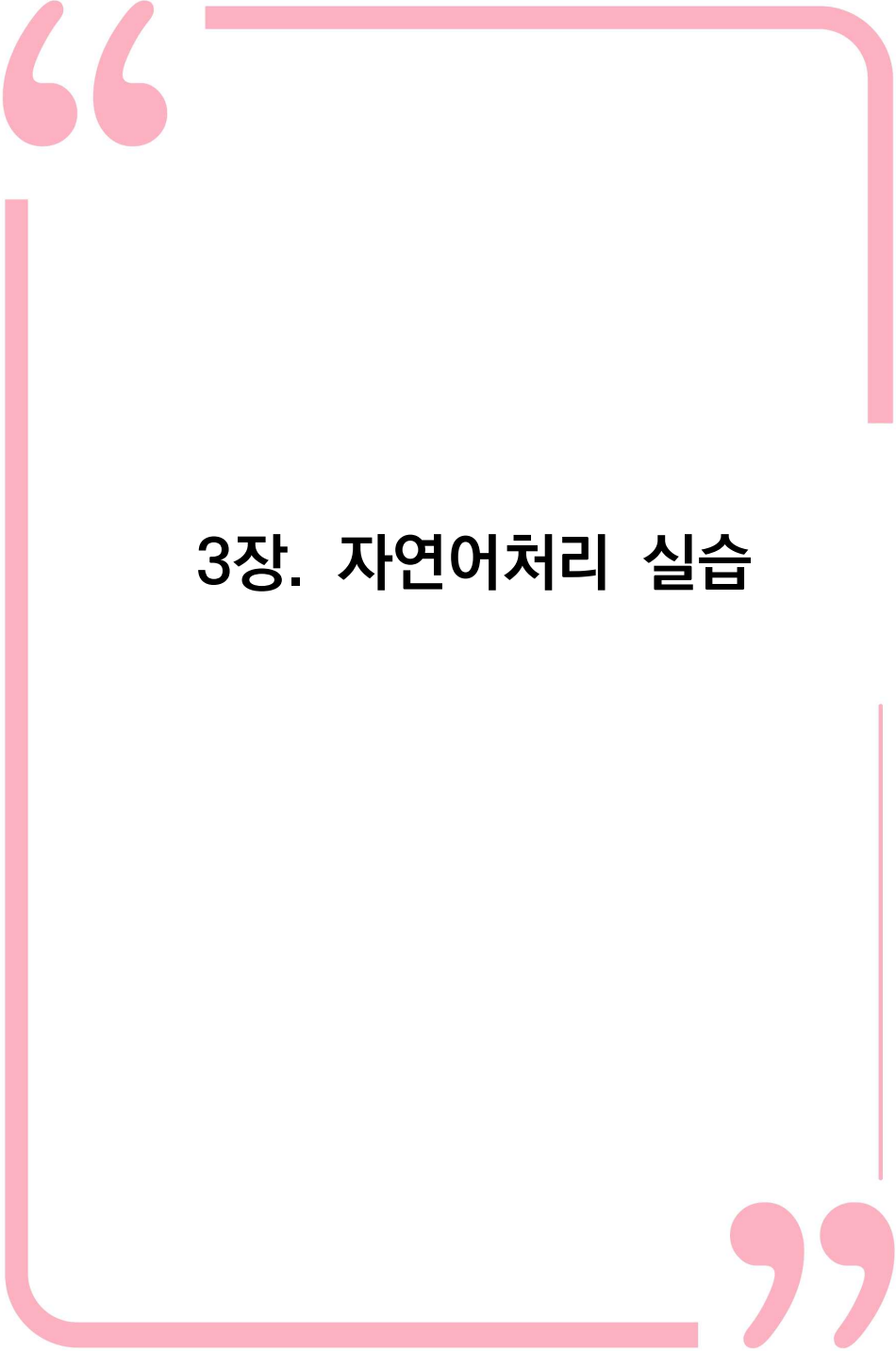
빈칸 하나의 예측값이 다른 빈칸 예측에 영향을 주지 않는 BERT의 한계를 극복하기 위해 Seq2Seq 구조의 Masked Language Model을 적용합니다. 이로 인해 T5는 SuperGLUE에서 인간과 비슷한 성능을 달성했습니다.

5) KoBERT

엑소브레인(Exobrain)²⁾ 사업에서 한국어의 특성을 반영하여 개발한 BERT 언어모델입니다. ETRI의 엑소브레인 연구진이 배포하는 한국어 최첨단 딥러닝 언어 모델은 한국어 분석, 기계 독해, 문서 분류 등 다양한 태스크에 활용 가능합니다.

5종의 한국어 처리 태스크에서 구글이 배포한 한국어 언어 모델과 비교 평가한 결과, ETRI의 언어 모델이 평균 4.5% 우수한 성능을 냈습니다. 여기에 사용한 5종은 의미역 인식, 기계 독해, 단락 순위화, 문장 유사도 추론, 문서 주제분류입니다.

2) 과학기술정보통신부와 IITP의 혁신성장동력 프로젝트로 추진 중인 사업입니다.
<https://ko.wikipedia.org/wiki/엑소브레인>

A large, stylized pink quotation mark is positioned at the top left, and a matching pink frame with rounded corners surrounds the central text. A vertical pink line extends from the bottom right of the frame towards the bottom right quotation mark.

3장. 자연어처리 실습

1절. 개요

1.1. 학습 환경

1) 텐서플로우와 케라스

텐서플로우(Tensorflow)는 파이썬 기계학습용 라이브러리이며, 구글에서 무료로 배포하고 있습니다. 2015년도에 버전 1.0 소스코드가 발표되었고, 구글의 TPU(Tensor processing Unit)와 연동됩니다. 버전 2.0의 소스가 2019년도 공개되었으며, PyTorch 기술을 많이 도입했습니다.

케라스는 프랑소와 솔레(Francois Chollet) 팀이 2015년도에 공개했으며 텐서플로우를 쉽게 사용하도록 하는 도우미 역할을 합니다. 이로 인해 인공지능망 구현, 학습, 평가가 매우 쉬워집니다. 그러나 텐서플로우에 비해 느리므로 기업형 서비스에는 적합하지 않습니다.



그림 48. Keras vs. TensorFlow

우리는 지금 AI를 배우고 있습니다. AI는 단일 기술이 아닙니다. AI를 이용해 어떤 문제를 해결하려면 다음의 기술들을 익혀야 합니다.

- 데이터 과학 + 알고리즘 + 자료구조
- 수학 + 통계학
- 관련 분야의 지식

2) 구글 코랩

이 설명서는 구글이 제공하는 주피터 노트북인 구글 코랩(Colab)을 사용하여 코드를 설명합니다. 구글 코랩은 구글이 제공하는 클라우드 환경의 컴퓨터에서 실행합니다. 구글이 제공하는 주피터 노트북은 웹 브라우저 기반이므로 프로그램을 설치할 필요가 없습니다.

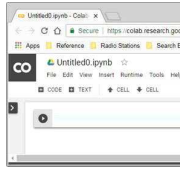


그림 49. Colab

직접 주피터 노트북을 사용하여 개발할 경우는 내 컴퓨터에서 동작하고 직접 코드를 관리할 수 있지만 다른 사람이 만든 패키지를 설치해서 사용할 때 관리가 쉽지 않다는 것입니다. 패키지는 다른 패키지에 의존하는 상황이 발행하고 이는 패키지의 버전 관리로 이어집니다.

3) 구글 캐글

캐글은 2010년 설립된 머신러닝/딥러닝 온라인 커뮤니티입니다. 기업 및 단체에서 데이터와 해결 과제를 등록하고, 데이터 과학자들이 모델을 개발하고 경쟁합니다. 캐글은 다양한 빅데이터를 무료로 제공합니다. 캐글은 2017년도에 구글에 인수되었습니다.



그림 50. Kaggle

1.2. 자연어처리의 몇 가지 예

1) 자동 이미지 캡션처리

요즘의 화두 중 하나는 자동 이미지 캡션처리입니다. AI는 이미지를 보고 비전(CNN)과 언어(RNN)를 결합한 인공지능이 되어 갑니다.



그림 51. a man in beige **guitar** is playing **pants**



그림 52. a young boy is holding a **baseball bat**

RNN과 CNN 둘 다 잘되는 것이 중요합니다. 첫 번째 그림과 캡션을 guitar와 pants의 순서가 잘못되었습니다. 이것은 RNN의 결과가 나쁨을 의미합니다. 두 번째 그림에서 캡션의 문법은 틀리지 않지만, 아이가 쥐고 있는 것은 야구 배트(baseball bat)가 아니므로 이미지 처리를 위한 CNN의 결과가 나쁩니다.

2) 구글의 이미지 캡션처리

구글의 자동 이미지 캡션처리 대회는 3백만 개의 사진에 자동 캡션을 다는 기술을 경쟁합니다. CNN을 이용해 이미지를 인식하고, RNN을 이용해 추출된 단어들을 문장으로 전환합니다.

구글이 제공하는 서비스 중에 가장 돈을 많이 버는 서비스는 검색하는 서비스입니다. 물론 검색의 결과와 매칭되는 광고를 통해 돈을 버는 것입니다. 과거에는 구글에서 사용자들이 단어 기반으로 검색을 했지만, 지금의 사용자들을 문장을 이용해 검색합니다. 물론 구글도 그에 맞게 문장을 인식하고 그에 맞는 서비스를 제공합니다.

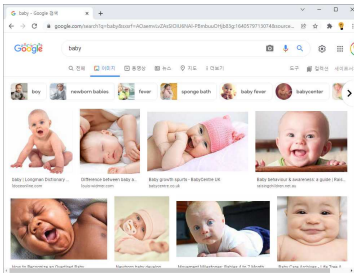


그림 53. 검색어: baby

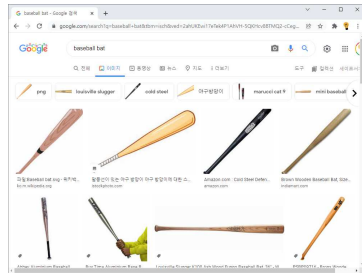


그림 54. 검색어: baseball bat

다음 그림은 문장으로 검색한 결과입니다.

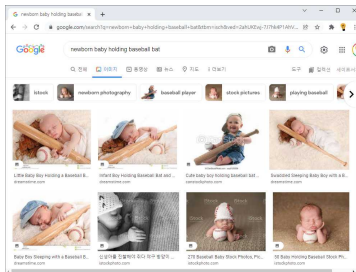


그림 55. 검색어: newborn baby holding baseball bat

3) 페이스북의 가짜 뉴스 판별

페이스북은 딥러닝 기술로 가짜 뉴스를 판별합니다. AI가 가짜 뉴스 판별을 위해 가짜 뉴스를 직접 작성합니다. 글의 저자가 기계인지, 사람인지 구분조차 어려운 상황입니다. MIT는 2019년도 AI로 AI가 쓴 글을 찾아내는 방법을 연구하기도 했습니다. 이는 가짜 뉴스를 판별하는데 탁월한 효과가 있으며 이를 SNS에 활용합니다.

페이스북은 2021년 10월 28일 사명을 메타(Meta)로 변경한다고 밝혔습니다.

4) 오피니언 마이닝(Opinion Mining)

오피니언 마이닝(평판 분석) 또는 감성 분석(Sentiment Analysis)이라고도 합니다. 컴퓨터 언어학과 자연어처리를 사용하여 텍스트(긍정적, 부정적, 중립적 등) 내에서 감정이나 의견을 자동으로 식별하고 추출하는 텍스트 분석 기술입니다.

이를 통해 고객의 머릿속으로 들어가 고객이 무엇을 좋아하고 싫어하는지, 그 이유를 파악하여 고객의 요구에 맞는 제품과 서비스를 만들 수 있습니다. 적절한 도구가 있으면 거의 모든 형태의 구조화되지 않은 텍스트에 대해 사람의 입력이 거의 필요 없이 자동으로 의견 마이닝을 수행할 수 있습니다. 오피니언 마이닝은 대중의 생각과 감정을 들여다볼 수 있는 창을 제공하여 기업이 고객 경험을 개선하고 경쟁력 있는 연구를 수행하며 의견을 이해할 수 있도록 합니다.

다음은 인기 있는 오피니언 마이닝 감정분석입니다.

- 소셜 미디어 분석
- 브랜드 인지도

- 고객 피드백
- 고객 서비스
- 시장 조사
- 마케팅 캠페인 평가

코랩에서 맷플롯립으로 시각화할 때 한글 폰트를 사용하려면 아래 내용을 참고하세요.



CoLab에서 Matplotlib 한글 폰트 지정 방법

- ▶ 셀에 아래 내용을 입력하소 실행한 후 런타임을 다시 시작하세요.
!sudo apt-get install -y fonts-nanum
!sudo fc-cache -fv
!rm ~/.cache/matplotlib -rf
- ▶ import matplotlib.pyplot as plt
- ▶ plt.rc('font', family='NanumBarunGothic') # 한글 폰트
- ▶ import matplotlib as mpl
- ▶ mpl.rc('axes', unicode_minus=False) # 유니코드 "-" sign

2절. RNN으로 영화평 분류하기

2.1. IMDB 영화평 데이터셋

IMDB(Internet Movie DataBase) 데이터셋은 영화 감상평을 제공합니다.

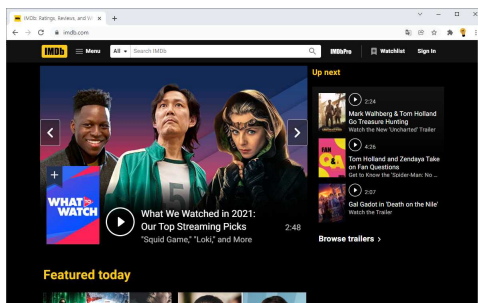


그림 56. imdb.com

다음 화면은 ‘오징어 게임’의 사용자 리뷰 화면입니다.

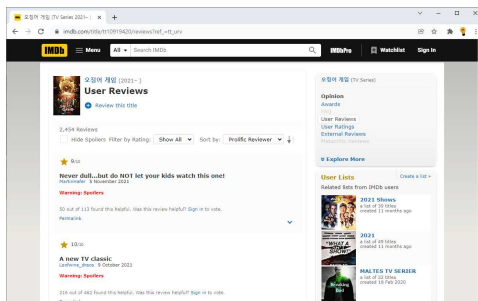


그림 57. imdb의 사용자 리뷰

여기에서는 영화의 리뷰 점수를 제공하지만, 우리가 사용할 데이터는 점수가 아닌 긍정/부정으로 분류할 데이터입니다. 우리가 이 데이터를

직접 수집하지는 않습니다. 케라스는 imdb 데이터셋을 제공하는데 이 데이터셋은 점수를 제공하지 않고 이 영화의 리뷰 텍스트와 긍정/부정 라벨을 제공합니다.

2.2. 구글 코랩에서 영화평 분석하기

코드 셀과 텍스트 셀을 사용할 수 있는데 화면에서 [+ 텍스트]를 클릭해서 이 코드의 간단한 설명을 포함할 수 있습니다.

코드 셀에 `print('hello world')`를 입력하고 이 셀의 실행을 위해서는 `Ctrl+Enter`를 누르면 쉽게 실행시킬 수 있습니다.

