

자연어 처리 기초부터 BERT, RoBERTa, 코파일럿, GPT-4 모델까지

# 트랜스포머로

# 시작하는

# 자연어 처리

Denis Rothman 저  
김윤기, 박지성, 임창대, 하한규 역

자연어 처리 기초부터 BERT, RoBERTa, 코파일럿, GPT-4 모델까지

# 트랜스포머로 시작하는 자연어 처리

Transformers for Natural Language Processing – Second Edition: Build, train, and fine-tune deep neural network architectures for NLP with Python, Hugging Face, and OpenAI’s GPT-3, ChatGPT, and GPT-4  
Copyright ©Packt Publishing 2022.

First published in the English language under the title

‘Transformers for Natural Language Processing – Second Edition –(9781803247335)’

All rights reserved.

KOREAN language edition © 2024 by YOUNGJIN.COM INC.

KOREAN translation rights arranged with Packt Publishing Ltd  
through Lee&Lee Foreign Rights Agency, Korea.

이 책의 한국어판 저작권은 리앤리에이전시를 통한 저작권자와의 독점 계약으로 영진닷컴이 소유합니다.  
저작권법에 의하여 한국 내에서 보호를 받는 저작물이므로 무단 전재 및 복제를 금합니다.

ISBN : 978-89-314-7586-9

## 독자님의 의견을 받습니다.

이 책을 구입한 독자님은 영진닷컴의 가장 중요한 비평가이자 조연가입니다. 저희 책의 장점과 문제점이 무엇인지, 어떤 책이 출판되기를 바라는지, 책을 더욱 알차게 꾸밀 수 있는 아이디어가 있으면 팩스나 이메일, 또는 우편으로 연락주시기 바랍니다. 의견을 주실 때에는 책 제목 및 독자님의 성함과 연락처(전화번호나 이메일)를 꼭 남겨 주시기 바랍니다. 독자님의 의견에 대해 바로 답변을 드리고, 또 독자님의 의견을 다음 책에 충분히 반영하도록 늘 노력하겠습니다.

이메일 : support@youngjin.com

주 소 : (우)08507 서울특별시 금천구 가산디지털1로 128 STX-V타워 4층 401호 (주)영진닷컴 기획1팀

등 록 : 2007. 4. 27. 제16-4189호

파본이나 잘못된 도서는 구입하신 곳에서 교환해 드립니다.

## STAFF

저자 Denis Rothman | 역자 김윤기, 박지성, 임창대, 하현규 | 총괄 김태경 | 기획 김용기 | 디자인·편집 김소연

영업 박준용, 임용수, 김도현, 이윤철 | 마케팅 이승희, 김근주, 조민영, 김민지, 김진희, 이현아

제작 황장협 | 인쇄 제이엠

## 추천사

트랜스포머는 4년도 채 되지 않아 NLP 커뮤니티를 강타하며 지난 30년 동안 달성한 모든 기록을 경신했다. 이제 컴퓨터 비전부터 음성 인식, 번역, 단백질 시퀀싱, 코드 작성에 이르기까지 모든 분야에서 새로운 애플리케이션을 위한 기본 구성 요소로 BERT, T5, GPT와 같은 모델을 사용하고 있다. 이러한 이유로 최근 스탠포드는 사전 학습된 트랜스포머 기반의 거대 언어 모델 집합을 일컫는 ‘파운데이션 모델’이라는 용어를 도입했다. 이러한 발전은 몇 가지 간단한 아이디어 덕분에 가능했다.

이 책은 트랜스포머의 동작 원리를 실용적인 관점과 이론적인 관점 모두 제공한다. 저자는 단계별로 코드를 실습하는 방식으로 트랜스포머 사용 방법을 탁월하게 설명한다. 이 책을 읽은 후, 최신 기술을 딥러닝 애플리케이션에 접목해 개선할 수 있을 것이다. 특히 이 책은 트랜스포머의 아키텍처에 대한 탄탄한 배경 지식을 제공한 후 BERT, RoBERTa, T5, GPT-3과 같은 인기 모델을 자세히 다룬다. 또한, 트랜스포머로 해결할 수 있는 다양한 사용 사례(텍스트 요약, 이미지 라벨링, 질의 응답, 감성 분석 및 가짜 뉴스 분석)를 살펴본다.

이러한 주제에 관심이 있다면 이 책은 확실히 가치 있는 책이다. 초판은 항상 책상 위에 놓여 있고, 2판도 마찬가지일 것이다.

안토니오 굴리 Antonio Gulli

구글 CTO 사무소 소속 엔지니어링 디렉터

최근 머신러닝에 대한 관심이 높아지고 기술이 발전함에 따라, 사람이 직접 수행하던 반복적인 작업들이 자동화되고 해결하지 못했던 문제들이 해결되고 있습니다. 시간이 지남에 따라 데이터는 계속해서 축적되고 있습니다. 이 방대한 양의 데이터를 학습하기 위해 더 크고 더 복잡한 모델이 계속해서 제안되었으며, 트랜스포머는 그중 가장 큰 혁신을 일으켰습니다. 이제 트랜스포머는 모든 인공지능 분야의 기본 소양입니다.

이 책은 트랜스포머의 기본 개념과 작동 방식에 대한 풍부한 정보를 제공합니다. 이 책을 통해 BERT, RoBERTa, GPT-3 등 다양한 트랜스포머 모델의 장단점과 사용 사례를 탐색하여 기계 번역, 감정 분석, 가짜 뉴스 탐지 등의 문제에 활용하는 방법을 이해할 수 있습니다. 또한 블랙박스처럼 감춰진 트랜스포머 모델을 해석하는 방법을 배우고, 모델이 어떻게 결정을 내리는지 이해하면서 모델의 신뢰성을 높일 수 있습니다. 마지막으로 초인간 트랜스포머의 등장과 OpenAI의 ChatGPT 및 GPT-4와 같은 트랜스포머 기술의 최신 동향에 대해 알아볼 수 있습니다.

첫 번역인 만큼 부족한 점이 있을 수 있습니다. 한글은 제 모국어이지만, 짧은 문장에서도 문법적 의미와 언어학적 의미를 살려 번역하는 것은 정말 어려운 일이었습니다. 그러나 원문의 내용을 최대한 보존하고, 계속해서 진화하는 트랜스포머 모델 생태계를 최대한 반영하고자 노력했습니다.

마지막으로, 번역 과정에서 저를 지지해 준 동료, 친구, 가족 등 모든 분들께 깊은 감사의 말씀을 전합니다. 독자 여러분이 이 책을 읽으면서 트랜스포머와 자연어 처리의 매혹적인 세계로 여행하며 통찰력과 영감을 받을 수 있으시길 진심으로 바랍니다.

자연어 처리(Natural Language Processing, NLP)는 글로벌 디지털 시장에서 인공지능을 대표하고 있으며, 트랜스포머(transformer)는 자연어 이해(Natural Language Understanding, NLU)의 판도를 바꾸고 있다.

트랜스포머 모델은 인공지능의 새로운 시대를 알렸다. 자연어 이해는 언어 모델링, 챗봇, 개인 비서, 질문 답변, 텍스트 요약, 음성 인식, 감성 분석, 기계 번역 등의 기반이 되었다. 우리는 소셜 네트워크, 이커머스, 디지털 신문, 스트리밍 서비스, 원격 진료, 재택근무 등 수백 개의 도메인에서 디지털 전환이 이뤄지는 시대를 살고 있다. 언어를 이해하는 인공지능이 없다면 웹브라우저, 스트리밍 서비스 등 언어와 관련된 모든 디지털 활동이 어려워진다. 디지털 전환이라는 패러다임 속에서 인공지능의 발전은 필수적이었다. 인공지능은 조 단위의 단어로 구성된 데이터셋을 학습하기 위해 수십억 이상의 파라미터를 가지도록 진화했다.

트랜스포머의 구조는 매우 혁신적이다. 기존의 RNN과 CNN을 탈피하고 과거의 기술로 만들었다. BERT와 GPT는 순환 네트워크를 셀프-어텐션(self-attention)으로 대체했다. 트랜스포머는 RNN 및 CNN 보다 좋은 성능을 보였다. 2020년대의 AI는 큰 변화를 겪고 있다.

트랜스포머의 인코더와 디코더의 어텐션 헤드(attention head)는 최첨단 하드웨어를 병렬로 활용할 수 있다. 또한 어텐션 헤드는 여러 GPU로도 연산할 수 있기 때문에 수십억 또는 수조 개의 파라미터를 가진 모델도 학습할 수 있다. OpenAI는 GPU 10,000개와 CPU 코어 285,000개로 구성된 슈퍼컴퓨터를 사용해 1,750억 개의 파라미터를 가진 GPT-3 트랜스포머 모델을 학습했다.

데이터가 증가함에 따라 대용량 AI 모델이 필요해졌고, 트랜스포머는 파라미터 기반 AI의 시대를 열었다. 수억 개의 단어가 문장에서 어떻게 어울리는지 이해하려면 엄청난 수의 파라미터가 필요하다.

구글의 BERT나 OpenAI의 GPT-3같은 트랜스포머 기반 모델은 더 높은 차원의 성능을 보여주었다. 트랜스포머는 학습하지 않은 수많은 NLP 작업도 수행할 수 있다.

트랜스포머는 이미지를 단어 시퀀스처럼 임베딩하여 이미지를 분류하거나 재구성 할 수 있다. 컴퓨터 비전(computer vision)에서 사용하는 최신 트랜스포머인 ViT(Vision Transformer), CLIP, DALL-E 또한 이 책에서 살펴볼 것이다.