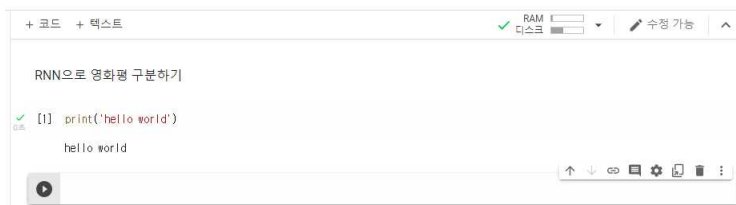


그림 58. 구글 코랩 실행

화면에 hello world 실행결과가 출력되면 코랩을 사용할 준비가 된 것입니다.

- 코드 셀에 라인 번호를 보고 싶다면 [도구] -> [설정] -> [편집기]에서 [행 번호 표시]에 체크해 주세요.

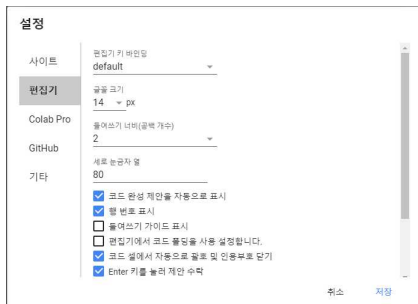


그림 59. 코랩 라인 번호 표시

화면의 빈 코드 셀을 추가하고 이곳에 다음 내용을 입력합니다. 다음 내용은 패키지를 가져옵니다.

```
1 import numpy as np
2 from keras.datasets import imdb
3 from keras.preprocessing.sequence import pad_sequences
4 from time import time
5 from keras.models import Sequential
6 from keras.layers import Embedding, LSTM, Dense
7 from sklearn.metrics import confusion_matrix, f1_score
```

다음은 하이퍼 파라미터를 설정합니다.

```
1 MY_WORDS = 10000      # 사전 안의 단어 수
2 MY_LENGTH = 80         # 영화평 길이, 길면 자름, 짧으면 제로패딩
3 MY_EMBED = 32          # 임베딩 차원
4 MY_HIDDEN = 64         # LSTM 차원
5 MY_SAMPLE = 5          # 임의의 샘플 데이터
6 MY_EPOCHS = 10         # 반복 학습 수
7 MY_BATCH = 200         # 매번 가져오는 데이터 수
8
9 np.random.seed(111)
```

다음은 데이터를 불러옵니다. 사전 안의 단어의 수는 앞에서 설정한 10,000개로 했습니다.

```
1 (X_train, Y_train), (X_test, Y_test) = imdb.load_data(num_words=MY_WORDS)

Downloading data from
https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17465344/17464789 [=====] - 0s 0us/step
17473536/17464789 [=====] - 0s 0us/step
```

imdb 데이터셋은 다음 구조로 되어있습니다. 전체 50,000개 데이터

의 영화 평가 텍스트 데이터를 저장한 X와 긍정/부정의 레이블 y를 포함합니다.

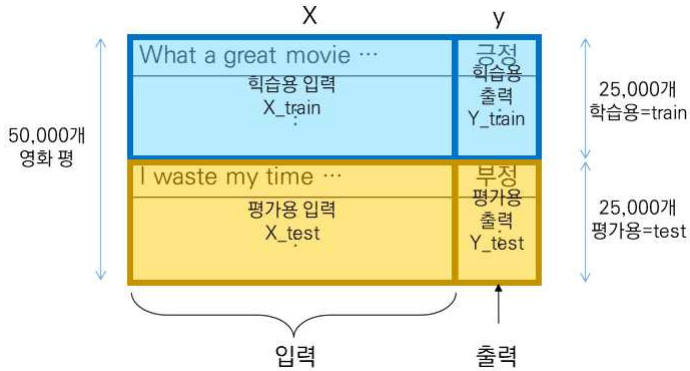


그림 60. imdb 데이터셋 구조

데이터셋의 모양을 확인해 보세요.

```
1 print("학습용 입력 데이터의 모양", X_train.shape)
2 print("학습용 출력 데이터의 모양", Y_train.shape)
3 print("평가용 입력 데이터의 모양", X_test.shape)
4 print("평가용 출력 데이터의 모양", Y_test.shape)
```

학습용 입력 데이터의 모양(25000,)

학습용 출력 데이터의 모양(25000,)

평가용 입력 데이터의 모양(25000,)

평가용 출력 데이터의 모양(25000,)

이 데이터셋에서 첫 번째 훈련데이터를 조회해 보면 다음처럼 숫자로 출력됩니다. 이 숫자들은 단어를 숫자로 매핑해 놓은 것입니다.

```
1 print(X_train[0])
```

```
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941,
4, 173, 36, 256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 480,
284, 5, 150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 172, 4536, 1111,
```

```
17, 546, 38, 13, 447, 4, 192, 50, 16, 6, 147, 2025, 19, 14, 22, 4,
1920, 4613, 469, 4, 22, 71, 87, 12, 16, 43, 530, 38, 76, 15, 13,
1247, 4, 22, 17, 515, 17, 12, 16, 626, 18, 2, 5, 62, 386, 12, 8,
316, 8, 106, 5, 4, 2223, 5244, 16, 480, 66, 3785, 33, 4, 130, 12,
16, 38, 619, 5, 25, 124, 51, 36, 135, 48, 25, 1415, 33, 6, 22, 12,
215, 28, 77, 52, 5, 14, 407, 16, 82, 2, 8, 4, 107, 117, 5952, 15,
256, 4, 2, 7, 3766, 5, 723, 36, 71, 43, 530, 476, 26, 400, 317, 46,
7, 4, 2, 1029, 13, 104, 88, 4, 381, 15, 297, 98, 32, 2071, 56, 26,
141, 6, 194, 7486, 18, 4, 226, 22, 21, 134, 476, 26, 480, 5, 144,
30, 5535, 18, 51, 36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38,
1334, 88, 12, 16, 283, 5, 16, 4472, 113, 103, 32, 15, 16, 5345, 19,
178, 32]
```

다음은 첫 번째 데이터에 사용된 단어의 총수와 몇 개의 단어가 사용됐는지를 확인해 봅니다. 첫 번째 데이터는 총 218개 단어가 있으며 중복을 제외하고 120개의 단어가 사용되었습니다.

```
1 print(len(X_train[0]), len(set(X_train[0])))
218 120
```

첫 번째 데이터의 y 를 출력해 보겠습니다. 결과 1은 영화평이 긍정에 해당한다는 의미입니다. 만일 y 값이 0이라면 부정을 의미합니다.

```
1 print(Y_train[0])
1
```

앞에서 X 를 출력하면 숫자로 출력됩니다. 이 데이터가 어떤 텍스트로 구성되어 있는지 알려면 단어와 단어의 인덱스를 저장한 딕셔너리를 불러와야 합니다. `imdb.get_word_index()`는 단어와 단어의 인덱스를 저장한 데이터를 불러옵니다.

```
1 word_to_idx = imdb.get_word_index()
```

```
Downloading data from
https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb_
word_index.json
1646592/1641221 [=====] - 0s 0us/step
1654784/1641221 [=====] - 0s 0us/step
```

단어 'the'의 인덱스를 확인해 봅니다.

```
1 print(word_to_idx['the'])
1
```

단어의 인덱스 순서는 가장 많이 나오는 단어부터 순서대로 인덱스를 부여합니다. 아마도 'the'가 가장 많이 사용되는 단어인가 봅니다.

반대로 인덱스를 이용해서 단어를 찾아보겠습니다.

```
1 idx_to_word = {}
2 for key, val in word_to_idx.items():
3     idx_to_word[val] = key
```

위 코드는 다음처럼 표현할 수 있습니다.

```
1 idx_to_word = dict([(value, key) for (key, value) in word_to_idx.items()])
```

실제 1번 인덱스가 'the'인지 출력해 봅니다.

```
1 print(idx_to_word[1])
the
```

다음 코드는 인덱스 영화평을 단어 영화평으로 바꿉니다.

```
1 def decoding(review_num):
2     decoded = []
3     for i in review_num:
```

```

4         word = idx_to_word[i]
5         decoded.append(word)
6         return decoded
7
8     print(decoding(X_train[0]))

```

```

['the', 'as', 'you', 'with', 'out', 'themselves', 'powerful',
'lets', 'loves', 'their', 'becomes', 'reaching', 'had', ...

```

그런데 첫 번째 영화평의 단어 배열이 문법에 맞지 않아 보입니다. 이렇게 출력되는 이유는 imdb 데이터는 특수 단어 <PAD>, <START>, <UNK>, <UNUSED>를 포함하기 때문입니다. 이들 텍스트는 인덱스로 지정되어 있지 않기 때문입니다. 그러므로 이들 텍스트를 각각 0부터 3까지 할당하고 이전의 텍스트를 3만큼 미뤄줘야 합니다. 다음 코드는 모든 단어의 인덱스를 3 증가시키고 단어 4개를 추가합니다.

```

1 idx_to_word = {k+3 :v for k,v in idx_to_word.items()}
2 idx_to_word[0] = "<PAD>"
3 idx_to_word[1] = "<START>"
4 idx_to_word[2] = "<UNK>" # unknown
5 idx_to_word[3] = "<UNUSED>"

```

```

['<START>', 'this', 'film', 'was', 'just', 'brilliant', 'casting',
'location', 'scenery', 'story', 'direction', "everyone's", ...

```

첫 번째 데이터의 영화평 텍스트를 확인했습니다. 다음은 영화평의 길이를 모두 같게 만들어야 합니다. 먼저 영화평의 길이를 같게 하지 전의 영화평의 길이를 확인해 봅니다.

```

1 def show_length():
2     print('영화평의 길이')
3     for i in range(10):
4         print('영화평', i, ':', len(X_train[i]))
5
6     show_length()

```



```
영화평 0 : 218
영화평 1 : 189
영화평 2 : 141
영화평 3 : 550
영화평 4 : 147
영화평 5 : 43
영화평 6 : 123
영화평 7 : 562
영화평 8 : 233
영화평 9 : 130
```

영화평의 길이가 모두 다른 것을 확인할 수 있습니다. 이 데이터는 2차원 데이터라고 할 수 없으므로 2차원 데이터로 만들기 위해 모든 영화평의 길이를 같게 해야 합니다. 이때 사용하는 함수가 `pad_sequences()`입니다. `maxlen` 매개변수는 최대 길이이며, `truncating`은 원본데이터가 길 때 자를 위치이며, `padding`은 원본데이터가 짧을 때 0을 채울 위치입니다.

```
1 X_train = pad_sequences(X_train, maxlen=MY_LENGTH,
2                           truncating='post', padding='post')
3 X_test = pad_sequences(X_test, maxlen=MY_LENGTH,
4                          truncating='post', padding='post')
```

다시 영화평의 길이를 확인해 보면 모두 같은 길이로 바뀐 것을 확인할 수 있습니다.

```
1 show_length()
영화평 0 : 80
영화평 1 : 80
영화평 2 : 80
영화평 3 : 80
영화평 4 : 80
```

```
}
```

영화평 5 : 80
 영화평 6 : 80
 영화평 7 : 80
 영화평 8 : 80
 영화평 9 : 80

이 데이터를 이용해서 영화평 분류를 만드시 케라스를 이용하란 법은 없습니다. 싸이킷런(sklearn)의 MLPClassifier를 이용해서도 모형을 만들 수 있습니다. 이것은 RNN을 사용하지 않았으므로 권하지는 않습니다. 은닉층의 수와 뉴런의 수는 (100, 50, 30)으로 설정했습니다.

```
1 from sklearn.neural_network import MLPClassifier
2 mlp = MLPClassifier(hidden_layer_sizes=(100, 50, 30))
3 mlp.fit(X_train, Y_train)
```

```
MLPClassifier(hidden_layer_sizes=(100, 50, 30))
```

모형의 정확도를 확인해 보지만 결과가 그리 좋지는 않습니다.

```
1 mlp.score(X_test, Y_test)
```

```
0.51136
```

다음은 RNN 구조입니다. 80개의 단어는 LSTM 신경망의 입력으로 전달되어 긍정/부정일 확률 출력됩니다.

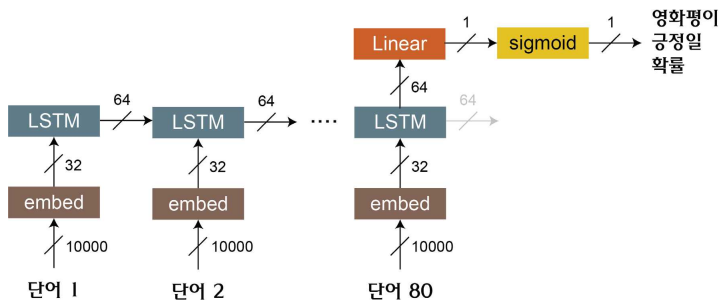


그림 61. 영화평 분류를 위한 RNN 구조

앞에서 사용할 단어의 수를 10,000개로 지정했었습니다. 그래서 각 단어는 원-핫 인코딩되어 10,000개의 이진 데이터가 임베딩(embed) 층으로 전달되어 32개로 줄어듭니다. 이 데이터가 LSTM으로 전달됩니다. 각 단어의 LSTM 출력은 다음 단어의 embed 출력과 결합하여 마지막 단어까지 전달됩니다. 이 결과는 최종적으로 시그모이드(sigmoid) 활성화 함수를 통해 출력됩니다.

이제 위 코드를 케라스를 이용해서 구현해 보겠습니다.

다음은 임베딩 층과 LSTM 층을 결합한 네트워크를 정의합니다. 임베딩 층의 입력은 사용할 단어의 수 10,000개(MY_WORDS), 출력은 임베딩 차원 32개(MY_EMBED), 입력 길이는 영화평의 길이 80(MY_LENGTH)으로 지정합니다. LSTM 층을 추가하기 전에 모델을 확인해 보세요.

```
1 model = Sequential()
2 model.add(Embedding(input_dim=MY_WORDS, output_dim=MY_EMBED,
                      input_length=MY_LENGTH))
3 model.summary()
```

Model: "sequential"

Layer(type)	Output Shape	Param #
embedding(Embedding)	(None, 80, 32)	320000

Total params: 320,000

Trainable params: 320,000

Non-trainable params: 0

이제 앞의 코드를 수정해서 LSTM 층을 추가하세요. 그리고 시그모이

드 활성화 함수를 사용하는 출력층을 추가하세요.

```
1 model = Sequential()
2 model.add(Embedding(input_dim=MY_WORDS, output_dim=MY_EMBED,
3                     input_length=MY_LENGTH))
4
5 model.add(LSTM(units=MY_HIDDEN,
6               input_shape=(MY_LENGTH, MY_EMBED)))
7
8 model.add(Dense(units=1, activation='sigmoid'))
9
10 model.summary()
```

Model: "sequential_1"

Layer(type)	Output Shape	Param #
embedding_1(Embedding)	(None, 80, 32)	320000
lstm(LSTM)	(None, 64)	24832
dense(Dense)	(None, 1)	65
Total params: 344,897		
Trainable params: 344,897		
Non-trainable params: 0		

이제 인공신경망 훈련을 정의합니다. 손실함수는 크로스 엔트로피, 옵티마이저는 Adam을 사용합니다. 모델의 평가방법은 정확도 (accuracy)입니다.

```
1 model.compile(loss='binary_crossentropy', optimizer='adam',
2               metrics='acc')
```

이제 학습용 데이터를 이용해서 인공신경망을 학습시킵니다. 학습횟수는 10번(MY_EPOCHS), 배치 크기는 200번(MY_BATCH)으로 지정합니다.

```
1 begin = time()
2 model.fit(X_train, Y_train,
3           epochs=MY_EPOCHS, batch_size=MY_BATCH)
4 end = time()
5 print("학습시간: ",(end-begin))
```

```
Epoch 1/10
125/125 [=====] - 21s 150ms/step - loss:
0.5546 - acc: 0.6888
Epoch 2/10
125/125 [=====] - 19s 149ms/step - loss:
0.3484 - acc: 0.8529
Epoch 3/10
125/125 [=====] - 19s 148ms/step - loss:
0.2859 - acc: 0.8849
Epoch 4/10
125/125 [=====] - 18s 148ms/step - loss:
0.2520 - acc: 0.9044
Epoch 5/10
125/125 [=====] - 18s 148ms/step - loss:
0.2156 - acc: 0.9182
Epoch 6/10
125/125 [=====] - 18s 148ms/step - loss:
0.1880 - acc: 0.9311
Epoch 7/10
125/125 [=====] - 19s 150ms/step - loss:
0.1673 - acc: 0.9384
Epoch 8/10
125/125 [=====] - 19s 149ms/step - loss:
0.1446 - acc: 0.9470
Epoch 9/10
```

```
125/125 [=====] - 19s 149ms/step - loss:
0.1305 - acc: 0.9540
Epoch 10/10
125/125 [=====] - 19s 149ms/step - loss:
0.1163 - acc: 0.9581
학습시간: 203.84211421012878
```

학습을 더 빠르게 하려면 실행 전 메뉴에서 [수정] -> [노트 설정]에서 [하드웨어 가속기]를 [GPU]로 설정하세요. 그러면 더 빨리 학습됩니다. 하드웨어 가속기를 바꾸면 모델을 처음부터 다시 실행시켜야 합니다.

위 결과에서의 정확도 95%는 사실과 다를 수 있습니다. 올바른 평가를 위해서는 평가용 데이터를 이용해서 평가해야 합니다.

```
1 score = model.evaluate(x=X_test, y=Y_test, verbose=1)
2 print(f"최종 정확도: {score[1]:.2f}")
```

```
782/782 [=====] - 11s 14ms/step - loss:
0.9537 - acc: 0.7637
최종 정확도: 0.76
```

테스트 데이터를 이용해서 긍정/부정을 예측해 보겠습니다.

```
1 pred = model.predict(X_test)
2 print(pred)
```

```
[[0.0084601 ]
 [0.99833375]
 [0.99829966]
 ...
 [0.00407651]
 [0.2914366 ]
 [0.9975072 ]]
```

앞의 결과대로라면 테스트 데이터의 첫 번째 영화평은 부정에 가깝고 두 번째 영화평은 긍정에 가깝다는 것을 확인할 수 있습니다. 이제 실제 정답 비교하기 위해 혼동 행렬(confusion matrix)을 출력해 보겠습니다. 그러기 위해서 pred가 0.5보다 크면 1, 그렇지 않으면 0으로 바꾼 후 혼동 행렬을 출력합니다.

```
1 pred =(pred > 0.5)
2 print('혼동 행렬')
3 print(confusion_matrix(Y_test, pred))
```

혼동 행렬

[[9627 2873]

[3034 9466]]

여러분의 실행결과는 이 설명서와 다를 수 있습니다. 위 결과에서 9627은 TN(True Negative)의 수입입니다. 이것은 실제 영화를 부정이라고 예측한 것 중에서 실제 부정인 것의 수입입니다. 9466은 영화를 긍정이라고 예측한 것 중에서 실제 긍정인 것의 수입입니다.

이제 imdb.com에서 영화 ‘오징어 게임’의 리뷰 하나를 이용해서 이 리뷰가 긍정을 의미하는지 부정을 의미하는지 알아보겠습니다.

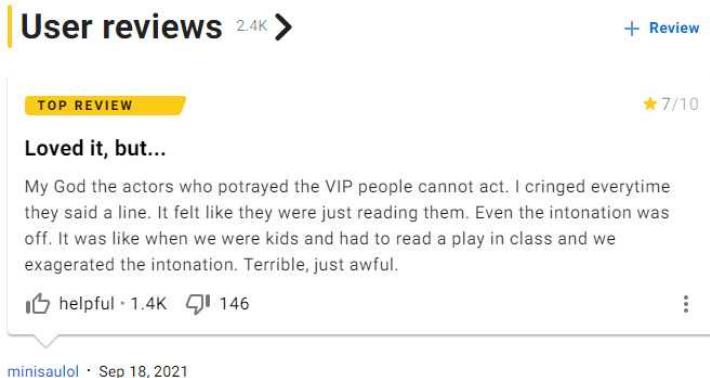


그림 62. 영화 리뷰 예

수집된 데이터는 다음과 같습니다.

```
1 text = "My God the actors who portrayed the VIP people cannot  
act. I cringed everytime they said a line. It felt like they  
were just reading them. Even the intonation was off. It was  
like when we were kids and had to read a play in class and we  
exagerated the intonation. Terrible, just awful."
```

다음 코드는 수집한 데이터를 모두 소문자로 바꾼 후 문자열을 공백으로 분리합니다.

```
1 input_text = text.lower().split()  
2 print(input_text)  
[ 'my', 'god', 'the', 'actors', 'who', 'potrayed', 'the', 'vip',  
'people', 'cannot', 'act.', 'i', 'cringed', 'everytime', 'they',  
'said', 'a', 'line.', 'it', 'felt', 'like', 'they', 'were',  
'just', 'reading', 'them.', 'even', 'the', 'intonation', 'was',  
'off.', 'it', 'was', 'like', 'when', 'we', 'were', 'kids', 'and',  
'had', 'to', 'read', 'a', 'play', 'in', 'class', 'and', 'we',  
'exagerated', 'the', 'intonation.', 'terrible,', 'just', 'awful.']
```

다음 코드는 단어-인덱스 딕셔너리에 특수 단어 <PAD>, <START>, <UNK>, <UNUSED>를 포함합니다.

```
1 word_to_idx = imdb.get_word_index()  
2 word_to_idx = {k:(v+3) for k,v in word_to_idx.items()}  
3 word_to_idx["<PAD>"] = 0  
4 word_to_idx["<START>"] = 1  
5 word_to_idx["<UNK>"] = 2 # unknown  
6 word_to_idx["<UNUSED>"] = 3
```

다음은 단어를 숫자로 변환합니다. word_to_idx에 없는 단어는 2(<UNK>)로 지정하며, 인덱스가 10000보다 크면 3(<UNUSED>)로

지정합니다.

```
1 def encoding(review_text):
2     encoded = []
3     for word in review_text:
4         try :
5             idx = word_to_idx[word]
6             if idx>10000 :
7                 encoded.append(3)
8             else :
9                 encoded.append(idx)
10        except :
11            encoded.append(2)
12    return encoded
13
14 input_encoded =encoding(input_text)
15 print(input_encoded)
```

```
[61, 558, 4, 156, 37, 2, 4, 3, 84, 566, 2, 13, 3, 3, 36, 301, 6, 2,
12, 421, 40, 36, 71, 43, 886, 2, 60, 4, 3, 16, 2, 12, 16, 40, 54,
75, 71, 362, 5, 69, 8, 332, 6, 297, 11, 707, 5, 75, 3, 4, 2, 2, 43,
2]
```

인덱스로 변환된 데이터를 길이 80인 배열로 변환합니다.

```
1 input_pad = pad_sequences(
2     np.array(input_encoded)[np.newaxis, :],
3     maxlen=MY_LENGTH, truncating='post', padding='post')
```

예측합니다. 실제 데이터에서 영화평의 점수는 7점이었지만 이 예측의 결과는 부정으로 판별됩니다.

```
1 model.predict(input_pad)
array([[0.01169005]], dtype=float32)
```

정확도를 올리려면 사용할 단어의 수를 더 늘려보거나 모델을 더 개

선택할 필요가 있습니다. 그러나 이 결과에 큰 의미를 두지 마세요. 이러한 과정으로 영화평 분석을 수행하는지에 대해서 알아보시길 바랍니다.

● 앞의 모형에서 아래의 내용을 바꿔보고 그 결과를 기록해 보세요.

- ① 최적화 함수를 SGD로 바꿔보세요.
- ② F1 점수의 임계치를 0.2로 바꿔보세요.
- ③ LSTM 대신 SimpleRNN을 사용해 보세요.
- ④ MY_EMBED 파라미터를 8로 바꿔보세요.
- ⑤ MY_WORDS 파라미터를 1,000으로 바꿔보세요.
- ⑥ MY_LENGTH 파라미터를 1600으로 바꿔보세요.
- ⑦ pad_sequences()의 매개변수를 'post'에서 'pre'로 바꿔보세요.

3절. 로이터 기사 분류하기

3.1. 로이터 데이터셋

로이터는 통신사의 하나입니다. 독일의 파울 올리우스 로이터(1851, Paul Reuter)가 설립한 영국의 뉴스 및 정보제공기업입니다. 현재는 신문, 방송 등에 뉴스를 공급하는 전통적인 통신사의 기능보다 증시 상황 속보 등 금융정보 제공 등의 비중이 커져 있습니다.



그림 63. 로이터 본사

[https://commons.wikimedia.org/wiki/File:Thomson_Reuters_building,_30_South_Colonnade,_Canary_Wharf_in_London,_June_2013_\(2\).JPG](https://commons.wikimedia.org/wiki/File:Thomson_Reuters_building,_30_South_Colonnade,_Canary_Wharf_in_London,_June_2013_(2).JPG)

우리가 사용할 로이터 데이터셋은 11,228개 기사, 46개 주제를 포함합니다. 다음은 46개 주제의 글의 수입니다.

표 3. 로이터 데이터셋 주제

번호	주제	글 수	번호	주제	글 수
0	cocoa	50	23	alum	36
1	grain	378	24	oilseed	56
2	veg-oil	66	25	gold	77
3	earn	2769	26	tin	18
4	acq	1701	27	strategic-metal	13
5	wheat	14	28	livestock	43
6	copper	39	29	retail	19
7	housing	15	30	ipi	38
8	money-supply	126	31	iron-steel	34
9	coffee	93	32	rubber	30
10	sugar	114	33	heat	9
11	trade	337	34	jobs	43
12	reserves	40	35	lei	10
13	ship	149	36	bop	46
14	cotton	18	37	zinc	17
15	carcass	19	38	orange	16
16	crude	387	39	pet-chem	20
17	nat-gas	33	40	dlr	32
18	cpi	59	41	gas	28
19	money-fx	475	42	silver	10
20	interest	238	43	wpi	19
21	gnp	91	44	hog	10
22	meal-feed	10	45	lead	14

- 위 표의 CSV 파일은 <http://javaspecialist.co.kr/board/1076>에서 내려받을 수 있습니다.

3.2. RNN으로 로이터 기사 자동 분류하기

RNN으로 로이터 기사를 자동으로 분류하는 모델을 만들겠습니다. 먼저 필요한 라이브러리를 가져옵니다. `to_categorical()` 함수는 텐서플로우에 있는 것을 확인하세요.

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 from keras.models import Sequential
5 from keras.layers import Dense, Dropout, Activation
6 from keras.datasets import reuters
7
8 from keras.preprocessing.text import Tokenizer
9 from tensorflow.keras.utils import to_categorical
```

다음은 하이퍼 파라미터를 설정합니다.

```
1 MY_SAMPLE = 2947
2 NUM_CLASS = 46          # 로이터 기사 데이터의 총 카테고리 수
3 MY_NUM_WORDS = 2000     # 사전 안의 단어의 수
4 MY_EPOCH = 10           # 학습횟수
5 MY_BATCH = 64           # 매번 가져와 처리할 데이터의 수
6 MY_HIDDEN = 512         # 은닉층의 뉴런 수
7 MY_DROPOUT = 0.5
```

다음은 레이블들을 정의했습니다. 이 코드를 모두 작성하기 번거롭다면 <http://jasvaspecialist.co.kr/board/1076>에서 내려받으세요.

```
1 labels = ['cocoa', 'grain', 'veg-oil', 'earn', 'acq', 'wheat',
            'copper', 'housing', 'money-supply', 'coffee', 'sugar',
            'trade', 'reserves', 'ship', 'cotton', 'carcass',
            'crude', 'nat-gas', 'cpi', 'money-fx', 'interest',
```

```
'gnp','meal-feed','alum','oilseed','gold','tin',  
'strategic-metal','livestock','retail','ipi',  
'iron-steel','rubber','heat','jobs','lei','bop',  
'zinc','orange','pet-chem','dlr','gas','silver',  
'wpi','hog','lead']
```

다음은 로이터 데이터셋을 불러옵니다. `test_split=0.3`은 불러올 때 평가데이터의 크기를 30%로 설정해서 불러옵니다.

```
1 (X_train, Y_train), (X_test, Y_test) = reuters.load_data(  
    num_words=MY_NUM_WORDS, test_split=0.3)
```

불러온 데이터셋의 모양을 확인해 보면 다음과 같습니다.

```
1 def show_shape():  
2     print('\n== DB SHAPE INFO ==')  
3     print('X_train shape = ', X_train.shape)  
4     print('X_test shape = ', X_test.shape)  
5     print('Y_train shape = ', Y_train.shape)  
6     print('Y_test shape = ', Y_test.shape)  
7  
8 show_shape()  
  
== DB SHAPE INFO ==  
X_train shape =(7859,)  
X_test shape =(3369,)  
Y_train shape =(7859,)  
Y_test shape =(3369,)
```

다음 코드는 레이블별로 데이터의 개수가 몇 개인지 출력해 봅니다. 넘파이의 `unique()` 함수에 `return_counts=True` 매개변수를 추가하면 넘파이 배열에서 유일하게 구분되는 값들의 개수를 제공합니다.

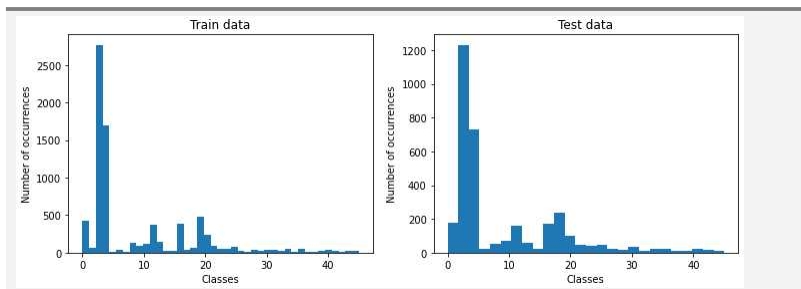
```
1 unique, counts = np.unique(Y_train, return_counts=True)
2 for i in range(len(unique)):
3     print(unique[i], labels[i], '=', counts[i])
```

```
0 cocoa = 50
1 grain = 378
2 veg-oil = 66
3 earn = 2769
4 acq = 1701
5 wheat = 14
6 copper = 39
7 housing = 15
8 money-supply = 126
9 coffee = 93
10 sugar = 114
11 trade = 337
12 reserves = 40
13 ship = 149
14 cotton = 18
15 carcass = 19
16 crude = 387
17 nat-gas = 33
18 cpi = 59
19 money-fx = 475
20 interest = 238
21 gnp = 91
22 meal-feed = 10
23 alum = 36
24 oilseed = 56
25 gold = 77
26 tin = 18
27 strategic-metal = 13
28 livestock = 43
29 retail = 19
```

```
30 ipi = 38
31 iron-steel = 34
32 rubber = 30
33 heat = 9
34 jobs = 43
35 lei = 10
36 bop = 46
37 zinc = 17
38 orange = 16
39 pet-chem = 20
40 dlr = 32
41 gas = 28
42 silver = 10
43 wpi = 19
44 hog = 10
45 lead = 14
```

다음 코드는 학습용 데이터와 평가용 데이터의 클래스별 개수를 시각화하기 위해서 히스토그램 그래프를 그립니다.