---

미세 조정(fine-tuning) 없이도 수백 가지 작업을 할 수 있는, 학습된 트랜스포머 모델을 파운데이션 모델(foundation model)이라고 한다. 파운데이션 모델은 방대한 정보 시대의 유용한 도구이다.

소셜 네트워크에 올라오는 메시지 수십억 개가 합법적이고 윤리적인지 판단하려면 몇 명이 필요할지 생각해보자.

하루에 수백만 개가 생성되는 웹페이지를 모두 번역하려면 몇 명의 사람이 필요할까. 또는, 분당 수백만 개가 생성되는 메시지를 사람이 직접 분류한다면 몇 명이 필요할까!

마지막으로, 인터넷에 있는 방대한 스트리밍 데이터에 자막을 달거나 수십억 이미지에 캡션을 달아야 할 때 AI가 없다면 정말 많은 인적 자원이 필요하다.

이 책은 코드 개발부터 프롬프트(prompt)를 디자인하는 방법까지 다룬다. 프롬프트는 트랜스포머의 동작을 제어하는 새로운 “프로그래밍” 기술이다. 각 챕터에서 파이썬(Python), 파이토치(PyTorch), 텐서플로우(TensorFlow) 등을 사용하여 언어 이해의 핵심 요소를 다룰 것이다.

오리지널 트랜스포머와, 구글 BERT, OpenAI GPT-3, T5 등 다양한 모델의 아키텍처를 살펴볼 것이다. 트랜스포머를 학습하고, 미세 조정하고 강력한 API를 사용하는 방법 또한 배울 것이다. 페이스북(Facebook), 구글, 마이크로소프트(Microsoft)와 같은 빅테크 기업이 공개한 거대한 데이터셋도 살펴볼 것이다.

미디어, 소셜 미디어, 연구 논문 등 다양한 분야에서 언어 이해 기술에 대한 시장의 요구를 파악할 것이다. 수백 가지의 AI 작업 중에서도 방대한 양의 데이터를 요약하거나, 모든 경제 분야의 문서를 번역하거나, 윤리적 및 법적으로 소셜 미디어 게시물을 검토하는 작업이 중요하다.

이 책에서는 파이썬, 파이토치, 텐서플로우를 직접 다룰 것이다. 먼저 언어 이해를 위한 주요 신경망 모델을 소개하고, 트랜스포머를 구현하는 방법을 배울 것이다.

이 혁신적인 4차 산업시대에서 AI 전문가에게 필요한 새로운 기술도 알아볼 것이다. 이 책으로 자연어 이해 모델을 효과적으로 개발하는데 필요한 딥러닝 지식과 도구를 제공하고자 한다.

## 이 책의 예상 독자

이 책은 파이썬 프로그래밍이나 기계 학습의 기초를 다루지 않는다. 대신 기계 번역, 음성 인식, 텍스트 음성 변환, 언어 모델링, 질문 답변 등 NLP 분야의 딥러닝 방법론에 집중한다.

이 책으로 가장 많은 도움을 얻을 수 있는 독자는 다음과 같다.

- 파이썬 프로그래밍에 익숙한 딥러닝 및 NLP 실무자
- 점점 증가하는 언어 데이터를 처리하기 위해 자연어 이해 도입을 원하는 데이터 분석가와 데이터 과학자

## 이 책에서 다루는 내용

**1장, 트랜스포머란 무엇인가?**는 트랜스포머를 높은 수준에서 설명한다. 트랜스포머의 생태계와 파운데이션 모델의 속성을 살펴볼 것이다. 4차 산업시대 AI 전문가의 역할과 사용 가능한 플랫폼 또한 알아볼 것이다.

**2장, 트랜스포머 모델 아키텍처 살펴보기**는 NLP의 배경 지식을 확인하며 트랜스포머 아키텍처가 어떻게 RNN, LSTM, CNN 등을 대체했는지 알아본다. 구글 리서치(Google Research)와 구글 브레인(Google Brain)이 독자적으로 개발하고 「Attention Is All You Need」에서 소개한 트랜스포머 아키텍처를 살펴볼 것이다. 트랜스포머의 이론 역시 알아본다. 멀티-헤드 어텐션(multi-head attention)의 동작 방식을 파이썬으로 직접 실행해 본다. 장이 끝나면 트랜스포머의 아키텍처를 이해하게 된다. 이어지는 챕터들에 등장하는 다양한 트랜스포머 응용사례를 살펴볼 준비가 되는 것이다.

**3장, BERT 모델 미세 조정하기**는 트랜스포머 아키텍처를 기반으로 시작한다. **BERT(Bidirectional Encoder Representations from Transformers)**는 NLP의 새로운 접근 방식을 보여준다. 다음 시퀀스를 예측하기 위해 이전 시퀀스만 고려하던 이전 방식 대신 BERT는 전체 시퀀스를 고려한다. BERT 아키텍처의 핵심 요소를 확인하고 구글 코랩(Google Colaboratory)에서 미세 조정을 단계별로 수행해 볼 것이다. 마치 인간처럼, BERT는 새로운 내용을 학습할 때 밑바닥부터 학습하지 않아도 된다.

**4장, RoBERTa 모델 처음부터 사전 학습하기**에서는 허깅페이스(Hugging Face) 파이토치 모듈을 사용하여 RoBERTa 모델을 밑바닥부터 학습해본다. RoBERTa는 BERT 및 DistilBERT와 유사하다. 먼저 커스텀 데이터셋으로 토큰라이저를 학습할 것이다. 모델 학습이 완료되면 마스크드 언어 모델링(masked language modeling) 작업을 수행해본다.

---

**5장, 트랜스포머를 사용한 다운스트림 NLP 작업**에서는 다운스트림(downstream) NLP 작업(task)으로 트랜스포머의 마법 같은 성능을 확인한다. 사전 학습된 트랜스포머를 미세 조정하여 BoolQ, CB, MultiRC, RTE, WiC 등의 NLP 작업을 수행한다. 트랜스포머는 GLUE와 SuperGLUE의 순위표를 점령했다. 여러 작업과 데이터셋, 지표 등과 함께 트랜스포머를 평가하는 과정을 알아볼 것이다.

**6장, 트랜스포머를 사용한 기계 번역**은 인간 기준값과 기계 트랜스덕션(transduction) 방법을 살펴보고 기계 번역 문제를 정의한다. 그런 다음 유럽 의회의 WMT 프랑스어-영어 데이터셋을 전처리할 것이다. 기계 번역에는 정확한 평가 방법이 필요하다. 이 장에서는 BLEU 방법을 사용한다. 마지막에는 트래크(Trax)를 사용하여 트랜스포머 기계 번역 모델을 구현해 볼 것이다.

**7장, GPT-3 엔진을 사용한 초인간 트랜스포머 등장**은 OpenAI의 GPT-2와 GPT-3의 다양한 측면을 살펴본다. 다양한 GPT-3 모델을 확인하기 전에, 먼저 GPT의 구조를 알아볼 것이다. 그 후 345M 파라미터의 GPT-2 모델과 상호 작용 하며 텍스트를 생성해 볼 것이다. 다음으로, GPT-3 플레이그라운드(playground)로 GPT-2의 결과를 비교한 후, GPT-3 모델을 구현하여 NLP 작업을 수행할 것이다.

**8장, 법률 및 금융 문서에 트랜스포머를 적용하여 요약하기**는 T5 모델의 컨셉과 아키텍처를 다룬다. 허깅페이스로 T5 모델을 초기화하고 문서를 요약할 것이다. T5 모델로 권리장전(Bill of Rights) 등을 비롯한 다양한 문서를 요약하고 전이학습(transfer learning)의 성공과 한계를 알아본다. 마지막에는 GPT-3를 사용하여 초등학교 2학년이 이해할 수 있는 수준으로 회사법을 요약할 것이다.

**9장, 데이터셋에 적합한 토큰라이저**에서는 토큰라이저의 한계를 분석하고 데이터의 인코딩(encoding) 품질을 개선하기 위한 방법을 알아본다. 먼저 파이썬 프로그램을 만들어서 Word2Vec 토큰라이저가 몇몇 단어를 누락하거나 잘못 해석하는 이유를 찾을 것이다. 이를 기반으로 학습된 토큰라이저의 한계를 확인한다. 토큰화 과정을 발전 시킬 수 있는 여러 아이디어를 적용하여 T5의 요약 성능을 개선해 볼 것이다. 마지막에는, GPT-3의 언어 이해 성능의 한계를 시험해본다.

**10장, BERT 기반 트랜스포머를 사용한 SRL**에서는 트랜스포머가 텍스트의 내용을 어떻게 이해하는지 살펴본다. 의미역결정(Semantic Role Labeling, SRL)은 사람에게도 쉽지 않은 과제이지만 트랜스포머는 놀라운 결과를 보인다. 앨런 인공지능 연구소(Allen Institute for AI)가 설계한 BERT 기반의 모델을 구글 코랩으로 실행할 것이다. SRL 결과를 시각화하는 온라인 도구 또한 사용할 것이다. 마지막에는 SRL의 한계와 원인에 대해 생각해본다.

**11장, 데이터가 말하게 하기: 스토리, 질문, 답변**에서는 트랜스포머가 어떻게 추론 능력을 학습하는지 알아본다. 트랜스포머는 텍스트와 스토리를 이해하고 추론 능력을 발휘해야 한다. NER과 SRL을 이용하여 질문-답변 능력을 강화해 볼 것이다. 트랜스포머를 학습하는 데 사용하기 위한 질문 생성기 또한 설계해 볼 것이다.