```
1 plt.figure(figsize=(12, 4))
2 plt.subplot(121)
3 plt.hist(Y_train, bins='auto')
4 plt.xlabel("Classes")
5 plt.ylabel("Number of occurrences")
6 plt.title("Train data")
7
8 plt.subplot(122)
9 plt.hist(Y_test, bins='auto')
10 plt.xlabel("Classes")
11 plt.ylabel("Number of occurrences")
12 plt.title("Test data")
13 plt.show()
```



샘플 기사(2947번 기사)의 카테고리과 단어의 수, 그리고 실제 데이터를 출력해 봅니다.

```
1 print("article #", MY_SAMPLE)
2 print("category", Y_train[MY_SAMPLE], labels[Y_train[MY_SAMPLE]])
3 print("number of words", len(X_train[MY_SAMPLE]))
4 print(X_train[MY_SAMPLE])
```

```
article # 2947
category 4 acq
number of words 61
[1, 2, 1229, 81, 8, 16, 515, 25, 270, 5, 4, 2, 1229, 111, 267, 7,
73, 2, 2, 7, 108, 13, 80, 1448, 28, 365, 12, 11, 15, 1986, 2, 69,
158, 18, 1296, 1275, 7, 2, 1627, 2, 2, 4, 393, 374, 1229, 323, 5, 2,
1229, 7, 2, 9, 25, 2, 473, 936, 4, 49, 8, 17, 12]
```

단어-인덱스 정보를 불러옵니다. 로이터 기사에는 전체 30,980개 단어를 포함하며, 단어 'the'의 인덱스는 1입니다.

```
1 word_to_idx = reuters.get_word_index()
2
3 print(f"딕셔너리에 {len(word_to_idx)+1}개 단어가 있음")
4 print(f"단어 'the'의 인덱스: {word_to_idx['the']}")
```

```
Downloading data from
https://storage.googleapis.com/tensorflow/tf-keras-datasets/reuters_word_index.json
```

```
557056/550378 [=====] - 0s 0us/step
565248/550378 [=====] - 0s 0us/step
딕셔너리에 30980개 단어가 있음
단어 'the'의 인덱스: 1
```

다음 코드는 단어-인덱스 딕셔너리를 인덱스-단어 딕셔너리로 만듭니다.

```
1 idx_to_word = {}
2 for word, index in word_to_idx.items():
3     idx_to_word[index] = word
```

다음 코드는 인덱스를 문자로 디코딩합니다. `get()` 함수의 두 번째 인수는 해당 인덱스가 없을 때 반환하는 기본값입니다.

```
1 def decoding():
2     decoded = []
3     for i in X_train[MY_SAMPLE]:
4         word = idx_to_word.get(i-3, "??")
5         decoded.append(word)
6     # print(' '.join(decoded))
7     return decoded
8
9 print(decoding())
```

```
['??', '??', 'telephone', 'corp', 'said', 'it', 'completed',
'its', 'acquisition', 'of', 'the', '??', 'telephone', 'co',
'based', 'in', 'new', '??', '??', 'in', 'exchange', 'for',
'stock', 'valued', 'at', '26', '3', 'mln', 'dlrs', 'enterprises',
'??', 'about', '16', '000', 'access', 'lines', 'in', '??',
'county', '??', '??', 'the', 'third', 'operating', 'telephone',
'subsidiary', 'of', '??', 'telephone', 'in', '??', 'and', 'its',
'??', 'largest', 'overall', 'the', 'company', 'said', 'reuter',
'3']
```

다음은 학습용 데이터와 평가용 데이터의 X를 단어의 수를 세서 행렬로 만듭니다. 사용할 단어는 2,000개(MY_NUM_WORDS)입니다. `sequence_to_matrix()` 함수는 문장(시퀀스, `sequence`)들을 행렬(`matrix`)로 변환합니다. 각 문장은 하나의 벡터로 변환되는데 벡터의 숫자는 사전의 각 단어의 사용 빈도수를 의미합니다. 이를 이용하면 입력 문장들의 길이가 달라도 변환 후 길이는 같게 됩니다. 그리고 각 문장은 고유의 수를 갖게 됩니다.

```
1 tok = Tokenizer(num_words=MY_NUM_WORDS)
2 X_train_tok = tok.sequences_to_matrix(X_train, mode='count')
3 X_test_tok = tok.sequences_to_matrix(X_test, mode='count')
```

토큰화한 데이터와 배열의 크기 그리고 총합을 출력해 봅니다. 결과에서 숫자는 지정한 2,000개 단어가 이 문서에서 사용된 빈도수입니다.

```
1 sample = X_train_tok[MY_SAMPLE]
2 print(*sample, sep=' ') # *는 언패킹(unpacking)을 의미함
3 print("Array size:", len(sample))
4 print("Sum:", np.sum(sample))
```

```
0.0 1.0 11.0 0.0 3.0 2.0 0.0 4.0 2.0 1.0 0.0 1.0 2.0 1.0 0.0 1.0 ...
Array size: 2000
Sum: 61.0
```

이번에는 종속변수 Y를 `to_categorical()` 함수를 이용해서 원-핫 인코딩하지 않겠습니다. 종속변수를 원-핫 인코딩하지 않으면 케라스 인공신경망 모형의 손실함수에 “`sparse_categorical_crossentropy`”를 사용하면 됩니다.

이제 인공신경망 구조를 정의합니다. 은닉층 1개와 출력층 1개를 갖는 구조를 정의합니다. 아래에 정의한 모델은 RNN 구조는 아닙니다. 단순히 은닉층만 갖는 DNN 구조입니다.

```

1 model = Sequential()
2 model.add(Dense(MY_HIDDEN, input_shape=(MY_NUM_WORDS,)))
3 model.add(Activation("relu"))
4 model.add(Dropout(MY_DROPOUT))
5 model.add(Dense(NUM_CLASS))
6 model.add(Activation("softmax"))
7 model.summary()

```

Model: "sequential"

Layer(type)	Output Shape	Param #
dense(Dense)	(None, 512)	1024512
activation(Activation)	(None, 512)	0
dropout(Dropout)	(None, 512)	0
dense_1(Dense)	(None, 46)	23598
activation_1(Activation)	(None, 46)	0

Total params: 1,048,110

Trainable params: 1,048,110

Non-trainable params: 0

다음은 훈련을 정의합니다. 손실함수는 크로스 엔트로피, 옵티마이저는 adam을 사용합니다.

```

1 model.compile(loss="sparse_categorical_crossentropy",
                optimizer="adam", metrics=["accuracy"])

```

다음 코드는 모델을 학습시킵니다. 학습횟수는 10회(MY_EPOCH)이고, 배치 크기는 64(MY_BATCH)입니다.

```
1 model.fit(X_train_tok, Y_train,  
            validation_data=(X_test_tok, Y_test),  
            epochs=MY_EPOCH, batch_size=MY_BATCH, verbose=1)
```

```
Epoch 1/10  
123/123 [=====] - 3s 16ms/step - loss:  
1.6360 - accuracy: 0.6679 - val_loss: 1.1010 - val_accuracy:  
0.7697  
Epoch 2/10  
123/123 [=====] - 1s 10ms/step - loss:  
0.8314 - accuracy: 0.8207 - val_loss: 0.9301 - val_accuracy:  
0.7988  
Epoch 3/10  
123/123 [=====] - 1s 9ms/step - loss:  
0.5719 - accuracy: 0.8660 - val_loss: 0.8932 - val_accuracy:  
0.8106  
Epoch 4/10  
123/123 [=====] - 1s 9ms/step - loss:  
0.4208 - accuracy: 0.9002 - val_loss: 0.8879 - val_accuracy:  
0.8127  
Epoch 5/10  
123/123 [=====] - 1s 9ms/step - loss:  
0.3321 - accuracy: 0.9187 - val_loss: 0.8984 - val_accuracy:  
0.8151  
Epoch 6/10  
123/123 [=====] - 1s 8ms/step - loss:  
0.2992 - accuracy: 0.9307 - val_loss: 0.9131 - val_accuracy:  
0.8136  
Epoch 7/10  
123/123 [=====] - 1s 11ms/step - loss:  
0.2410 - accuracy: 0.9375 - val_loss: 0.9490 - val_accuracy:
```

}

```

0.8127
Epoch 8/10
123/123 [=====] - 1s 10ms/step - loss:
0.2281 - accuracy: 0.9459 - val_loss: 1.0000 - val_accuracy:
0.8053
Epoch 9/10
123/123 [=====] - 1s 8ms/step - loss:
0.2135 - accuracy: 0.9469 - val_loss: 1.0204 - val_accuracy:
0.8080
Epoch 10/10
123/123 [=====] - 1s 8ms/step - loss:
0.1964 - accuracy: 0.9467 - val_loss: 1.0497 - val_accuracy:
0.8118
<keras.callbacks.History at 0x7f9c43c3af50>

```

학습 후 샘플 데이터를 이용해서 예측해 봅니다.

```

1 sample = X_train_tok[MY_SAMPLE]
2 sample = sample.reshape(1, sample.shape[0])
3 pred = model.predict(sample, verbose=0)
4 guess = np.argmax(pred)
5 answer = Y_train[MY_SAMPLE]
6 print("샘플 기사의 예측값:", guess, labels[guess])
7 print("실제 정답:", answer, labels[answer])

```

샘플 기사의 예측값: 4 acq

실제 정답: 4 acq

모델을 평가해 봅니다.

```

1 model.evaluate(X_test_tok, Y_test)
106/106 [=====] - 0s 3ms/step - loss:
1.0169 - accuracy: 0.8106
[1.0169134140014648, 0.8106263279914856]

```

모델을 저장하려면 `save()` 함수를 이용하세요. 저장되는 모델의 형식은 H5(HDF5)입니다.

```
1 model.save('reuter.h5')
```

단순히 DNN으로 인공신경망을 구성한다면 머신러닝 라이브러리를 이용해도 됩니다. 다음 코드는 싸이킷런 패키지의 `MLPClassifier`를 이용해서 모델을 만들고 훈련한 후 샘플 데이터로 예측해 봅니다.

```
1 from sklearn.neural_network import MLPClassifier
2 mlp = MLPClassifier(hidden_layer_sizes=(MY_HIDDEN,),
                      solver="adam", activation="relu",
                      max_iter=10)
3 mlp.fit(X_train_tok, Y_train)
MLPClassifier(hidden_layer_sizes=(512,), max_iter=10)
```

예측한 결과는 앞에서 케라스를 이용한 결과와 같습니다.

```
1 mlp.predict(sample)
array([4])
```

평가데이터를 이용해 평가한 결과도 케라스의 결과와 크게 다르지 않습니다.

```
1 mlp.score(X_test_tok, Y_test)
0.8011279311368359
```

케라스의 모형과 다른 점은 드롭아웃을 사용하지 않은 것과 가중치의 초기값입니다. `MLPClassifier`는 드롭아웃과 가중치 초기값을 변경할 수 없습니다.

4절. 영-한 번역기

4.1. 번역기

1) 번역기의 한계

2021년 12월의 구글 번역기와 네이버의 파파고 번역기가 번역한 내용입니다. 일부 문장의 경우 실제와는 많이 다른 결과로 번역하기도 합니다.

한글	구글 번역기	네이버 파파고
영문을 모르겠어.	I don't know English.	I don't know English.
씨가 말랐다.	seeds are dry	The seeds are dry.
어딜 도망가?	Where are you running?	Where are you going?
뭐하니?	what are you doing?	What are you doing?
전하~ 통촉하여 주시옵소서.	Please, please contact me.	Your Highness, please give us a lot of pressure.

이 번역 결과가 아직 만족할만한 수준은 아니겠지만 인공지능망의 발전과 더불어 번역기의 발전성도 높으리라 생각됩니다. 번역 기술이 더 발전하면 위의 번역보다 더 발전된 번역 기능으로 사용할 수 있습니다.

4.2. 영-한 번역기

영어 단어를 한국어로 번역하는 예를 들어보겠습니다. 물론 구글 번역기만큼의 성능을 내는 번역기를 원하시는 것은無理입니다. 여기에서 입력은 4글자 영어 단어이며, 출력은 2글자 한글 단어를 사용합니다.

사용할 단어의 예로 다음을 들 수 있습니다.

- word: 단어, wood: 나무, game: 놀이, girl: 소녀, kiss: 키스,
love: 사랑 등등

우리는 영-한 번역기에 Seq2Seq 방식의 번역을 사용할 예정입니다.
이를 이용하면 똑똑한 번역기를 만들 수 있습니다. 예를 들면 ‘love’
만 ‘사랑’으로 번역하지 않고 ‘love’ 단어의 문자가 뒤바뀐 것 예를
들면 ‘lvoe’, ‘loev’ 등도 ‘사랑’으로 번역이 가능하게 합니다.

Seq2Seq으로 데이터를 처리하기 위해 다음 과정을 따릅니다.

- 사전 만들기
- 단어 전환

실제 우리가 사용할 영어와 한글 단어는 translate.csv 파일의 데이터를
이용합니다. 그리고 한글 문자들의 정보는 korean.csv 파일에 저장된
데이터를 사용합니다.

이 두 파일은 <http://javaspecialist.co.kr/board/1077>에서 내려받을 수 있습니다.

1) 사전 만들기

다용한 단어를 이용하여 사전을 만드는 과정이 있어야 합니다. 사전
만들기는 다음 단계를 따릅니다.

- 알파벳 만들기
- 각 글자 떼어내기
- 글자를 순서로 전환하는 사전 만들기

알파벳은 맨 앞에 특수문자 SEP를 붙이고 영어와 한글에 사용된 문자들을 이용해서 알파벳 문자열을 만듭니다. 그러면 다음처럼 만들어 집니다.

‘SEPabcdefghijklmnopqrstuvwxyz단어나무놀이소녀키스사랑’

S는 Start의 약어이며, 디코더의 입력 앞에 추가할 때 사용합니다. E는 End의 약어이며, 디코더의 출력 맨 뒤에 추가할 때 사용합니다. P는 평가할 경우 디코더의 입력으로 사용합니다. 한글은 자/모를 분리하지 않고 문자를 그대로 사용합니다. 이렇게 만들어진 알파벳은 다음과 같습니다. 실제로는 한글도 ‘가’부터 ‘흐’까지 모두 포함하도록 만들어야 합니다.

이 알파벳에서 각 글자를 떼어 놓은 다음 글자를 순서로 전환하는 딕셔너리를 만듭니다. 그러면 각 문자는 고유 인덱스를 갖게 됩니다.

```
{'S': 0, 'E': 1, 'P': 2, 'a': 3, 'b': 4, 'c': 5, 'd': 6, 'e': 7, 'f': 8, 'g': 9, 'h': 10, 'i': 11, 'j': 12, 'k': 13, 'l': 14, 'm': 15, 'n': 16, 'o': 17, 'p': 18, 'q': 19, 'r': 20, 's': 21, 't': 22, 'u': 23, 'v': 24, 'w': 25, 'x': 26, 'y': 27, 'z': 28, '단': 29, '어': 30, '나': 31, '무': 32, '놀': 33, '이': 34, '소': 35, '녀': 36, '키': 37, '스': 38, '사': 39, '랑': 40}
```

2) 단어 변환

영어 단어와 한글 단어를 숫자로 변환합니다. 만일 [‘wood’, ‘나무’] 데이터가 있다면 wood는 [25, 17, 17, 6]으로 변환하고, 나무는 [0, 31, 32]로 변환합니다.

숫자로 변환된 영어 단어는 인코더의 입력으로 사용되고, 숫자로 변환된 한글은 디코더의 입력으로 사용합니다. 한글 단어의 경우 맨 앞에

특수문자 S(=0)를 추가하기 때문에 [0, 31, 32]가 됩니다.

타켓 데이터를 만들어 디코더의 출력으로 사용하는데 나무는 특수문자 E(=1)를 뒤에 추가해서 [31, 32, 1]로 만듭니다.

이렇게 변환된 인코더 입력과 디코더 입력을 원-핫 인코딩합니다. 예를 들어 인코더의 입력 [25, 17, 16, 6]은 다음처럼 만들어집니다.

[illegible]

디코더의 입력 [0, 31, 32]는 다음처럼 만들어집니다.

[illegible]

디코더의 출력은 손실함수로 ‘sparse_categorical_crossentropy’를 사용하므로 원-핫 인코딩하지 않습니다.

4.3. 코드 구현

필요한 라이브러리를 가져오기를 합니다. 여기에서는 Sequential 클래스를 이용하지 않고 Functional API를 이용하므로 Sequential 클래스는 가져오기를 하지 않았습니다.

```
1 import numpy as np
2 import pandas as pd
3 from numpy.random import randint
4 from time import time
5 from keras.layers import Input, LSTM, Dense
6 from keras.models import Model
```

은닉층의 뉴런의 수와 학습횟수 하이퍼 파라미터를 선언합니다.

```
1 MT_HIDDEN = 128
2 MY_EPOCH = 500
```

영문과 한글 문자 셋을 이용해서 알파벳글자를 만듭니다. korean.csv 파일은 <http://javaspecialist.co.kr/board/1077>에서 내려받아 구글 코랩의 파일 디렉토리에 드래그해서 올려주세요.

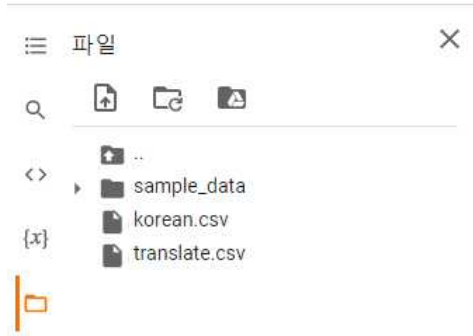


그림 64. 구글 코랩 파일 폴더

다음 코드는 영문자와 한글 문자를 연결한 알파벳/문자 데이터를 생성합니다.

```
1 arr1 = [c for c in 'SEPabcdefghijklmnopqrstuvwxyz']
2 arr2 = pd.read_csv('korean.csv', header=None)
3 arr2 = arr2[0].values.tolist()
4 num_to_char = arr1 + arr2
```

다음은 각 알파벳/문자에 인덱스를 부여합니다.

```
1 char_to_num = {n: i for i, n in enumerate(num_to_char)}
2 n_input = len(char_to_num)
3 print("글자 사전\n", char_to_num)
4 print("총 글자 수:", len(num_to_char))
5 print("영어 글자 수:", len(arr1))
6 print("한글 글자 수:", len(arr2))
```

글자 사전

```
{'S': 0, 'E': 1, 'P': 2, 'a': 3, 'b': 4, 'c': 5, 'd': 6, 'e': 7,
'f': 8, 'g': 9, 'h': 10, 'i': 11, 'j': 12, 'k': 13, 'l': 14, 'm':
15, 'n': 16, 'o': 17, 'p': 18, 'q': 19, 'r': 20, 's': 21, 't': 22,
'u': 23, 'v': 24, 'w': 25, 'x': 26, 'y': 27, 'z': 28, '가': 29,
```

'각': 30, '간': 31, '감': 32, '개': 33, '거': 34, '것': 35, '게': 36, '계': 37, '고': 38, '관': 39, '광': 40, '구': 41, '굴': 42, '규': 43, '그': 44, '금': 45, '기': 46, '깊': 47, '나': 48, '날': 49, '남': 50, '내': 51, '넙': 52, '녀': 53, '노': 54, '놀': 55, '농': 56, '높': 57, '뉴': 58, '늦': 59, '다': 60, '단': 61, '도': 62, '동': 63, '들': 64, '람': 65, '랑': 66, '래': 67, '램': 68, '류': 69, '름': 70, '륜': 71, '리': 72, '많': 73, '망': 74, '매': 75, '머': 76, '먼': 77, '명': 78, '메': 79, '명': 80, '모': 81, '목': 82, '무': 83, '물': 84, '미': 85, '바': 86, '반': 87, '방': 88, '번': 89, '복': 90, '부': 91, '분': 92, '붕': 93, '비': 94, '뿌': 95, '사': 96, '상': 97, '색': 98, '생': 99, '서': 100, '선': 101, '소': 102, '손': 103, '수': 104, '쉽': 105, '스': 106, '시': 107, '식': 108, '실': 109, '싸': 110, '아': 111, '약': 112, '얹': 113, '어': 114, '언': 115, '얼': 116, '여': 117, '연': 118, '오': 119, '옥': 120, '원': 121, '요': 122, '용': 123, '우': 124, '운': 125, '움': 126, '위': 127, '유': 128, '은': 129, '을': 130, '음': 131, '의': 132, '이': 133, '익': 134, '인': 135, '임': 136, '입': 137, '자': 138, '작': 139, '장': 140, '적': 141, '제': 142, '중': 143, '주': 144, '지': 145, '짜': 146, '쪽': 147, '찾': 148, '책': 149, '출': 150, '칙': 151, '크': 152, '키': 153, '탈': 154, '택': 155, '통': 156, '파': 157, '팔': 158, '편': 159, '피': 160, '핑': 161, '한': 162, '합': 163, '해': 164, '행': 165, '힘': 166, '회': 167, '획': 168, '휴': 169, '흐': 170}

총 글자 수: 171

영어 글자 수: 29

한글 글자 수: 142

다음은 미리 준비한 영어와 한글 쌍을 가진 파일을 불러옵니다. 이 파일도 <http://javaspecialist.co.kr/board/1077>에서 내려받아 구글 코랩의 파일 폴더에 올려놓으세요.

```
1 raw = pd.read_csv("translate.csv", header=None)
2 eng_kor = raw.values.tolist()
```

```
3 print("사전 내용\n", eng_kor)
4 print("총 단어 수:", len(eng_kor))
```

사전 내용

```
[['cold', '감기'], ['come', '오다'], ['cook', '요리'], ['copy',
'복사'], ['cost', '비용'], ['date', '날짜'], ['deal', '거래'],
['deep', '깊은'], ['desk', '책상'], ['down', '아래'], ['dust',
'먼지'], ['duty', '의무'], ['each', '각각'], ['east', '동쪽'],
['easy', '쉽다'], ['exit', '탈출'], ['face', '얼굴'], ['fact',
'사실'], ['fall', '가을'], ['farm', '농장'], ['feet', '다리'],
['find', '찾다'], ['fine', '좋다'], ['fish', '생선'], ['flow',
'흐름'], ['fund', '기금'], ['gain', '수익'], ['game', '놀이'],
['gift', '선물'], ['girl', '소녀'], ['give', '주다'], ['goal',
'목적'], ['gray', '회색'], ['hair', '머리'], ['harm', '피해'],
['hell', '지옥'], ['help', '도움'], ['high', '높은'], ['hole',
'구멍'], ['hope', '소망'], ['hour', '시각'], ['join', '합류'],
['kiss', '키스'], ['knee', '무릎'], ['lady', '부인'], ['late',
'늦은'], ['left', '왼쪽'], ['life', '생명'], ['loss', '손해'],
['love', '사랑'], ['luck', '행운'], ['mail', '우편'], ['male',
'남자'], ['many', '많은'], ['meal', '식사'], ['meat', '고기'],
['menu', '메뉴'], ['milk', '우유'], ['name', '이름'], ['news',
'뉴스'], ['next', '다음'], ['once', '한번'], ['pain', '고통'],
['part', '부분'], ['pick', '선택'], ['pink', '핑크'], ['plan',
'계획'], ['read', '읽다'], ['rest', '휴식'], ['ring', '반지'],
['rock', '바위'], ['roof', '지붕'], ['root', '뿌리'], ['rose',
'장미'], ['rule', '규칙'], ['salt', '소금'], ['sand', '모래'],
['sell', '팔다'], ['shop', '가게'], ['sign', '싸인'], ['size',
'크기'], ['skin', '피부'], ['slow', '늦다'], ['song', '노래'],
['soon', '금방'], ['tall', '크다'], ['test', '시험'], ['them',
'그들'], ['thin', '얇은'], ['this', '이것'], ['time', '시간'],
['tiny', '작은'], ['tool', '도구'], ['tour', '관광'], ['tree',
'나무'], ['trip', '여행'], ['very', '매우'], ['wave', '파도'],
['weak', '약한'], ['wear', '입다'], ['west', '서쪽'], ['when',
'언제'], ['wide', '넓은'], ['wife', '아내'], ['wind', '바람'],
```

```
['wing', '날개'], ['wish', '바램'], ['wood', '나무'], ['word',  
'단어'], ['year', '연도']]
```

총 단어 수: 110

다음은 영어와 한글을 인코딩 입력, 디코딩 입력 그리고 디코딩 출력으로 변환하는 함수입니다. RNN의 출력은 `expand_dim()` 함수를 이용해서 3차원 구조로 바꿔야 하는 것을 잊지 마세요.

```
1 def encode(eng_kor):  
2     enc_in = []  
3     dec_in = []  
4     rnn_out = []  
5  
6     for seq in eng_kor:  
7         eng = [char_to_num[c] for c in seq[0]]  
8         enc_in.append(np.eye(n_input)[eng])  
9         kor = [char_to_num[c] for c in ('S'+seq[1])]   
10        dec_in.append(np.eye(n_input)[kor])  
11        target = [char_to_num[c] for c in (seq[1] + 'E')]  
12        rnn_out.append(target)  
13  
14    enc_in = np.array(enc_in)  
15    dec_in = np.array(dec_in)  
16    rnn_out = np.array(rnn_output)  
17  
18    rnn_out = np.expand_dims(rnn_out, axis=2)  
19    return enc_in, dec_in, rnn_out
```

다음은 임의의 단어 하나를 인코딩해 봅니다.

```
1 sample = [['word', '단어']]  
2 enc_in, dec_in, rnn_out = encode(sample)
```



```
1 print('\n인코더 입력 값 샘플')
2 print('데이터 모양:', enc_in.shape)
3 print(enc_in)
4 for i in range(4):
5     char = np.argmax(enc_in[0, i])
6     print(num_to_char[char])
```

데이터 모양: (1, 4, 171)

5

```

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. 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. 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.]]]

w
o
r
d

```

다음은 디코더 입력 값 샘플을 출력해 봅니다.

```

1 print('\n디코더 입력 값 샘플')
2 print('데이터 모양:', dec_in.shape)
3 print(dec_in)
4 for i in range(3):
5     char = np.argmax(dec_in[0,i])
6     print(num_to_char[char])

```

디코더 입력 값 샘플

데이터 모양:(1, 3, 171)

```

[[[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. 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. 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. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 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. 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. 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. 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. 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. 0. 0. 0.]

```

S
단
어

다음은 RNN 출력 값 샘플입니다.

```

1 print('\nRNN 출력 값 샘플')
2 print('데이터 모양:', rnn_out.shape)
3 print(rnn_out)
4 for i in range(3):
5     num = rnn_out[0, i, 0]
6     print(num_to_char[num])

```

```

RNN 출력 값 샘플
데이터 모양:(1, 3, 1)
[[[ 61]
    [114]
    [ 1]]]

```

단
어
E

다음은 인코더와 디코더의 입력 데이터, 그리고 출력 데이터의 shape

를 확인해 봅니다.

```
1 X_enc, X_dec, Y_rnn = encode(eng_kor)
2 print('인코더 입력값 모양', X_enc.shape)
3 print('디코더 입력값 모양', X_dec.shape)
4 print('출력값 모양', Y_rnn.shape)
```

인코더 입력값 모양(110, 4, 171)

디코더 입력값 모양(110, 3, 171)

출력값 모양(110, 3, 1)

- 결과에서 110은 총 단어의 수입니다. 인코더 입력 모양에서 4는 단어의 길이이며, 디코더 입력 모양과 출력 모양에서 3은 한글 단어 2에 디코더 입력에는 S가 포함되고, 디코더 출력에는 E가 포함되어 3이 됩니다. 인코더 디코더의 171은 영문알파벳, 한글 문자 그리고 특수문자 SEP를 포함한 총 글자의 수입니다. 원-핫 인코딩 되어야 하므로 크기가 171입니다.

이제 Seq2Seq 구조를 보겠습니다.

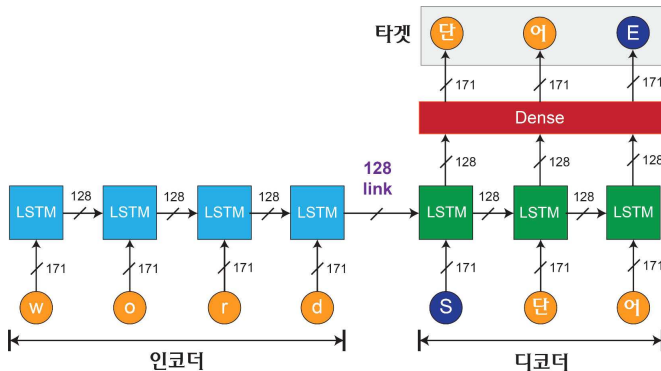


그림 65. Seq2Seq 구조

예제에서는 입력에 영어 단어 4글자와 한글 단어(S 1글자 + 2글자)이며, 출력에 한글 단어(2글자 + E 1글자)이고 활성화 함수는 softmax를 사용합니다.

인코딩 층의 LSTM 출력에 hidden_state와 cell state 정보를 동시에 디코딩 층으로 전달해야 하므로 return_state=True를 사용해야 합니다. 그리고 디코딩의 출력이 LSTM의 모든 단의 출력값을 사용해야 하므로 return_sequences=True를 사용해야 합니다.

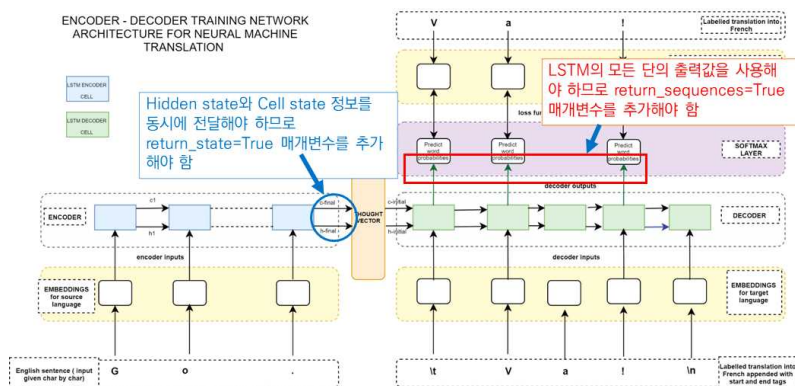


그림 66. Seq2Seq와 LSTM

다음은 LSTM을 이용하여 인공신경망 모델을 정의합니다.