**12장, 고객 감정을 감지해 예측하기**는 트랜스포머로 감성을 분석하는 방법을 보여준다. STT(Stanford Sentiment Treebank) 데이터셋으로 복잡한 문장을 분석하며, 여러 가지 트랜스포머 모델이 문장 구조뿐만 아니라 논리 형식을 파악하도록 할 것이다. 감성 분석 결과에 따른 행동을 예측하기 위해 트랜스포머를 사용하는 방법도 살펴볼 것이다. 챕터의 마지막에는 GPT-3로 엣지 케이스(edge case)를 확인한다.

**13장, 트랜스포머로 가짜 뉴스 분석하기**에서는 가짜 뉴스의 다양한 주제를 탐구하고 우리가 매일 마주하는 다양한 관점의 온라인 콘텐츠를 이해하기 위해 트랜스포머를 활용하는 방법을 알아본다. 소셜 미디어, 웹사이트 등 다양한 실시간 커뮤니케이션이 가능한 다양한 곳에서 매일 수십억 개의 메시지, 게시물, 아티클 등이 발행된다. 이전 챕터들에서 살펴본 기술들을 활용하여 기후 변화와 총기 규제에 대한 논쟁과 전직 대통령의 트윗을 분석할 것이다. 주관적인 뉴스와 합리적 의심을 넘는 가짜 뉴스를 판단하는 도덕적이고 윤리적인 문제를 살펴볼 것이다.

**14장, 블랙박스 트랜스포머 모델 해석하기**에서는 트랜스포머의 행동을 시각화하여 블랙박스의 뚜껑을 열어본다. BertViz로 어텐션 헤드를 시각화하고 LIT(Language Interpretability Tool)로 PCA(Principal Component Analysis)를 적용할 것이다. 마지막에는 LIME과 디셔너리 러닝을 활용하여 트랜스포머를 시각화한다.

**15장, NLP부터 범용 트랜스포머 모델까지**에서는 허깅페이스의 예제를 실행해보며 심화 모델인 리포머(Reformer)와 DeBERTa를 탐구한다. 트랜스포머는 이미지를 단어의 시퀀스처럼 처리 할 수 있다. ViT, CLIP, DALL-E 같은 다양한 비전 트랜스포머 또한 알아보고 이미지 생성 등 컴퓨터 비전 과제를 테스트할 것이다.

**16장, 트랜스포머 기반 코파일럿의 등장**은 4차 산업의 성숙도를 확인한다. 먼저 비공식적이고 케주얼한 영어로 프롬프트 엔지니어링을 시도할 것이다. 그리고 나서, 깃허브(GitHub)의 코파일럿(Copilot)과 OpenAI의 코덱스(Codex)를 사용하여 몇 줄의 명령어만으로 코드를 생성해본다. 비전 트랜스포머의 도움으로 NLP 트랜스포머가 세상을 시각화 하는 것도 볼 수 있다. 메타버스 속 디지털 휴먼을 위한 트랜스포머 기반 추천 시스템도 만들 것이다!

17장, 초인간 트랜스포머를 사용한 OpenAI의 ChatGPT와 GPT-4는 이전 챕터에서 배운 내용들을 바탕으로 OpenAI의 최신 모델들을 살펴본다. ChatGPT로 대화형 AI를 만들고 설명 가능한 AI로 활용하는 방법을 배운다. GPT-4를 활용하여 간단한 프롬프트로 k-means 클러스터링을 수행하는 프로그램도 만들어 본다. 앞서 배운 프롬프트 엔지니어링을 바탕으로 고급 프롬프트 엔지니어링 또한 알아 본다. 마지막으로 DALL-E 2를 활용하여 다양한 이미지를 만들어 볼 것이다.

부록 I, 트랜스포머 용어 설명은 스택(stack), 서브 층(sublayer), 어텐션 헤드의 전체 구조를 살펴본다.

부록 II, 트랜스포머 모델의 하드웨어 제약사항은 트랜스포머를 실행하는 CPU와 GPU를 살펴본다. 구글 코랩에서 CPU, 무료 GPU, 프로 GPU로 실험을 진행하며 GPU가 트랜스포머에 적합한 이유를 확인할 것이다.

부록 III, GPT-2를 사용한 일반 텍스트 완성에는 7장, GPT-3 엔진을 사용한 초인간 트랜스포머 등장에서 사용한 GPT-2로 일반 텍스트 완성에 대해 좀 더 자세히 설명한다.

부록 IV, GPT-2를 사용해 커스텀 텍스트 완성하기는 7장, GPT-3 엔진을 사용한 초인간 트랜스포머 등장 내용을 보완한다. GPT-2를 구축하고 학습시킨 후 커스텀 텍스트로 상호 작용한다.

부록 V, 문제 정답은 각 장 마지막에 있는 문제의 정답이다.

## 이 책을 최대한 활용하려면

이 책에 등장하는 프로그램은 대부분 코랩 노트북(Colaboratory notebook)이다. 구글의 지메일(Gmail) 무료계정만 있다면 코랩의 가상머신에서 노트북을 실행할 수 있다.

몇몇 프로그램은 파이썬이 설치된 환경이 필요하다.

시간을 들여서 2장, 트랜스포머 모델 아키텍처 살펴보기와 부록 I, 트랜스포머 용어 설명을 읽어보길 권장한다. 2장은 부록 I, 트랜스포머 용어 설명에서 설명하는 블록으로 구축한 오리지널 트랜스포머를 소개한다. 오리지널 트랜스포머는 책 전반에 걸쳐 등장할 것이다. 만약 어렵게 느껴진다면, 장에서 소개하는 직관적인 아이디어를 위주로 살펴보자. 다른 장들을 더 공부하고 트랜스포머가 익숙해진 후에 한번 더 읽어보는 것도 좋다.

각 장을 읽고 나면, 요구사항에 맞는 트랜스포머를 어떻게 구현할 수 있을지 또는 어떻게 활용하여 참신한 아이디어로 커리어를 발전시킬 수 있을지 생각해보자.

## 예제 코드 다운받기

깃허브 주소 <https://github.com/Denis2054/Transformers-for-NLP-2nd-Edition>에서 이 책의 코드를 확인할 수 있다. <https://github.com/PacktPublishing/>에도 다양한 책과 비디오와 함께 코드를 제공하고 있다.

영진닷컴 자료실(<https://youngjin.com/reader/pds/pds.asp> (영진닷컴 > 고객센터 > 부록CD 다운로드))도 다운로드가 가능하다.

## 그림 내려받기

이 책의 스크린샷과 다이어그램 이미지를 PDF로 다음 주소에서 다운받을 수 있다.  
[https://static.packt-cdn.com/downloads/9781803247335\\_ColorImages.pdf](https://static.packt-cdn.com/downloads/9781803247335_ColorImages.pdf).

# 목차

## 1장 트랜스포머란 무엇인가?

1.1 트랜스포머 생태계	020
1.2 트랜스포머로 NLP 모델 최적화	026
1.3 어떤 리소스를 사용해야 하나요?	029
1.4 정리하기	036
1.5 문제	037
1.6 참고 문헌	037

## 2장 트랜스포머 모델 아키텍처 살펴보기

2.1 트랜스포머의 시작: Attention is All You Need	041
2.2 학습과 성능	076
2.3 허깅페이스의 트랜스포머 모델	077
2.4 정리하기	078
2.5 문제	079
2.6 참고 문헌	079

## 3장 BERT 모델 미세 조정하기

3.1 BERT 아키텍처	084
3.2 BERT 미세 조정하기	093
3.3 정리하기	114
3.4 문제	115
3.5 참고 문헌	116

## 4장 RoBERTa 모델 처음부터 사전 학습하기

4.1 토큰라이저 학습하기 및 트랜스포머 사전 학습하기	120
4.2 처음부터 KantaiBERT 구축하기	122
4.3 다음 단계	144
4.4 정리하기	145
4.5 문제	146
4.6 참고 문헌	146

## 5장 트랜스포머를 사용한 다운스트림 NLP 작업

5.1 트랜스포머의 트랜스덕션과 귀납적 상속	149
5.2 트랜스포머 성능 vs 인간 기준값	154
5.3 다운스트림 작업 실행하기	167
5.4 정리하기	173
5.5 문제	174
5.6 참고 문헌	174

## 6장 트랜스포머를 사용한 기계 번역

6.1 기계 번역 정의하기	178
6.2 WMT 데이터셋 전처리하기	180
6.3 BLEU로 기계 번역 평가하기	189
6.4 구글 번역으로 번역하기	194
6.5 트랙스로 번역하기	196
6.6 정리하기	200
6.7 문제	201
6.8 참고 문헌	202

## 7장 GPT-3 엔진을 사용한 초인간 트랜스포머 등장

7.1 GPT-3 트랜스포머 모델을 사용한 초인간 NLP	206
7.2 OpenAI GPT 트랜스포머 모델의 아키텍처	207
7.3 GPT-2를 사용한 일반 텍스트 완성	215
7.4 커스텀 GPT-2 언어 모델 학습	217
7.5 OpenAI GPT-3 작업 실행하기	221
7.6 GPT-2와 GPT-3의 출력 비교하기	232
7.7 GPT-3 미세 조정하기	233
7.8 4차 산업혁명 AI 전문가의 역할	238
7.9 정리하기	240
7.10 문제	241
7.11 참고 문헌	242