```

1 enc_IN = Input(shape=(4, n_input))
2 _, state_h, state_c = LSTM(units=MY_HIDDEN,
                             return_state=True)(enc_IN)
3 link = [state_h, state_c]
4
5 dec_IN = Input(shape=(3, n_input))
6
7 dec_Y1 = LSTM(units=MY_HIDDEN,
                return_sequences=True)(dec_IN, initial_state=link)
8
9 dec_Y2 = Dense(units=n_input, activation='softmax')(dec_Y1)
10 model = Model(inputs=[enc_IN, dec_IN], outputs=dec_Y2)
11
12 model.summary()

```

Model: "model"

Layer(type)	Output Shape	Param #	Connected to
input_1(InputLayer)	[(None, 4, 171)]	0	[]
input_2(InputLayer)	[(None, 3, 171)]	0	[]
lstm(LSTM)	[(None, 128), (None, 128), (None, 128)]	153600	['input_1[0][0]']
lstm_1(LSTM)	(None, 3, 128)	153600	['input_2[0][0]', 'lstm[0][1]', 'lstm[0][2]']
dense(Dense)	(None, 3, 171)	22059	['lstm_1[0][0]']

Total params: 329,259

Trainable params: 329,259

Non-trainable params: 0

다음은 훈련을 정의합니다.

```
1 model.compile(optimizer='adam',  
2               loss='sparse_categorical_crossentropy')
```

모델을 학습시킵니다.

```
1 begin = time()  
2 model.fit([X_enc, X_dec], Y_rnn, epochs=MY_EPOCH)  
3 end = time()  
4 print(f"총 학습시간:{(end-begin):.2f}초")
```

Epoch 1/500

4/4 [=====] - 9s 15ms/step - loss: 5.1319

```
Epoch 2/500
4/4 [=====] - 0s 16ms/step - loss: 5.0954
... 생략 ...
Epoch 499/500
4/4 [=====] - 0s 13ms/step - loss: 0.0041
Epoch 500/500
4/4 [=====] - 0s 10ms/step - loss: 0.0041
총 학습시간:89.23초
```

임의의 데이터 5개를 선택하고 그 뒤에 'love' 단어를 알파벳이 바뀐 단어로 만들고 예측에 사용할 예제 데이터로 만듭니다. 'love' 단어의 알파벳이 바뀐 단어 중에서 어떤 것을 '사랑'이라고 번역할 수 있을지 기대하세요. 'PP'는 디코딩층의 입력이 없으므로 앞에서 지정한 P문자 두 개를 사용한 것입니다.

```
1 try_this = [['love', 'PP'], ['olve', 'PP'], ['lvoe', 'PP'],
  ['loev', 'PP'], ['eovl', 'PP']]
2 new_data = choose + try_this
```

샘플데이터를 이용해서 예측합니다.

```
1 enc_in, dec_in, _ = encode(new_data)
2 pred = model.predict([enc_in, dec_in])
3 print("예측 결과의 모양:", pred.shape)
```

예측한 결과에서 한글을 추출하려면 `argmax()` 함수와 `num_to_char` 배열을 이용합니다.

```
1 for i in range(len(new_data)):
2     eng = new_data[i][0]
3     word = np.argmax(pred[i], axis=1)
```

```
4     kor = ''
5     for j in range(2):
6         kor = kor + num_to_char[word[j]]
7     print(eng, '->', kor)
```

```
part -> 부분
feet -> 다리
wide -> 넓은
west -> 서쪽
rock -> 바위
love -> 사랑
olve -> 사랑
lvoe -> 사랑
loev -> 사랑
eovl -> 도다
```

- 이 장에서 실습에 사용한 소스코드는 다음 주소에서 내려받을 수 있습니다.

- <http://javaspecialist.co.kr/board/1078>

5절. BBC 기사 분류하기

5.1. BBC 기사 데이터셋

이 절은 BBS 기사를 분류하는 것을 설명합니다. BBC는 1927년에 영국 런던에 설립된 세계 최대 규모의 공영 방송 기업입니다. BBC는 영국 내에서 모든 가구에 부과되는 시청료로 운영됩니다. 영국 이외 지역에서는 BBC 월드와이드를 통해 방송 서비스가 제공됩니다.



그림 67. BBC TV Centre

https://commons.wikimedia.org/wiki/File:BBC_TV_Centre.jpg

BBC 뉴스 기사 데이터셋은 2004년부터 2005년까지 2,225개의 뉴스 기사이며 5개의 카테고리(business, entertainment, politics, sport, tech)를 포함합니다.

이 데이터셋은 모두 영문 소문자로 변환되어 있으며, 제목과 본문의

구분이 없고, 콤마 등 구두점이 일부는 남아있지만, 어느 정도 전체적인 데이터입니다.

앞 장의 로이터 데이터는 텍스트가 숫자 데이터로 변환되어 있지만, BBC 뉴스테이터는 텍스트로 되어있습니다. 다음은 BBC 뉴스 기사 샘플입니다.

```
blair prepares to name poll date tony blair is likely to name 5 may
as election day when parliament returns from its easter break the
bbc s political editor has learned. andrew marr says mr blair will
ask the queen on 4 or 5 april to dissolve parliament at the end of
that week. mr blair has so far resisted calls for him to name the
day but all parties have stepped up campaigning recently. ...
```

각 카테고리당 관련어 순위를 보면 아래와 같습니다.

표 5. BBC 카테고리당 관련어 순위

	기사 수	한 단어	두 단어
business	510	growth, oil, bank	oil price, economic growth, stock market
entertainment	386	oscar, award, film	log angeles, film festival, box office
politics	417	blair, election, labour	prime minister, mr brown, mr blair
sport	511	champion, cup, match	champion league, australian open, grand slam
tech	401	technology, user, mobile	high definition, mobile phone, email

실습을 위해 데이터파일이 필요합니다. 이 파일은 아래의 주소에서 내려받을 수 있습니다. 이 데이터의 첫 번째 열은 카테고리이며 두 번째 열은 뉴스 기사 데이터입니다.

- <http://jaspecialist.co.kr/board/1079>

```

1 category,text
2 tech,tv future in the hands of viewers with home theatre systems plasma high-defi
3 business,worldcom boss left books alone former worldcom boss bernie ebbers who
4 sport,tigers wary of farrell gamble leicester say they will not be rushed into m
5 sport,yeading face newcastle in fa cup premiership side newcastle united face a tr
6 entertainment,ocean s twelve raids box office ocean s twelve the crime caper sequ
7 politics,howard hits back at mongrel jibe michael howard has said a claim by peter
8 politics,blair prepares to name poll date tony blair is likely to name 5 may as el
9 sport,henman hopes ended in dubai third seed tim henman slumped to a straight sets
10 sport,wilkinson fit to face edinburgh england captain jonny wilkinson will make hi
11 entertainment,last star wars not for children the sixth and final star wars movi
12 entertainment,berlin cheers for anti-nazi film a german movie about an anti-nazi r
13 business,virgin blue shares plummet 20% shares in australian budget airline virgin
14 business,crude oil prices back above $50 cold weather across parts of the united s
15 politics,hague given up his pm ambition former conservative leader william Hague
16 sport,moya emotional after davis cup win carlos moya described Spain s davis cup v
17 business,s Korean credit card firm rescued South Korea s largest credit card firm
18 politics,Howard backs stem cell research Michael Howard has backed stem cell resea
19 sport,connors boost for British tennis former world number one Jimmy Connors is pl
20 business,japanese banking battle at an end Japan s Sumitomo Mitsui Financial has w
21 tech,games maker fights for survival one of Britain s largest independent game mak

```

그림 68. bbc-text.csv

이 데이터는 Tokenizer 기술을 이용해서 문장을 단어로 변환하고, 단어를 숫자로 변환하는 작업을 해야 합니다. 이때 사용하는 클래스와 함수는 다음과 같습니다.

- Tokenizer
- fit_on_texts
- texts_to_sequences

5.2. 데이터 전처리하기

BBC 기사를 분석하기 전에 패키지를 불러오고 데이터를 전처리해야 합니다.

다음은 사용할 패키지를 불러옵니다.

```

1 import csv
2 import numpy as np
3 from time import time
4

```

```

5 import nltk
6
7 from keras.preprocessing.text import Tokenizer
8 from keras.preprocessing.sequence import pad_sequences
9
10 from sklearn.model_selection import train_test_split
11 from sklearn.metrics import confusion_matrix
12
13 from keras.models import Sequential
14 from keras.layers import Dense, Dropout
15 from keras.layers import LSTM, Embedding
16 from keras.layers import Bidirectional

```

다음은 하이퍼 파라미터를 정의합니다.

```

1 MY_WORDS = 5000 # 사전 안의 단어 수
2 MY_EMBED = 64   # 임베딩 출력 차원
3 MY_HIDDEN = 100 # RNN 뉴런의 개수 규모
4 MY_LEN = 200    # 기사 길이
5
6 MY_SPLIT = 0.8  # 학습용 데이터의 비율
7 MY_SAMPLE = 123 # 임의의 샘플 기사
8 MY_EPOCH = 10   # 반복 학습 수

```

다음은 기사 원본과 전처리한 데이터 그리고 카테고리 라벨을 저장할 변수를 선언합니다.

```

1 original = [] # 기사 원본
2 processed = [] # 전처리된 기사
3 labels = []   # 기사 카테고리 라벨

```

nltk 자연어처리 패키지에서 불용어를 불러옵니다. 불용어(또는 제외

어)는 데이터 분석에서 사용하지 않을 단어들을 의미합니다.

```
1 nltk.download("stopwords")
2 MY_STOP = set(nltk.corpus.stopwords.words('english'))
3
4 print("불용어 수:", len(MY_STOP))
5 print(MY_STOP)
```

[nltk_data] Downloading package stopwords to /root/nltk_data...

[nltk_data] Unzipping corpora/stopwords.zip.

불용어 수: 179

{'before', 'off', "haven't", 'this', 'these', 'not', 's', 'hasn',
'herself', "should've", 've', 'having', 'him', 'hadn', 'too',
'other', 'isn', 'yourself', 'from', 'when', "needn't", 'only',
'couldn', 'your', ... 생략 ...

다음은 데이터파일에서 데이터를 불러옵니다. 불러올 때 불용어에 해당하는 단어는 삭제 후 별도의 변수에 저장합니다.

```
1 with open('bbc-text.csv', 'r') as file :
2     reader = csv.reader(file)
3     next(reader) # 첫 번째 행(열이름) 건너뛰기
4     for row in reader:
5         labels.append(row[0]) # 라벨
6         original.append(row[1]) # 원본 텍스트
7         news = row[1]
8         for word in MY_STOP:
9             token = ' ' + word + ' '
10            news = news.replace(token, ' ')
11            processed.append(news) # 불용어가 삭제된 텍스트
```

불용어를 삭제하기 전의 샘플 기사를 출력하고 단어 수를 확인합니다.

```
1 sample = original[MY_SAMPLE]
2 print('불용어 삭제 전 샘플 기사')
3 print(sample)
```

```

4 print('카테고리:', labels[MY_SAMPLE])
5 print('단어 수:', len(sample.split()))

```

불용어 삭제 전 샘플 기사

screensaver tackles spam websites net users are getting the chance to fight back against spam websites internet portal lycos has made a screensaver that endlessly requests data from sites that sell the goods and services mentioned in spam e-mail. lycos hopes it will make the monthly bandwidth bills of spammers soar by keeping their servers running flat out. ... 생략 ...

카테고리: tech

단어 수: 536

불용어를 삭제한 후 샘플 기사를 확인합니다.

```

1 print('불용어 삭제 후 샘플 기사')
2 sample = processed[MY_SAMPLE]
3 print(sample)
4 print('단어 수:', len(sample.split()))

```

불용어 삭제 후 샘플 기사

screensaver tackles spam websites net users getting chance fight back spam websites internet portal lycos made screensaver endlessly requests data sites sell goods services mentioned spam e-mail. lycos hopes make monthly bandwidth bills spammers soar keeping servers running flat out. ... 생략 ...

단어 수: 303

다음 코드는 불용어가 삭제된 데이터를 토큰화합니다.

```

1 A_token = Tokenizer(num_words=MY_WORDS, oov_token='OOV')
2
3 A_token.fit_on_texts(processed)
4 A_tokenized = A_token.texts_to_sequences(processed)
5
6 maxlen = max([len(x) for x in A_tokenized])

```

```

7 minlen = min([len(x) for x in A_tokenized])
8
9 print('토큰화 결과')
10 print(f'총 단어 수: {len(A_token.word_counts)}')
11 print(f'가장 긴 기사: {maxlen}')
12 print(f'가장 짧은 기사: {minlen}')
```

토큰화 결과

총 단어 수: 29698
가장 긴 기사: 2280
가장 짧은 기사: 50

이제 기사의 길이를 모두 200(MY_LEN)으로 맞춥니다. `pad_sequences()` 함수는 정수 시퀀스(sequence)의 리스트를 반환합니다. 길면 뒤를 자르고 짧으면 뒤에 0을 채웁니다. 결과의 숫자는 단어의 고유 인덱스입니다. 가장 많이 나오는 단어가 인덱스 1을 갖습니다.

```

1 A_tokenized = pad_sequences(A_tokenized, maxlen=MY_LEN,
                             padding='post', truncating='post')
2 A_tokenized = np.array(A_tokenized)
3
4 sample = A_tokenized[MY_SAMPLE]
5 print('샘플 기사 토큰화 후 패딩 결과')
6 print(sample)
7 print('단어 수:', len(sample))
```

샘플 기사 토큰화 후 패딩 결과

```
[3170  1 816 878 115 136 382 347 716  28 816 878 228  1
 3171 27 3170  1 4869 204 569 733 1771 126 4025... 생략 ...
단어 수: 311
```

다음 코드는 카테고리를 숫자로 만듭니다.

```

1 C_token = Tokenizer()
2 C_token.fit_on_texts(labels)
```

```

3 C_tokenized = C_token.texts_to_sequences(labels)
8 C_tokenized = np.array(C_tokenized).reshape(-1)
4
5 print('카테고리 토큰화 결과')
6 print(C_token.word_index)
7 print('샘플 기사 카테고리:', C_tokenized[MY_SAMPLE])

```

카테고리 토큰화 결과

```
{'sport': 1, 'business': 2, 'politics': 3, 'tech': 4,
'entertainment': 5}
```

샘플 기사 카테고리: 4

예측한 결과를 이용해서 카테고리를 알기 위해 인덱스-라벨 딕셔너리 데이터를 만듭니다.

```

1 idx_to_label = {}
2 for label, index in C_token.word_index.items():
3     idx_to_label[index] = label
4
5 print(idx_to_label)

```

```
{1: 'sport', 2: 'business', 3: 'politics', 4: 'tech', 5:
'entertainment'}
```

● 실행결과는 다를 수 있습니다.

다음은 학습 데이터와 평가데이터로 나누고 모양을 확인해 봅니다.

```

1 X_train, X_test, Y_train, Y_test = \
2     train_test_split(A_tokenized, C_tokenized,
3                       train_size=MY_SPLIT, shuffle=False)
4 print(X_train.shape, Y_train.shape, X_test.shape, Y_test.shape)

```

```
(1780, 200) (1780,) (445, 200) (445,)
```


5.3. RNN 모델

다음은 RNN 구조입니다. 임베딩 층과 LSTM 층을 가지고 있습니다.

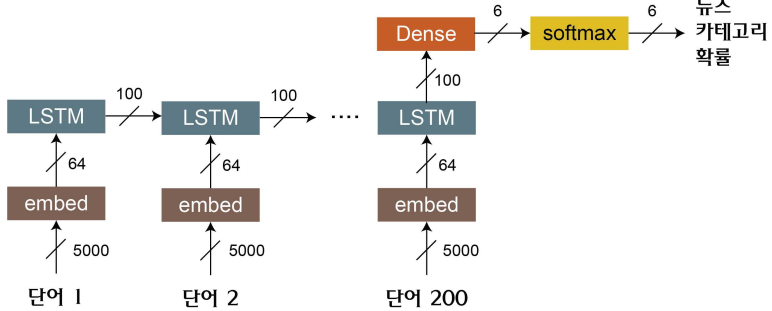


그림 69. 구현한 RNN 구조

여기에서는 주목해야 하는 것은 LSTM 신경망과 Bidirectional 신경망을 사용했다는 것입니다.

다음 코드는 모델의 구조를 정의합니다. LSTM 신경망을 양방향으로 구성하기 위해 Bidirectional() 클래스를 사용했습니다.