## 8장 법률 및 금융 문서에 트랜스포머를 적용하여 요약하기

8.1 범용 텍스트 두 텍스트 모델 디자인하기	246
8.2 T5를 사용해 요약하기	252
8.3 GPT-3로 요약하기	266
8.4 정리하기	268
8.5 문제	269
8.6 참고 문헌	270

## 9장 데이터셋에 적합한 토큰나이저

9.1 데이터셋에 적합한 토큰나이저	274
9.2 특정 어휘가 포함된 표준 NLP 작업	290
9.3 GPT-3의 범위 살펴보기	299
9.4 정리하기	301
9.5 문제	302
9.6 참고 문헌	303

## 10장 BERT 기반 트랜스포머를 사용한 SRL

10.1 SRL(Semantic Role Labeling, 의미역 결정)	307
10.2 BERT 기반 모델을 사용한 SRL 실험	311
10.3 기본 예시	313
10.4 어려운 예시	321
10.5 SRL 적용 범위에 대한 의문	328
10.6 정리하기	333
10.7 문제	334
10.8 참고 문헌	334

## 11장 데이터가 말하게 하기: 스토리, 질문, 답변

11.1 방법론	338
11.2 방법 0: 시행착오	340
11.3 방법 1: NER	343
11.4 방법2: SRL	350
11.5 다음 단계	362
11.6 정리하기	366
11.7 문제	367
11.8 참고 문헌	367

## 12장 고객 감정을 감지해 예측하기

12.1 SST(Stanford Sentiment Treebank)	372
12.2 감성 분석으로 고객 행동 예측하기	378
12.3 GPT-3를 사용한 감성 분석	388
12.4 4차 산업 시대에 관한 몇 가지 생각	390
12.5 정리하기	398
12.6 문제	399
12.7 참고 문헌	399

## 13장 트랜스포머로 가짜 뉴스 분석하기

13.1 가짜 뉴스에 대한 감정 반응	404
13.2 가짜 뉴스에 대한 이성적 접근법	413
13.3 마치고 전에	430
13.4 정리하기	431
13.5 문제	432
13.6 참고 문헌	432

## 14장 블랙박스 트랜스포머 모델 해석하기

14.1 BertViz로 트랜스포머 시각화하기	435
14.2 LIT	441
14.3 딕셔너리 러닝을 활용한 트랜스포머 시각화	445
14.4 내부를 볼 수 없는 모델 분석하기	452
14.5 정리하기	454
14.6 문제	455
14.7 참고 문헌	455

## 15장 NLP부터 범용 트랜스포머 모델까지

15.1 모델과 생태계 선택하기	460
15.2 리포머	463
15.3 DeBERTa	466
15.4 범용 모델에서 비전 트랜스포머까지	468
15.5 확장되는 모델 세계	485
15.6 정리하기	487
15.7 문제	488
15.8 참고 문헌	489

## 16장 트랜스포머 기반 코파일럿의 등장

16.1 프롬프트 엔지니어링	494
16.2 코파일럿	505
16.3 도메인별 GPT-3 엔진	508
16.4 컴퓨터 비전	528
16.5 메타버스에서 인간과 AI 코파일럿	532
16.6 정리하기	533
16.7 문제	534
16.8 참고 문헌	534

## 17장 초인간 트랜스포머를 사용한 OpenAI의 ChatGPT와 GPT-4

17.1 ChatGPT와 GPT-4에 초인간 NLP 연동하기	538
17.2 ChatGPT API 시작하기	543
17.3 ChatGPT Plus로 코드와 주석 작성하기	547
17.4 GPT-4API 시작하기	552
17.5 고급 프롬프트 엔지니어링	561
17.6 설명 가능한 AI(XAI)	566
17.7 DALL-E 2 API 시작하기	572
17.8 모든 것을 종합하기	576
17.9 정리하기	578
17.10 문제	579
17.11 참고 문헌	579

## 부록 I 트랜스포머 용어 설명

I.1 스택	583
I.2 서브 층	583
I.3 어텐션 헤드	584

## 부록 II 트랜스포머 모델의 하드웨어 제약사항

II.1 트랜스포머의 아키텍처와 규모	587
II.2 GPU가 특별한 이유	588
II.3 GPU는 병렬 연산을 위해 설계되었다	589
II.4 GPU는 또한 행렬 곱셈을 위해 설계되었다	589
II.5 GPU를 사용하는 코드	590
II.6 구글 코랩으로 GPU 테스트하기	591
II.7 구글 코랩의 무료 CPU	592
II.8 구글 코랩의 유료 CPU	594

## 부록 III GPT-2를 사용한 일반 텍스트 완성

III.1 1단계: GPU 활성화	597
III.2 2단계: OpenAI GPT-2 저장소 복제하기	597
III.3 3단계: 요구사항 설치하기	599
III.4 4단계: 텐서플로우 버전 확인하기	599
III.5 5단계: 345M 파라미터 GPT-2 모델 다운로드하기	601
III.6 6~7단계: 중간 지침	602
III.7 7b~8단계: 모델 가져오기 및 정의하기	603
III.8 9단계: GPT-2와 상호 작용하기	605
III.9 참고 문헌	607

## 부록 IV GPT-2를 사용해 커스텀 텍스트 완성하기

IV.1 GPT-2 모델 학습하기	611
IV.2 참고 문헌	622

## 부록 V 문제 정답

찾아보기	645
------	-----

the 1990s, the number of people in the world who are illiterate has increased from 1.2 billion to 1.5 billion (UNESCO 2003).

There are a number of reasons for this increase. One of the main reasons is that the population of the world is growing rapidly. In 1990, the world population was 5.3 billion. In 2003, it was 6.1 billion (UNESCO 2003).

Another reason is that the number of people who are illiterate is increasing in many developing countries. In 1990, the number of illiterate people in developing countries was 1.1 billion. In 2003, it was 1.4 billion (UNESCO 2003).

There are a number of reasons for this increase. One of the main reasons is that the population of these countries is growing rapidly. In 1990, the population of developing countries was 4.5 billion. In 2003, it was 5.3 billion (UNESCO 2003).

Another reason is that the number of people who are illiterate is increasing in many of these countries. In 1990, the number of illiterate people in developing countries was 1.1 billion. In 2003, it was 1.4 billion (UNESCO 2003).

There are a number of reasons for this increase. One of the main reasons is that the population of these countries is growing rapidly. In 1990, the population of developing countries was 4.5 billion. In 2003, it was 5.3 billion (UNESCO 2003).

Another reason is that the number of people who are illiterate is increasing in many of these countries. In 1990, the number of illiterate people in developing countries was 1.1 billion. In 2003, it was 1.4 billion (UNESCO 2003).

There are a number of reasons for this increase. One of the main reasons is that the population of these countries is growing rapidly. In 1990, the population of developing countries was 4.5 billion. In 2003, it was 5.3 billion (UNESCO 2003).

Another reason is that the number of people who are illiterate is increasing in many of these countries. In 1990, the number of illiterate people in developing countries was 1.1 billion. In 2003, it was 1.4 billion (UNESCO 2003).

There are a number of reasons for this increase. One of the main reasons is that the population of these countries is growing rapidly. In 1990, the population of developing countries was 4.5 billion. In 2003, it was 5.3 billion (UNESCO 2003).

Another reason is that the number of people who are illiterate is increasing in many of these countries. In 1990, the number of illiterate people in developing countries was 1.1 billion. In 2003, it was 1.4 billion (UNESCO 2003).

There are a number of reasons for this increase. One of the main reasons is that the population of these countries is growing rapidly. In 1990, the population of developing countries was 4.5 billion. In 2003, it was 5.3 billion (UNESCO 2003).

Another reason is that the number of people who are illiterate is increasing in many of these countries. In 1990, the number of illiterate people in developing countries was 1.1 billion. In 2003, it was 1.4 billion (UNESCO 2003).

There are a number of reasons for this increase. One of the main reasons is that the population of these countries is growing rapidly. In 1990, the population of developing countries was 4.5 billion. In 2003, it was 5.3 billion (UNESCO 2003).

Another reason is that the number of people who are illiterate is increasing in many of these countries. In 1990, the number of illiterate people in developing countries was 1.1 billion. In 2003, it was 1.4 billion (UNESCO 2003).

There are a number of reasons for this increase. One of the main reasons is that the population of these countries is growing rapidly. In 1990, the population of developing countries was 4.5 billion. In 2003, it was 5.3 billion (UNESCO 2003).

Another reason is that the number of people who are illiterate is increasing in many of these countries. In 1990, the number of illiterate people in developing countries was 1.1 billion. In 2003, it was 1.4 billion (UNESCO 2003).

# 01장

---

## 트랜스포머란 무엇인가?

---

- 1.1 트랜스포머 생태계
- 1.2 트랜스포머로 NLP 모델 최적화
- 1.3 어떤 리소스를 사용해야 하나요?
- 1.4 정리하기
- 1.5 문제
- 1.6 참고 문헌

트랜스포머는 산업적인 단일화(homogenized)된 포스트-딥러닝(post-deep learning)<sup>1</sup> 모델로, 슈퍼 컴퓨터가 병렬로 처리할 수 있게 고안됐다. 트랜스포머의 단일화된 특성 덕분에 별도 미세 조정없이도 다양한 작업에 모델을 활용할 수 있고, 수십억 파라미터로 라벨링되지 않은 수십억 원시데이터(raw data)에 대해 자기 지도 학습(self-supervised learning)을 수행할 수 있다.

포스트 딥러닝의 이와 같은 아키텍처를 **파운데이션 모델(foundation model)**이라고 한다. 2015년도에 시작된 4차 산업혁명은 모든 것을 연결함으로써 기계 간 자동화를 실현했다. 파운데이션 모델인 트랜스포머는 4차 산업혁명을 대표한다. 인공지능 전반에서, 특히 자연어 처리(NLP, Natural Language Process)에서 전통적인 과거 소프트웨어 보다 비약적 발전을 이뤘다.

5년도 채 지나지 않아, AI는 효과적인 클라우드 서비스로 자리 잡았다. 과거와 같이 로컬(local) 환경에서 라이브러리들을 다운로드하여 개발하는 형태는 이제 교과서 연습문제에서나 보일 뿐, 현업에서는 클라우드 환경에서 개발하는 것이 일반적이다.

4차 산업 시대의 프로젝트 매니저는 OpenAI 클라우드 플랫폼에 접속하여 가입하고 API Key를 얻어 몇 분 안에 작업을 시작할 수 있다. 해당 서비스의 사용자는 원하는 NLP 작업을 선택하고 분석하고자 하는 문장을 입력하여 결과를 받아볼 수 있다. 또한, GPT-3 코덱스(Codex)와 같은 서비스를 이용해 프로그래밍 지식 없이도 애플리케이션을 만들 수 있다. 이런 상황에 태어난 신기술이 바로 프롬프트 엔지니어링(prompt engineering)이다.

물론 GPT-3가 적합하지 않은 작업도 있으며, GPT-3를 적용하기 힘들 때 프로젝트 매니저, 컨설턴트, 개발자 등은 구글 AI, **Amazon Web Services(AWS)**, 앨런 인공지능 연구소(Allen Institute for AI), 허깅페이스(Hugging Face)에서 제공하는 다른 시스템을 사용할 수 있다.

그렇다면 프로젝트 매니저는 로컬에서 개발해야 할까? 아니면 구글 클라우드(Google Cloud), 마이크로소프트 애저(Microsoft Azure), 혹은 아마존 웹 서비스(AWS) 상에서 직접 개발해야 할까? 개발팀이 허깅페이스나 구글 트랙스(Google Trax), OpenAI, AllenNLP를 선택해야 할까? 인공지능 전문가나 데이터 과학자는 실질적인 AI 개발이 필요없는 서비스형 API를 사용해야 할까?

1 역주. 포스트 딥러닝 모델은 딥러닝 이론과 기술의 발전을 바탕으로 초기 딥러닝 모델보다 더 발전된 모델을 가리킨다. 더 깊거나 넓은 신경망 구조, 보다 효율적인 학습 알고리즘, 데이터 효율성 향상을 위한 기술적 발전 등이 있다.

앞의 모든 경우가 정답이 될 수 있다. 미래에 회사나 고객들이 어떤 기능을 요구할지 모르기 때문에 어떠한 요구사항에도 대비해야 한다. 이 책은 현재 시장에 존재하는 모든 기능을 서술하지는 않지만 4차 산업 시대 NLP 문제를 해결하기 위한 충분한 솔루션을 제공한다.

이번 장에서는 먼저 트랜스포머에 대해 개략적으로 설명한다. 그리고 다양한 트랜스포머 구현 방법론을 알아야 하는 이유를 설명한다. 플랫폼, 프레임워크, 라이브러리, 언어의 경계는 시장에 존재하는 많은 자동화 기술과 API로 인해 모호해졌다.

장의 마지막 부분에서는, 임베디드 트랜스포머의 발전에 따른 4차 산업혁명에서 AI 전문가의 역할을 소개하겠다.

다양한 트랜스포머 구현체를 살펴보기 전에 이러한 중요한 개념들을 짚고 넘어갈 필요가 있다.

이 장에서는 다음 주제들을 다룬다.

- 4차 산업 혁명의 등장
- 파운데이션 모델의 패러다임 전환
- 새로운 기술, 프롬프트 엔지니어링
- 트랜스포머의 등장 배경
- 트랜스포머 구현을 위한 과제들
- 시장의 판도를 바꾼 트랜스포머 모델 API
- 트랜스포머 라이브러리 선택의 어려움
- 트랜스포머 모델 선택의 어려움
- 4차 산업 시대에서 인공지능 전문가의 새로운 역할
- 임베디드 트랜스포머

먼저 트랜스포머 생태계에 대해 알아보자.

트랜스포머 모델은 패러다임 전환을 대표하기에 이것들을 묘사할 새로운 이름이 필요했다. 바로 **파운데이션 모델**(foundation models)이다. 스탠퍼드 대학은 파운데이션 모델 연구소인 **CRFM(Center for Research on Foundation Models)**을 설립했다. 2021년 8월에 CRFM은 백 명 이상의 과학자와 전문가가 함께 이백여 페이지의 논문 「On the Opportunities and Risks of Foundation Models」를 발간했다.

파운데이션 모델은 학계가 아닌 빅테크 산업에 의해 만들어졌다. 예를 들면 구글은 트랜스포머 모델에 이어 BERT(Bidirectional Encoder Representations from Transformer)를 발명했고 마이크로소프트는 OpenAI와 협력하여 GPT-3를 만들었다.

빅테크 기업은 데이터 센터로 유입되는 페타바이트급 데이터에 대응하기 위해 더 나은 모델을 찾아야 했고, 그 결과 트랜스포머가 탄생했다.

산업화된 인공지능 모델의 필요성을 이해하기 위해, 4차 산업혁명에 대해 먼저 이야기해 보겠다.

### 1.1.1 4차 산업혁명(Industry 4.0)

농업혁명은 1차 산업혁명으로 이어졌고 그 결과 기계가 탄생했다. 2차 산업혁명에선 전기, 전화, 비행기가 탄생했고, 3차 산업혁명에선 디지털이 탄생했다.

4차 산업혁명은 봇(bots), 로봇(robots), 커넥티드 기기(connected devices), 자율 주행 자동차(autonomous cars), 스마트폰, 소셜 미디어 데이터를 수집하는 자동화봇 간의 연결들을 탄생시켰다.

결과적으로 수백만 기계와 봇들이 매일 이미지, 소리, 단어, 이벤트 등의 수십억 데이터를 생성하고 있다.

## 4차 산업혁명의 영향과 범위

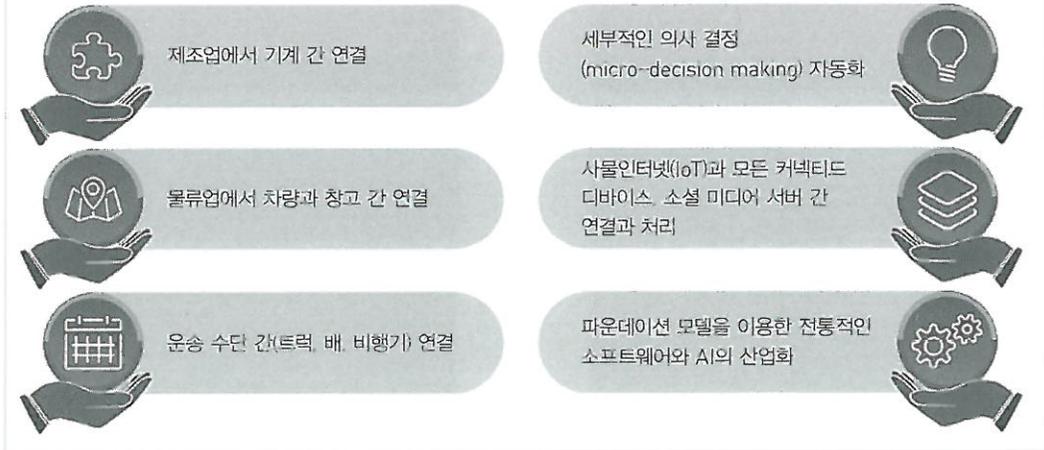


그림 1.1 4차 산업혁명의 영향과 범위

4차 산업혁명에서 역사상 가장 방대한 데이터를 처리하기 위해, 인간 개입 없이도 데이터를 처리하고 결정을 내릴 수 있는 지능형 알고리즘이 필요했고, 빅테크 기업은 과거 여러 알고리즘으로 해결하던 다양한 작업을 수행할 수 있는 단일 AI 모델을 찾아야 했다.

### 1.1.2 파운데이션 모델

트랜스포머 모델은 두 가지 고유한 특징이 있다. 바로 고차원의 단일화와 놀라운 수준의 이머전스(Emergence)<sup>2</sup>이다. “단일화”는 하나의 모델이 다양한 작업을 수행할 수 있게 만드는 특징으로, 슈퍼 컴퓨터로 수십억 파라미터를 학습하여 생긴다.

그림 1.2에서 파운데이션 모델 등장으로 인한 포스트 딥러닝 생태계의 패러다임 전환을 보여준다.

2 역주. 이머전스란 시스템 행동이 프로그래밍으로 결정되지 않고 데이터를 통해 유추되는 특징이다.

## AI의 새로운 패러다임



그림 1.2 4차 산업혁명 시대 AI 전문가의 범위

파운데이션 모델은 혁신적인 아키텍처로 AI의 역사를 쓰고 있다. 그 결과 인공지능 전문가 기술 영역은 현재에도 계속 늘어나고 있다.

트랜스포머 생태계는 지금까지의 인공지능 발전과는 다른 4가지 특징이 있다.

### ● 모델 아키텍처

각 층이 병렬 처리 시 모든 프로세스에 동일하게 동작하도록 고안하여 산업에 활용하기 적합하다. **2장, 트랜스포머 모델 아키텍처 살펴보기**에서 보다 자세하게 트랜스포머의 아키텍처를 알아볼 예정이다.