```
1 model = Sequential()
2 model.add(Embedding(input_dim=MY_WORDS, output_dim=MY_EMBED))
3 model.add(Dropout(rate=0.5))
4 model.add(Bidirectional(LSTM(units=MY_HIDDEN)))
5 model.add(Dense(units=6, activation='softmax'))
6 model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, None, 64)	320000
dropout (Dropout)	(None, None, 64)	0

```
bidirectional (Bidirectional) (None, 200)          132000
```

```
dense (Dense)          (None, 6)          1206
```

```
=====
Total params: 453,206
Trainable params: 453,206
Non-trainable params: 0
-----
```

다음은 훈련을 정의합니다. 손실함수는 크로스 엔트로피, 옵티마이저는 adam을 사용합니다.

```
1 model.compile(loss='sparse_categorical_crossentropy',
                optimizer='adam', metrics=['acc'])
```

다음 코드는 모델을 학습시킵니다.

```
1 begin = time()
2 model.fit(x=X_train, y=Y_train, epochs=MY_EPOCH, verbose=2)
3 end = time()
4 print(f'학습시간: {end-begin:.2f}')
```

```
Epoch 1/10
```

```
56/56 - 13s - loss: 1.5731 - acc: 0.2966 - 13s/epoch - 236ms/step
```

```
Epoch 2/10
```

```
56/56 - 5s - loss: 1.4337 - acc: 0.4022 - 5s/epoch - 90ms/step
```

```
Epoch 3/10
```

```
56/56 - 5s - loss: 0.8429 - acc: 0.6624 - 5s/epoch - 90ms/step
```

```
Epoch 4/10
```

```
56/56 - 5s - loss: 0.5420 - acc: 0.8567 - 5s/epoch - 90ms/step
```

```

Epoch 5/10
56/56 - 5s - loss: 0.3359 - acc: 0.8966 - 5s/epoch - 90ms/step
Epoch 6/10
56/56 - 5s - loss: 0.1160 - acc: 0.9708 - 5s/epoch - 90ms/step
Epoch 7/10
56/56 - 5s - loss: 0.0472 - acc: 0.9904 - 5s/epoch - 90ms/step
Epoch 8/10
56/56 - 5s - loss: 0.0243 - acc: 0.9955 - 5s/epoch - 90ms/step
Epoch 9/10
56/56 - 5s - loss: 0.0145 - acc: 0.9978 - 5s/epoch - 90ms/step
Epoch 10/10
56/56 - 5s - loss: 0.0233 - acc: 0.9955 - 5s/epoch - 90ms/step
학습시간: 59.03

```

이제 모델을 평가해 보겠습니다.

```

1 score = model.evaluate(x=X_test, y=Y_test, verbose=2)
2 print(f"최종 정확도: {score[1]}")

```

```

14/14 - 1s - loss: 0.2172 - acc: 0.9461 - 1s/epoch - 98ms/step
최종 정확도: 0.9460673928260803

```

평가용 데이터를 이용해서 예측하고 출력합니다.

```

1 pred = model.predict(x=X_test)
2 pred = np.argmax(pred, axis=1)
3 print(pred)

```

```

[5 4 3 1 1 4 2 4 5 5 3 3 2 5 1 5 5 2 1 3 4 2 1 5 4 3 3 1 1 3 2 2 2 5 2 3
 3 4 4 5 3 5 2 3 1 1 3 4 2 4 1 2 2 3 1 1 3 3 5 5 3 2 3 3 2 4 3 3 3 3 5 5
 4 3 1 5 1 4 5 1 1 5 4 5 4 1 4 1 1 5 5 2 5 5 3 2 1 4 4 3 2 1 5 5 1 3 5 1 1
 2 3 4 4 2 2 5 3 5 1 1 3 5 4 5 5 2 3 1 3 4 5 1 3 2 5 3 5 3 1 3 2 2 3 2 4 1
 2 5 2 1 1 5 4 3 4 3 3 1 1 1 2 4 5 2 1 2 3 2 4 2 2 2 2 1 5 1 2 2 5 2 2 2 1
 1 1 4 4 1 5 1 2 5 4 4 4 3 2 2 4 2 4 1 1 3 3 3 1 1 3 3 4 2 1 1 1 1 2 1 2 2
 2 2 1 3 1 4 4 1 4 2 5 2 1 2 4 4 3 5 2 5 2 2 3 5 2 5 5 4 3 4 4 2 3 1 5 2 3

```

```

5 2 4 1 4 3 1 3 2 3 3 2 2 2 4 3 2 3 2 5 3 1 3 3 1 5 4 4 2 4 1 4 2 2 1 4 4
4 1 5 1 3 2 3 3 5 4 2 4 1 5 5 1 2 5 4 4 1 5 2 3 5 3 4 4 2 3 2 4 3 5 5 4 2
4 5 4 4 1 3 1 1 3 5 5 2 3 3 1 2 2 4 2 4 4 1 2 3 1 2 2 1 4 1 4 5 1 5 5 2 4
1 1 3 4 2 3 1 1 3 2 4 4 4 2 1 5 4 4 2 3 4 1 1 4 4 3 2 1 5 5 1 5 4 1 2 2 2
1 1 4 1 2 4 2 2 1 2 3 2 2 5 3 4 3 4 5 3 4 5 1 3 5 1 4 2 4 5 4 1 2 2 3 5 3
1]

```

다음은 혼동 행렬을 출력해 봅니다.

```
1 print(confusion_matrix(y_true=Y_test, y_pred=pred))
```

```

[[ 96   0   4   0   1]
 [  0 102   4   0   0]
 [  2   3  78   3   0]
 [  1   1   0  82   2]
 [  0   1   1   1  63]]

```

이제 bbc.com 사이트에 방문에서 뉴스 기사 하나를 가져와 분류해 보겠습니다. 수집한 뉴스 기사는 다음 코드에서 안내한 곳에 붙여넣고 실행시키세요.

```

1 import re
2 news = ['']
3 여기에 기사를 붙여넣으세요.
4 ''.lower()]
5
6 news[0] = re.sub(r'^\w\s', '', news[0]) # 구두점 제거
7
8 print("불용어 처리 전 문자 수:", len(news[0]))
9 for word in MY_STOP: # 불용어 처리
10     token = ' ' + word + ' '
11     news[0] = news[0].replace(token, ' ')
12 print("불용어 처리 후 문자 수:", len(news[0]))
13
14 seq = A_token.texts_to_sequences(news)
15 padded = pad_sequences(seq, maxlen=MY_LEN,

```

```
16             padding='post', truncating='post')
17
18 my_pred = model.predict(padded)
19 print("예측 결과:", idx_to_label[np.argmax(my_pred[0])])
```

불용어 처리 전 문자 수: 915

불용어 처리 후 문자 수: 661

예측 결과: sport

실행결과는 다음 주소의 뉴스 텍스트를 이용해서 카테고리를 분류한 결과입니다.

- <https://www.bbc.com/news/world-us-canada-59814542>

6절. 음성 인식하기

코드를 작성하기 전에 실습 파일(audio.zip)을 코랩 가상환경에 올려주세요. 그리고 빈 코드 셀에서 다음 명령을 실행시켜 압축파일을 풀어야 합니다. -qq 옵션은 압축을 풀 때 로그를 출력하지 않습니다. -d ./audio 옵션은 audio 폴더에 압축을 풉니다.

```
1 ! unzip -qq -d ./audio audio.zip
```

제공한 데이터셋은 캐글의 “Tensorflow Speech Recognition Challenge”³⁾ 데이터셋 일부입니다.

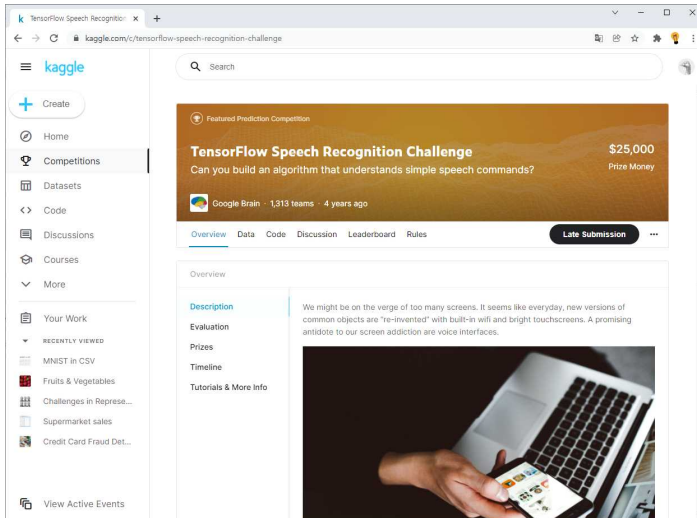


그림 70. TensorFlow Speech Recognition Challenge

3) <https://www.kaggle.com/c/tensorflow-speech-recognition-challenge>

6.1. librosa 패키지

코랩 환경에서 이 절의 예제를 실행시키기 위해서는 sklearn, keras 등 외에도 소리 파일을 처리하기 위해 librosa 패키지가 필요합니다.

librosa(리브로사) 패키지는 파이썬의 음성 및 음악을 처리하는 패키지 중에서 가장 많이 사용되는 패키지이며 예제코드에서 이 패키지의 load() 함수와 resample() 함수를 사용합니다.




그림 71. <https://librosa.org/>

librosa 패키지의 공식 사이트는 <https://librosa.org/>입니다. 도움말을 얻으려면 다음 주소를 참고하세요.

- <https://librosa.org/doc/latest/index.html>


load() 함수는 오디오 파일을 읽어옵니다.

 `librosa.load(path, sr=22050, mono=True, offset=0.0, duration=None, dtype='numpy.float32', res_type='kaiser_best') → y, sr`

- ▶ path: 파일의 경로를 지정합니다.
- ▶ sr: 샘플링 비율을 지정합니다.
- ▶ mono: 신호를 모노로 변환합니다.
- ▶ offset: 지정한 초 이후부터 읽기 시작합니다.
- ▶ duration: 지정한 초 만큼만 읽습니다.
- ▶ dtype: 출력 y의 타입을 지정합니다.
- ▶ res_type: 리샘플링 타입을 지정합니다. 기본값은 고품질 모드인 'kaiser_best'입니다. 다른 모드는 resample() 함수를 참고하세요.

- ▶ 반환하는 값: y , sr
 y 는 shape가 $(n,)$ 이거나 $(2,n)$ 인 넘파이 배열입니다.
 sr 은 샘플링 비율입니다.

`resample()` 함수는 샘플링비율을 변경합니다.

 `librosa.resample(y, orig_sr, target_sr, res_type='kaiser_best', fix=True, scale=False, **kwargs) -> y_hat`

- ▶ y : 넘파이 배열(`np.ndarray`)이며, shape는 $(m,)$ 또는 $(2,n)$ 여야 합니다.
- ▶ `orig_sr`: 원본의 샘플링 비율이며, 0보다 큰 숫자여야 합니다.
- ▶ `target_sr`: 리샘플링 후의 샘플링 비율입니다. 0보다 큰 숫자여야 합니다.
- ▶ `res_type`: 리샘플링 타입입니다. 'kaiser_best'(기본값), 'kaiser_fast', 'fft' 또는 'scipy', 'polyphase', 'linear', 'zero_ord_hold', 'sinc_best', 'sinc_medium', 'sinc_fastest', 'soxr_vhq', 'soxr_hq', 'soxr_mq', 'soxr_lq', 'soxr_qq' 중 하나를 지정할 수 있습니다.
- ▶ `fix`: 리샘플링 신호의 길이를 정확히 다음 식의 결과 크기로 조정합니다.
 $\text{ceil}(\text{target_sr} * \text{len}(y) / \text{orig_sr})$
- ▶ `scale`: 리샘플링된 신호의 크기를 조정하여 y 와 $\hat{y}(y_hat)$ 의 총 에너지가 거의 같게합니다.
- ▶ 반환하는 값: y_hat
 넘파이 배열 $\hat{y}(y_hat)$ 을 반환합니다.
 결과의 shape는 $\text{shape}=(n * \text{target_sr} / \text{orig_sr},)$ 입니다.

1) 샘플링 비율

샘플링 비율(`sampling rate` 또는 `sampling frequency`)은 이산적(離散的)인 신호를 만들기 위해 연속적 신호에서 얻어진 단위 시간(주로 초)당 샘플링 횟수를 정의합니다. 단위는 헤르츠($1/s$, s^{-1})입니다.

오디오에서의 샘플링 비율은 1초당 추출되는 샘플 개수입니다. 오디오에서 44.1KHz(44100Hz), 22KHz(22050Hz)라고 부르는 것을 말

하는데, 44.1KHz는 1초 동안 44,100개로 등분해서 샘플을 추출합니다. 값이 커질수록 더욱더 세밀하게 등분해서 정확한 오디오 데이터를 추출할 수 있습니다. 그러나 너무 큰 값은 추출되는 데이터 크기를 너무 크게 만들어서 처리하기 힘듭니다. 보통 44.1KHz가 CD 음질로 많이 사용되므로 그 이상 추출하는 것은 특수한 경우를 제외하고 의미가 없습니다.

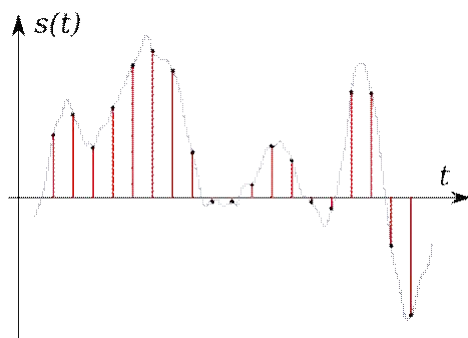


그림 72. 샘플링 비율

https://commons.wikimedia.org/wiki/File:Analog_digital_series.svg

이와 비슷한 단어로 비트율(bit rate)이 있습니다. 이것은 1초당 전송되는 비트 수입니다. 비트율이 나올 수 있는 것은 오디오 압축 기술이 나오면서 가능하게 되었습니다.

비트율은 보통 192Kbps, 128Kbps, 56Kbps 등을 사용합니다. 128Kbps 정도면 음질은 CD 음질 정도 되며, 192Kbps 정도면 최상입니다. 그 이상이 되면 용량이 늘어날 뿐 보통 음질을 잘 구분하지 못합니다.

6.2. 모델의 구조

1) 모델 구조

데이터를 불러와 학습시킬 데이터로 만들었다면 이제 모델의 구조를 정의해야 합니다. 다음은 예제에서 사용할 모델의 구조입니다.