### ● 데이터

3차 산업혁명(디지털)에서 등장한 데이터는 4차 산업혁명에서 가늠할 수 없을 수준으로 방대해졌다. 빅테크 기업은 이런 거대한 데이터를 보유하고 있다.

### ● 컴퓨팅 파워

빅테크 기업은 이전에는 볼 수 없었던 규모의 컴퓨팅 파워를 보유하고 있다. 예를 들어, GPT-3는 초당 50 페타플롭스(PetaFlops) 속도로 학습되었으며, 현재 구글은 초당 80 페타플롭스를 초과하는 도메인 특화 슈퍼컴퓨터를 보유하고 있다.

### ● 프롬프트 엔지니어링

고도로 학습된 트랜스포머는 프롬프트(prompt)를 이용해 작업을 수행할 수 있다. 프롬프트에 자연어(natural language)를 입력할 수 있다. 그러나, 입력한 단어들은 어떤 구조를 필요로 하기에 프롬프트는 일종의 메타 언어(meta language)이다.

따라서, 파운데이션 모델은 슈퍼컴퓨터로 수십억 데이터를 학습한 수십억 파라미터를 가진 트랜스포머 모델이다. 그런 다음 이 모델은 추가 미세 조정(fine-tuning) 없이 다양한 작업을 수행할 수 있다. 이렇게 완전히 학습한 모델을 흔히 엔진이라고 부른다. 따라서, GPT-3, BERT, 그리고 소수 트랜스포머 엔진만이 파운데이션 모델로 인정된다.

#### NOTE

이 책에서는 파운데이션 모델은 OpenAI GPT-3 또는 BERT 모델만을 가리킨다. 그 이유는 GPT-3와 BERT가 슈퍼컴퓨터에서 완전히 학습됐기 때문이다. 다른 모델이 제한적인 용도에서 흥미롭고 효과적일지라도, 리소스 부족으로 파운데이션 모델의 단일화 수준에 도달하지 못하기 때문이다.

이제 파운데이션 모델이 어떻게 동작하고 프로그램 개발 방식을 어떻게 변화시켰는지 살펴보자.

### 1.1.2.1 프로그래밍이 NLP의 하위 영역이 되고 있나?

2021년 8월 “첸(Chen et al.)”은 자연어를 소스 코드로 변환할 수 있는 GPT-3 코텍스에 관한 기막힌 논문을 발표했다. 코텍스는 깃허브 공개 레포지토리 5400만 개를 학습했다. 코텍스는 소스 코드에 흥미로운 자연어를 추가할 수 있다. 이는 16장, **트랜스포머 기반 코파일럿의 등장**에서 살펴보자.

이제 프로그래밍은 자연어를 소스 코드로 번역하는 작업이 된 것인가?

프로그래밍이 GPT-3 엔진 사용을 위한 NLP 작업이 되고 있나?

질문에 답하기 앞서 한 예시를 살펴보자.

코텍스는 확률적 알고리즘으로 메타 언어는 까다롭다는 점을 유의하자. 프롬프트를 올바르게 설계하지 않으면 예상한 결과를 얻을 수 없다.

필자는 코텍스를 실험하면서 몇 가지 프롬프트를 만들었다. 이 예는 코텍스 동작 원리 아이디어를 제공할 순전히 교육 목적으로 작성됐다.

필자의 프롬프트는 다음과 같다.

- “generate a random distribution of 200 integers between 1 and 100” in Python (파이썬으로 1에서 100 사이의 무작위 정수 200개를 생성하라)
- “plot the data using matplotlib”(matplotlib로 데이터를 그려라)
- “create a k-means clustering model with 3 centroids and fit the model” (3개의 중심점으로 k-means 클러스터링 모델을 학습시켜라)
- “print the cluster labels”(클러스터 라벨을 출력하라)
- “plot the clusters”(클러스터를 시각화하라)
- “plot the clusters with centroids”(클러스터를 중심과 함께 시각화하라)

코덱스는 필자의 메타 언어를 파이썬 소스 코드로 번역해 주었다.

코덱스는 확률적 모델이므로 시도할 때마다 정확히 동일한 소스 코드를 재현하지 않을 수 있다. 여러 분은 여러 차례 시도해 자유자재로 구사할 때까지 메타 언어를 배워야 할 것이다.

다음 코드는 자동으로 생성된 것이며 복사하여 테스트할 수 있다.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets.samples_generator import make_blobs

# 무작위 데이터 생성
np.random.seed(0)
X, y = make_blobs(n_samples=200, centers=3, n_features=2, cluster_std=2,
random_state=0)

# 데이터를 그래프로 표현
plt.scatter(X[:, 0], X[:, 1], s=50)
plt.show()

# k-means 모델 생성
kmeans = KMeans(n_clusters=3, random_state=0)

# 데이터로 모델 학습
kmeans.fit(X)

# 클러스터 라벨 출력
print(kmeans.labels_)

# 클러스터 시각화
plt.scatter(X[:, 0], X[:, 1], c=kmeans.labels_, cmap='rainbow')
plt.show()

# 클러스터와 중심점을 시각화
plt.scatter(X[:, 0], X[:, 1], c=kmeans.labels_, cmap='rainbow')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
c='black', s=100, alpha=0.5)
plt.show()
```

여러분은 앞의 코드를 복사하고 붙여 넣어 동작을 확인할 수 있다.<sup>3</sup> 자바스크립트나 다른 언어로 실험을 할 수도 있다.

깃허브 코파일럿(GitHub Copilot)은 마이크로소프트 개발 도구를 이용해 현재 사용할 수 있다. 이는 **16장. 트랜스포머 기반 코파일럿의 등장**에서 살펴보자. 여러분이 프롬프트 엔지니어링을 위한 메타 언어를 익힌다면 개발 시간을 단축할 수 있다.

최종 사용자가 메타 언어를 마스터하면 프로토타입이나 작은 작업들을 만들 수 있다. 미래에는 코딩 코파일럿 사용이 확대될 것이다.

인공지능의 미래에 코덱스가 어떤 역할을 할 수 있을지 16장에서 확인해 볼 수 있다.

여기서는 인공지능 전문가의 밝은 미래를 조망해 보자.

### 1.1.2.2 인공지능 전문가의 미래

파운데이션 모델의 사회적 영향력을 과소평가해서는 안된다. 프롬프트 엔지니어링은 인공지능 전문가에게 필요한 기술이 되었다. 그러나 인공지능 전문가의 미래는 트랜스포머에만 국한될 수는 없다. 4차 산업혁명에서 인공지능과 데이터 과학은 공통점이 있다.

AI 전문가는 고전적인 AI, IoT, 에지 컴퓨팅(edge computing) 등을 사용하는 기계 간(machine-to-machine) 알고리즘에 참여하게 될 것이다. 또한 고전적인 알고리즘을 사용하는 붓, 로봇, 서버 및 모든 유형의 커넥티드 기기 간의 흥미로운 연결을 설계하고 개발할 것이다.

이 책은 프롬프트 엔지니어링에 국한하지 않고 4차 산업혁명 인공지능 전문가가 필요로 하는 광범위한 설계 기술을 다루고 있다. 프롬프트 엔지니어링은 AI 전문가가 익혀야 할 설계 기술의 부분 집합이다. 이 책에서 미래 인공지능 전문가는 4차 산업혁명 인공지능 전문가를 가리킨다.

이제 트랜스포머가 NLP 모델을 최적화하는 방법을 알아보자.

3 역주:sklearn.datasets.samples\_generator"는 scikit-learn 0.22에서 제거됐다. 해당 코드를 실행하기 위해선 "from sklearn.datasets import make\_blobs"와 같이 수정이 필요하다.

## 1.2

# 트랜스포머로 NLP 모델 최적화

수십 년 동안 NLP 시퀀스 모델에 LSTM을 포함한 순환 신경망(RNN, Recurrent Neural Networks)을 적용해 왔다. 그러나 긴 시퀀스와 많은 파라미터에 직면하면 순환 신경망의 순환부는 한계에 도달한다. 따라서 첨단 트랜스포머가 대세가 됐다.

이 섹션에서 트랜스포머로 이어진 NLP의 간략한 배경을 살펴보자. 트랜스포머 모델 아키텍처는 2장, 트랜스포머 모델 아키텍처 살펴보기에서 살펴보자. 먼저 NLP 신경망의 RNN 층을 대체한 트랜스포머의 어텐션 헤드(attention head)를 직관적으로 살펴보자.

트랜스포머 핵심 개념을 대략적으로 요약하면 “토큰을 섞는 것”이라 할 수 있다. NLP 모델은 먼저 단어 시퀀스를 토큰으로 변환한다. RNN은 토큰들을 순환부(recurrent functions)에서 분석한다. 트랜스포머는 각 토큰을 시퀀스에 존재하는 다른 토큰들과 연관시킨다. 그림 1.3을 보자.

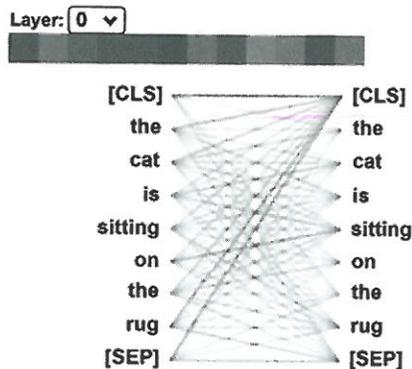


그림 1.3 트랜스포머 층의 어텐션헤드

2장에서 어텐션 헤드를 자세히 살펴볼 것이다. 그림 1.3의 핵심은 시퀀스의 각 단어(토큰)는 동일 시퀀스의 다른 모든 단어와 연관되어 있다는 점이다. 이 모델이 4차 산업혁명인 NLP로 가는 문을 열어준다.

트랜스포머의 배경에 대해 간단히 살펴보자.

## 1.2.1 트랜스포머의 배경

많은 천재들이 지난 100년 이상 동안 시퀀스 패턴 및 언어 모델링을 연구해 왔다. 그 결과 기계는 가능한 단어 시퀀스를 예측하는 방법을 점진적으로 학습할 수 있었다. 이를 가능하게 한 거인들을 모두 인용하려면 책 한 권 분량이 될 것이다.

이 섹션에서는 트랜스포머 등장 근거를 소개하기 위해 필자가 가장 좋아하는 연구자들을 독자에게 소개하겠다. 20세기 초, 안드레이 마르코프(Andrey Markov)는 무작위 값(random values) 개념을 도입하고 확률적 프로세스(stochastic processes) 이론을 만들었다. AI에서 우리는 이를 **마르코프 의사 결정 과정(Markov Decision Process, MDP)**, **마르코프 체인(Markov Chains)**, **마르코프 프로세스(Markov Processes)**라 부른다. 20세기 초, 마르코프는 시퀀스의 마지막 요소만으로 다음 요소 예측이 가능하다는 것을 보였다. 그는 수천 개의 문자들을 포함하는 과거 시퀀스 데이터셋에 자신의 기법을 적용하여 문장의 다음 문자를 예측했다. 기억할 점은 그는 컴퓨터가 없었음에도 오늘날 인공지능 분야에서 여전히 사용하는 이론을 증명했다는 점이다.

1948년, 클로드 섀넌(Claude Shannon)이 「통신의 수학적 이론(The Mathematical Theory of Communication)」을 발간했다. 클로드 섀넌은 소스 인코더(source encoder), 수신기(transmitter), 시맨틱 디코더(semantic decoder) 기반의 통신 모델 근거를 마련했다. 그는 오늘날 우리가 알고 있는 정보 이론(Information Theory)을 창안했다.

1950년, 앨런 튜링(Alan Turing)이 「컴퓨팅 기계와 지능(Computing Machinery and Intelligence)」 논문을 발간했다. 앨런 튜링은 2차 세계대전 중 독일군의 암호화된 메시지를 해독한 튜링 머신의 성공을 바탕으로 기계 지능에 대해 설명했다. 메시지는 단어와 숫자의 시퀀스로 이뤄져 있었다.

1954년, 조지타운-IBM 실험에서 컴퓨터로 규칙 시스템(rule system)을 사용한 번역 실험을 했다. 러시아어 문장을 영어로 번역한 실험이었다. 규칙 시스템은 언어 구조를 분석하는 규칙 목록을 실행하는 프로그램이다. 오늘날에도 규칙 시스템은 어디에나 존재한다. 그러나 경우에 따라서는 인공지능이 수십억 개의 언어 조합에 대한 규칙 목록을 자동으로 학습하여 규칙 목록을 대체할 수 있다.

1956년, 기계가 학습할 수 있다는 사실이 밝혀지면서 존 매카시(John McCarthy)가 “인공지능”이라는 표현을 처음 사용했다.

1982년, 존 홉필드(John Hopfield)는 홉필드 네트워크(Hopfield networks) 또는 “연관”(associative) 신경망으로 알려진 RNN을 소개했다. 존 홉필드는 1974년 수십 년간 학습 과정 이론적 토대를 마련한 『뇌에 지속적 상태 존재성(The existence of persistent states in the brain)』을 저술한 W.A. Little로부터 영감을 받았다. RNN은 진화했고 오늘날 우리가 알고 있는 LSTM이 등장했다.

그림 1.4에서 볼 수 있듯이 RNN은 시퀀스의 지속 상태(persistent states)를 효율적으로 기억한다.

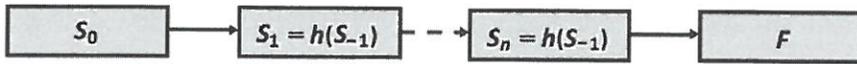


그림 1.4 RNN 과정

각 상태  $s_n$ 은  $s_{n-1}$ 의 상태를 포착한다. 네트워크의 끝에 도달하면 함수  $F$ 가 트랜스덕션(transduction)<sup>4</sup>, 모델링, 그리고 기타 시퀀스 기반 작업을 수행할 것이다.

1980년대에 얀 르쿤(Yann LeCun)은 다목적 컨볼루션 신경망(Convolutional Neural Network, CNN)을 설계했다. 그는 텍스트 시퀀스에 CNN을 적용했다. 오늘날 시퀀스 트랜스덕션 및 모델링에도 CNN이 적용된다. CNN 또한 정보를 계층별로 처리하는 W.A Little의 지속적 상태를 기반으로 한다. 1990년대에 얀 르쿤은 수년간의 작업을 요약하여 오늘날 우리가 알고 있는 많은 CNN 모델을 낳은 LeNet-5를 개발했다. 그러나 CNN의 효율적인 아키텍처는 길고 복잡한 시퀀스에서 장기 의존성<sup>5</sup> 처리 시 문제가 발생한다.

그 외 AI 전문가라면 누구나 인정할 만한 훌륭한 이름, 논문, 모델을 더 언급할 수 있다. 지난 몇 년 동안 AI 분야에서 모두가 올바른 방향으로 나아가고 있는 것처럼 보였다. 마르코프 필드, RNN, CNN은 여러 다른 모델로 발전했다. 마지막 토큰뿐만 아니라 시퀀스 내의 다른 토큰도 살펴보는 어텐션 개념이 등장했다. 어텐션 개념은 RNN과 CNN 모델에 추가됐다.

그 후, AI 모델이 컴퓨팅 성능을 더 요구하는 긴 시퀀스를 분석해야 하는 경우 AI 개발자는 더 강력한 컴퓨터를 사용하고 파라미터를 최적화하는 방법을 찾았다.

시퀀스-투-시퀀스(sequence-to-sequence) 모델 연구가 일부 있었지만 기대에 미치지 못했다.

더 이상 진전을 이루기 위해 할 수 있는 일은 없는 것처럼 보였다. 그렇게 30년이 흘렀다. 그러다 2017년 말부터 산업화된 어텐션 헤드 서브 층 등을 갖춘 최첨단 트랜스포머가 등장했다. RNN은 더 이상 시퀀스 모델링에 꼭 필요해 보이지 않았다.