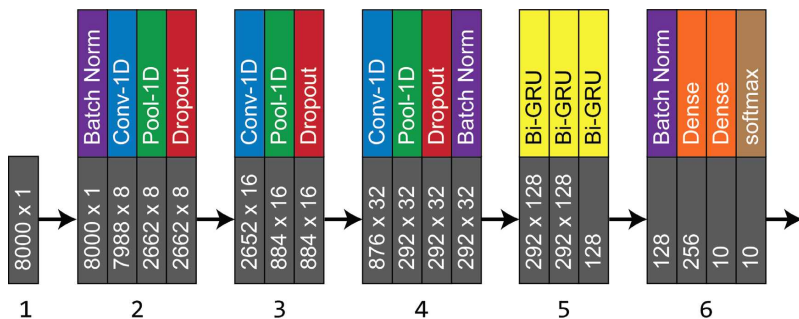


그림 73. 음성인식 모델의 구조

이 모델은 3개의 합성곱 계층과 3개의 GRU, 그리고 은닉층 1개와 출력층으로 구성되어 있습니다. 출력층의 활성화 함수는 소프트맥스를 사용합니다.

모델에서 주의 깊게 봐야 할 중 하나는 합성곱 계층의 맨 앞과 뒤에 배치 정규화를 추가한 것입니다. 배치 정규화는 특정 뉴런의 가중치가 너무 커지는 것을 방지합니다.

6.3. RNN으로 음성 인식하기 구현 코드

이제 음성을 인식하는 코드를 구현해 보겠습니다.

1) 데이터 처리

다음 코드는 라이브러리를 가져옵니다.

```
1 import os
2 import numpy as np
3 import random
4 from time import time
5 from tqdm import tqdm
6
7 import librosa
8 import librosa.display
9
10 import matplotlib.pyplot as plt
11
12 from sklearn.preprocessing import LabelEncoder
13 from sklearn.model_selection import train_test_split
14
15 from keras.utils.np_utils import to_categorical
16 from keras.layers import Bidirectional, BatchNormalization
17 from keras.layers import GRU, Dense, Dropout
18 from keras.layers import Conv1D, Input, MaxPooling1D
19 from keras.models import Model
```

다음은 하이퍼 파라미터를 설정합니다.

```
1 MY_SPLIT = 0.8      # 학습용 데이터 비율
2 MY_HIDDEN = 128     # RNN 다음 단으로 넘겨주는 데이터 숫자
3 MY_DROP = 0.3       # 드롭아웃 비율
4 MY_BATCH = 128      # 학습 시 매번 전달받는 데이터 수
```

```

5 MY_EPOCH = 100      # 반복할 학습횟수
6 MY_PATH = './audio' # 압축을 풀어 놓은 오디오 파일의 경로
7 MY_RATE = 8000      # 샘플링 비율

```

샘플 파일 하나를 불러와서 몇 가지 정보를 조회해 봅니다.

```

1 sound, rate = librosa.load('yes-sample.wav')
2 print("샘플 파일 데이터:", sound)
3 print("샘플 데이터 개수:", len(sound)) # rate와 같음
4 print("샘플링 비율:", rate)

```

```

샘플 파일 데이터: [ 0.00035      0.00059625  0.00060085 ...
-0.00169658 -0.00170654 -0.00091077]

```

```

샘플 데이터 개수: 22050

```

```

샘플링 비율: 22050

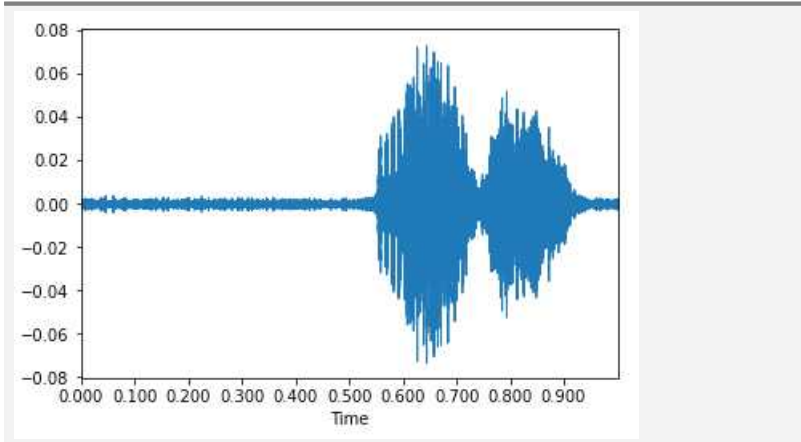
```

waveplot() 함수를 이용하면 파일의 파형을 그려볼 수 있습니다.

```

1 librosa.display.waveplot(sound, sr=rate)
2 plt.show()

```



만일 오디오 파일을 듣고 싶다면 IPython의 Audio 클래스를 사용해 보세요.

```
1 import IPython
2 IPython.display.Audio('yes-sample.wav')
```



다음 코드는 디렉토리 이름을 가져와서 레이블로 만듭니다. 그리고 레이블 인코더를 만들어 학습시킵니다.

```
1 labels = [f for f in os.listdir(MY_PATH) if not os.path.isdir(f)]
2 print(labels)
3 encoder = LabelEncoder()
4 encoder.fit(labels)

['yes', 'happy', 'seven', 'stop', 'go']
LabelEncoder()
```

다음은 오디오 파일을 읽어 배열에 저장하는 함수입니다.

```
1 def read_wave():
2     global all_wave, all_label
3     all_wave = []
4     all_label = []
5     for label in tqdm(labels):
6         path = MY_PATH + '/' + label
7         waves = [f for f in os.listdir(path)]
8         for wav in waves:
9             file = path + '/' + wav
10            samples, rate = librosa.load(file, sr=16000)
11            samples = librosa.resample(samples, orig_sr=rate,
                                       target_sr=MY_RATE)
12            if(len(samples)==MY_RATE):
13                all_wave.append(samples)
14                all_label.append(label)
```

```

15     all_label = encoder.transform(all_label)
16     all_wave = np.array(all_wave).reshape(-1, MY_RATE, 1)
17     print('Resampling 종료')
18     print('전체 샘플링 수:', len(all_wave))
19     return all_wave, all_label

```

다음은 파일을 불러와 넘파이 배열로 만듭니다. 이미 넘파이를 저장한 파일이 있으면 파일에서 불러옵니다.

```

1  if os.path.exists('arrays.npy'):
2      with open('arrays.npy', 'rb') as f:
3          all_wave = np.load(f)
4          all_label = np.load(f)
5  else :
6      all_wave, all_label = read_wave()
7      with open('arrays.npy', 'wb') as f:
8          np.save(f, all_wave)
9          np.save(f, all_label)
10     print('데이터 파일 완성')

```

다음은 훈련데이터와 평가데이터로 나눕니다.

```

1  X_train, X_test, Y_train, Y_test = train_test_split(
2      all_wave, all_label, train_size=MY_SPLIT, shuffle=True )
3  print(X_train.shape, X_test.shape, Y_train.shape, Y_test.shape)

```

(711, 8000, 1) (178, 8000, 1) (711, 5) (178, 5)

이제 RNN 모델링을 해야 합니다. GRU를 사용하고 GRU Stacking을 사용하여 모델을 구현하면 더 좋은 결과를 얻을 수 있습니다.

2) RNN 모델링 코드

이 모델에서는 GRU Stacking을 사용했습니다.

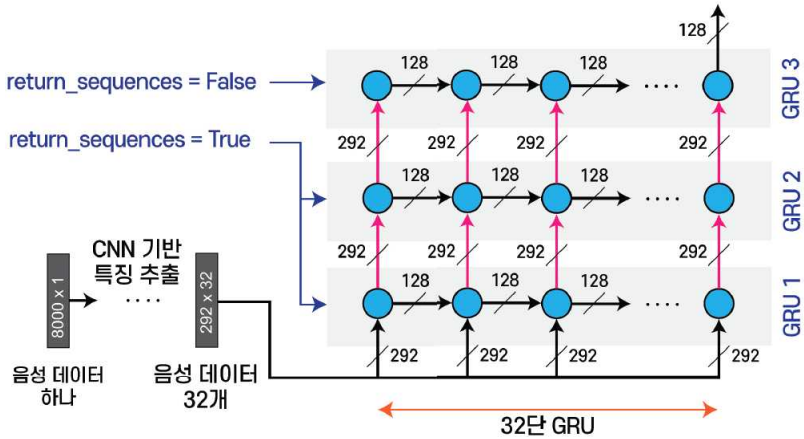


그림 75. GRU Stacking

이 모델은 3개의 GRU 층을 쌓아 만들었습니다. GRU Stacking을 사용하기 위해서 3개의 GRU 층 중에서 3번째 GRU는 return_sequences=False 설정을 해야 합니다.

다음은 모델의 구조를 정의합니다.

```

1 inputs = Input(shape=(MY_RATE, 1))
2 x = BatchNormalization()(inputs)
3
4 # 첫 번째 합성곱
5 x = Conv1D(filters=8, kernel_size=13, padding='valid',
              strides=1, activation='relu')(x)
6 x = MaxPooling1D(pool_size=3)(x)
7 x = Dropout(rate=MY_DROP)(x)
8

```

```

9 # 두 번째 합성곱
10 x = Conv1D(filters=16, kernel_size=11, padding='valid',
             strides=1, activation='relu')(x)
11 x = MaxPooling1D(pool_size=3)(x)
12 x = Dropout(rate=MY_DROP)(x)
13
14 # 세 번째 합성곱
15 x = Conv1D(filters=32, kernel_size=9, padding='valid',
             strides=1, activation='relu')(x)
16 x = MaxPooling1D(pool_size=3)(x)
17 x = Dropout(rate=MY_DROP)(x)
18 x = BatchNormalization()(x)
19
20 # 첫 번째 GRU 층
21 x = Bidirectional(GRU(units=MY_HIDDEN, return_sequences=True),
                   merge_mode='sum')(x)
22
23 # 두 번째 GRU 층
24 x = Bidirectional(GRU(units=MY_HIDDEN, return_sequences=True),
                   merge_mode='sum')(x)
25
26 # 세 번째 GRU 층
27 x = Bidirectional(GRU(units=MY_HIDDEN, return_sequences=False),
                   merge_mode='sum')(x)
28
29 # 은닉층
30 x = BatchNormalization()(x)
31 x = Dense(units=256, activation='relu')(x)
32
33 # 출력층
34 outputs = Dense(units=len(labels), activation='softmax')(x)
35 model = Model(inputs=inputs, outputs=outputs)
36
37 model.summary()

```

Model: "model"

Layer (type)	Output Shape	Param #
--------------	--------------	---------

=====		
input_1 (InputLayer)	[(None, 8000, 1)]	0
batch_normalization (Batch Normalization)	(None, 8000, 1)	4
conv1d (Conv1D)	(None, 7988, 8)	112
max_pooling1d (MaxPooling1D)	(None, 2662, 8)	0
dropout (Dropout)	(None, 2662, 8)	0
conv1d_1 (Conv1D)	(None, 2652, 16)	1424
max_pooling1d_1 (MaxPooling1D)	(None, 884, 16)	0
dropout_1 (Dropout)	(None, 884, 16)	0
conv1d_2 (Conv1D)	(None, 876, 32)	4640
max_pooling1d_2 (MaxPooling1D)	(None, 292, 32)	0
dropout_2 (Dropout)	(None, 292, 32)	0
batch_normalization_1 (Batch Normalization)	(None, 292, 32)	128
bidirectional (Bidirectional)	(None, 292, 128)	124416

bidirectional_1 (Bidirectio	(None, 292, 128)	198144
nal)		
bidirectional_2 (Bidirectio	(None, 128)	198144
nal)		
batch_normalization_2 (Batc	(None, 128)	512
hNormalization)		
dense (Dense)	(None, 256)	33024
dense_1 (Dense)	(None, 5)	1285

```
=====
Total params: 561,833
Trainable params: 561,511
Non-trainable params: 322
-----
```

다음은 훈련을 정의합니다. 손실함수는 크로스 엔트로피, 옵티마이저는 nadam을 사용합니다.

```
1 model.compile(loss='sparse_categorical_crossentropy',
                optimizer='nadam', metrics='acc')
```

모델을 학습시킵니다. 학습된 모델이 있으면 모델을 불러옵니다.

```
1 if os.path.exists('speech.h5'):
2     model.load_weights('speech.h5')
3 else :
4     begin = time()
```

```

5     hist = model.fit(x=X_train, y=Y_train, epochs=MY_EPOCH,
                       batch_size=MY_BATCH, verbose=1)
6     end = time()
7     print(f'총 학습시간: {(end-begin):.2f}초')
8     model.save_weights('speech.h5')

```

Epoch 1/100

6/6 [=====] - 26s 621ms/step - loss: 1.7316 - acc: 0.1983

Epoch 2/100

6/6 [=====] - 4s 595ms/step - loss: 1.6960 - acc: 0.2363

... 생략 ...

Epoch 99/100

6/6 [=====] - 4s 590ms/step - loss: 0.0171 - acc: 0.9944

Epoch 100/100

6/6 [=====] - 4s 590ms/step - loss: 0.0183 - acc: 0.9944

총 학습시간: 395.40초

- 실행결과에서 총 학습시간은 코랩에서 하드웨어 가속기를 GPU로 설정하고 예제 데이터 1000개의 파일을 100번 학습시킨 시간입니다. 실제 모든 데이터를 이용하고 GPU가 없는 환경이라면 훨씬 더 많은 시간이 소요됩니다. 제 PC에서 캐글의 train 데이터셋 전체를 학습시켰을 때 1 epoch 당 대략 40초 정도 걸렸습니다.

다음은 평가데이터를 이용하여 모델을 평가합니다.

```

1 scores = model.evaluate(x=X_test, y=Y_test, verbose=1)
2 print(f'평가데이터의 정확도: {scores[1]:.2f}')

```

6/6 [=====] - 10s 185ms/step - loss:

0.1282 - acc: 0.9775

평가데이터의 정확도: 0.98

다음은 데이터를 입력해서 예측한 결과를 출력하는 함수입니다.

```
1 def audio_to_text(audio):
2     pred = model.predict(audio.reshape(1, MY_RATE, 1))
3     index = np.argmax(pred[0])
4     return labels[index]
```

예제 파일 중에서 임의의 파일 하나를 선택합니다.

```
1 rand_dir = random.choice(os.listdir(MY_PATH))
2 print(rand_dir)
3
4 rand_path = random.choice(os.listdir(MY_PATH+"/"+rand_dir))
5 picked = os.path.join(MY_PATH, rand_dir, rand_path)
6 picked_label = rand_dir
7 print(picked)
8
9 IPython.display.Audio(picked)
```



선택한 파일을 불러와 리샘플링한 후 예측한 결과를 출력합니다.

```
1 y, sr = librosa.load(picked, sr=16000)
2 sample = librosa.resample(y, orig_sr=sr, target_sr=MY_RATE)
3 sample = pad_sequences(np.array(sample).reshape(1, -1),
4                         maxlen=MY_RATE, padding='post',
5                         dtype=float)
4
```

```

5 sample = np.array(sample).reshape(-1, MY_RATE, 1)
6 print(audio_to_text(sample))

```

yes

fit() 함수의 결과는 loss와 accuracy를 반환합니다. fit() 함수의 결과를 시각화하면 학습횟수마다 loss와 accuracy의 변화를 확인할 수 있습니다.

```

1 fig, loss_ax = plt.subplots()
2 loss_ax.plot(hist.history['loss'], 'y', label='train loss')
3 loss_ax.set_xlabel('epoch')
4 loss_ax.set_ylabel('loss')
5 loss_ax.legend(loc='upper left')
6
7 acc_ax = loss_ax.twinx()
8 acc_ax.plot(hist.history['acc'], 'b', label='train acc')
9 acc_ax.set_ylabel('accuracy')
10 acc_ax.legend(loc='lower left')
11 plt.show()

```