**2장, 트랜스포머 모델 아키텍처 살펴보기**에서 다들 초기 트랜스포머 아키텍처에 대해 자세히 알아보기 전에, 트랜스포머 모델을 학습하고 구현하는 데 사용해야 하는 소프트웨어 리소스의 패러다임 변화를 간략하게 살펴보자.

4 역주. 이 책에서 등장하는 트랜스덕션은 두 가지를 의미한다. 첫째로, 시퀀스를 다른 시퀀스로 변환하는 것을 의미한다. 둘째로, 인지하는 구조를 패턴 등으로 표현하는 것을 말한다. 여기서는 첫 번째 의미의 트랜스덕션을 가리킨다. 두 번째 의미의 트랜스덕션은 5.1.1에서 등장한다.

5 역주. 장기 의존성 문제(the problem of long-term dependencies)란, 길고 복잡한 입력 시퀀스를 처리할 때 시퀀스 뒷부분에서 시퀀스의 앞부분 정보를 잃는 문제를 가리킨다. 즉, 장기 의존성을 잃는 문제를 가리킨다.

## 1.3

# 어떤 리소스를 사용해야 하나요?

4차 산업혁명 AI는 클라우드 플랫폼, 프레임워크, 라이브러리, 프로그래밍 언어, 모델 간 경계를 허물었다. 트랜스포머는 새로운 개념이며, 생태계의 범위와 수는 놀라운 수준이다. 구글 클라우드는 바로 사용할 수 있는 트랜스포머 모델을 제공한다.

OpenAI는 프로그래밍이 거의 필요 없는 “트랜스포머” API를 배포했다. 허깅페이스는 수많은 클라우드 라이브러리 서비스를 제공한다.

이 챕터에서는 이 책에서 구현할 몇 가지 트랜스포머 생태계에 대한 대략적인 분석을 살펴본다.

자연어 처리용 트랜스포머를 구현하기 위한 리소스 선택은 매우 중요하다. 이는 프로젝트의 생존이 걸린 문제다. 실제 면접, 발표를 상상해 보라. 독자의 미래나 현재 고용주, 팀, 고객과 대화하고 있다고 상상해 보라.

예를 들어 허깅페이스가 포함된 멋진 파워포인트로 발표한다고 가정하자. 한 매니저가 “죄송하지만 우리는 이런 유형의 프로젝트에는 허깅페이스가 아닌 구글 트랙스를 사용해야 합니다. 혹시 구글 트랙스를 구현해 주시겠어요?” 라며 부정적인 반응을 보일 수 있다. 구현할 수 없다고 말하면, 게임은 끝난 것이다.

매니저가 구글 트랙스 전문가인 당신에게 API만 있는 OpenAI GPT-3 엔진 사용을 요구할 수도 있다. 반대로 OpenAI GPT-3 엔진 API 전문가인데, 허깅페이스의 AutoML API를 선호하는 매니저를 마주할 수 있다. 가장 최악은 매니저가 독자의 솔루션을 받아들였지만, 결국 해당 프로젝트의 NLP 작업에는 작동하지 않는 것이다.

### TIP

명심해야 할 점은 자신이 좋아하는 솔루션에만 집중하면 언젠가는 침몰할 가능성이 크다는 것이다.

마음에 드는 시스템이 아닌, 필요한 시스템에 집중해야 한다.

이 책은 시중에 나와 있는 모든 트랜스포머 솔루션을 설명하기 위한 책이 아니다. 대신, NLP 프로젝트에서 직면하는 모든 상황에 유연하게 적용할 수 있도록 충분한 트랜스포머 생태계를 설명하는 것을 목표로 한다.

이 섹션에서 독자가 직면하게 될 몇 가지 챌린지를 살펴보자. 먼저 API부터 시작하자.

### 1.3.1 트랜스포머 4.0 끊임 없는(seamless) API의 등장

오늘날 우리는 인공지능의 산업화 시대에 접어들었다. 마이크로소프트, 구글, 아마존 웹 서비스(AWS), IBM 등은 어떤 개발자, 개발자 팀도 능가할 수 없는 AI 서비스를 제공한다. 빅테크 기업은 트랜스포머 모델과 일반적인 AI 모델을 학습하기 위해 방대한 데이터셋을 갖춘 수백만 달러짜리 슈퍼컴퓨터를 보유하고 있다.

빅테크 기업은 이미 클라우드 서비스를 사용하는 다양한 기업 고객을 보유하고 있다. 따라서, 기존 클라우드 아키텍처에 트랜스포머 API를 추가하는 데 다른 어떤 솔루션보다 적은 노력이 든다.

소규모 회사나 개인도 개발에 거의 투자하지 않고 API를 통해 가장 강력한 트랜스포머 모델 사용이 가능하다. 인턴도 며칠이면 API를 구현할 수 있다. 이러한 간단한 구현에 박사 학위자나 엔지니어가 필요 없다.

OpenAI 플랫폼은 이제 시장에서 가장 효과적인 트랜스포머 일부 모델 API를 SaaS(Software as a Service) 형태로 제공한다.

OpenAI 트랜스포머 모델은 매우 효과적이고 인간과 유사하기에 지금은 정책상 사용을 위해 먼저 요청 양식을 작성해야 한다.<sup>6</sup> 일단 요청이 수락되면 사용자는 자연어 처리 세계로 진입할 수 있다.

사용자를 놀라게 하는 OpenAI API의 간소한 사용 방법은 다음과 같다.

1. 한 번의 클릭으로 API Key을 얻는다.
2. Jupyter notebook에서 소스 코드 한 줄로 OpenAI를 불러온다.
3. 프롬프트에 원하는 NLP 작업을 입력한다.
4. 요청 작업의 토큰 완성(completion) 응답을 받는다.

이게 전부다! 4차 산업혁명에 온 걸 환영한다!

소스 코드 솔루션에 집중했던 3차 산업혁명 속 개발자는 여러 분야를 넘나드는 4차 산업혁명 개발자로 진화할 것이다.

#### NOTE

3차 산업혁명 개발자가 해왔던 트랜스포머 모델에 직관적으로 무엇을 해야 하는지 알려주는 방식이 아닌, 트랜스포머 모델에 무엇을 기대하는지 보여주는 방식을 설계하는 법을 4차 산업혁명 개발자는 배울 것이다. 7장, GPT-3 엔진을 사용한 초인간 트랜스포머 등장에서 GPT-2 및 GPT-3 모델을 통해 이 새로운 접근 방식을 살펴볼 것이다.

<sup>6</sup> 역주. 2024년 3월 기준 OpenAI API를 사용하기 위해서는 결제 정보(payment methods)를 입력하면 사용할 수 있다.

AllenNLP는 트랜스포머 온라인 교육 인터페이스를 무료로 제공한다. AllenNLP는 노트북에 설치할 수 있는 라이브러리도 제공한다. 예를 들어 상호참조해결(coreference resolution) 구현 요청을 받았다고 해보자. 온라인에서 예제를 실행하는 것으로 시작할 수 있다.

상호참조해결 작업은 그림 1.5에 표시된 문장과 같이 단어가 참조하는 개체를 찾는 작업을 의미한다.

## Document

A user visited the AllenNLP website, tried a transformer model, and found it interesting.

Run Model

그림 1.5 NLP작업 온라인으로 실행하기

단어 “it”은 웹사이트 또는 트랜스포머 모델을 가리킬 수 있다. 이 경우 BERT(Bidirectional Encoder Representations from Transformers) 같은 모델은 “it”이 트랜스포머 모델(a transformer model)을 참조하는 것으로 결과를 반환한다. AllenNLP는 그림 1.6과 같이 형식화된 출력을 제공한다.

## Model Output

A user visited the AllenNLP website, tried 0 a transformer model , and found 0 it interesting.

그림 1.6 AllenNLP 트랜스포머 모델의 출력

<https://demo.allennlp.org/coreference-resolution><sup>7</sup>에서 이 예제를 실행할 수 있다. 트랜스포머 모델이 지속적으로 업데이트되므로 독자가 실행하는 시점에는 다른 결과를 얻을 수도 있다.

API는 많은 요구 사항을 충족시킬 수도 있지만 한계도 존재한다. 다목적 API는 모든 작업에 적합할 수 있지만 특정 자연어 처리 작업에는 충분하지 않을 수 있다. 트랜스포머를 사용해 번역하는 작업은 쉬운 일이 아니다. 이 경우, 개발자, 컨설턴트, 혹은 프로젝트 관리자는 API만으로는 필요한 특정 NLP 작업을 해결할 수 없음을 증명해야 한다. 적합한 라이브러리를 찾아야 한다.

7 역주. allennlp 데모의 서비스가 종료되었다. 대신 <https://corener-demo.aiola-lab.com>의 Coreference Resolution 항목에서 테스트를 수행할 수 있다.

### 1.3.2 바로 사용할 수 있는 API 기반 라이브러리 선택하기

이 책에서는 몇 가지 라이브러리를 살펴볼 것이다. 예를 들어, 구글은 세계에서 가장 진보된 AI 연구소를 보유하고 있다. 구글 트랙스는 구글 코랩에서 단 몇 줄로 설치할 수 있다. 무료 또는 유료 서비스를 선택할 수 있다. 구글 클라우드나 보유하고 있는 서버에서 직접 소스 코드를 작성하고 모델을 수정하고 학습까지도 가능하다. 예를 들어, 바로 사용할 수 있는 API에서 한 단계 더 나아가 번역 작업용 트랜스포머 모델을 커스터마이징할 수 있다.

경우에 따라서 API 사용과 커스터마이징 과정은 교육적이고 효과적인 수가 있다. 6장, 트랜스포머를 사용한 기계 번역에서 번역 분야에서 구글의 최근 발전과 구글 트랙스 구현에 대해 살펴보자.

OpenAI와 같은 API는 약간의 개발 기술을 필요로 하고, 구글 트랙스와 같은 라이브러리는 코드를 좀 더 깊이 파고들어야 한다는 것을 알 수 있었다. 두 가지 접근 방식 모두 API 수정 측면에서 더 많은 개발을 필요로 하지만 트랜스포머 구현에는 훨씬 적은 노력이 필요하다는 것을 보여준다.

트랜스포머를 사용하는 가장 유명한 온라인 애플리케이션 중 하나는 구글 번역이다. 구글 번역은 온라인 또는 API로 사용할 수 있다.

구글 번역을 사용해 상호참조해결이 필요한 문장을 영어에서 한국어로 번역해 보자.

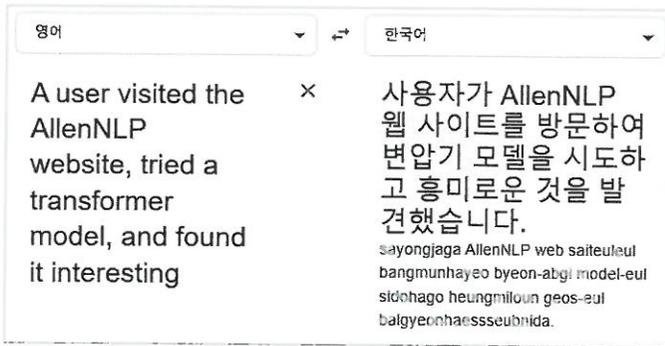


그림 1.7 구글 번역과 상호참조해결

구글 번역이 상호참조해결을 잘 해결한 것으로 보이지만, 단어 “transformer”를 “변압기”로 번역했다. 여기서의 “transformer”는 “트랜스포머”라는 신조어를 의미한다. 특정 프로젝트는 인공지능 전문가에게 언어적인 능력을 요구한다. 이 경우는 엄청난 개발 능력보다 번역기에 적절한 맥락(context)을 넣어 주는 것이 더 중요하다.

앞 예는 맥락 입력 작업을 위해 언어학자와 팀을 구성하거나 언어 능력을 습득해야 할 수도 있음을 보여준다. 또한 맥락 입력 인터페이스로 입력을 개선하려면 많은 개발이 필요할 수 있다.

따라서 구글 번역을 사용하기 위해 소스 코드를 추가하는 번거로운 일이 따를 수 있다. BERT, T5, 또는 기타 모델과 같이 특정 번역 요구 사항에 맞는 트랜스포머 모델을 찾아야 할 수도 있다. 이 책에서는 이러한 모델을 살펴볼 것이다.

솔루션의 종류가 다양해지면서 모델 선택이 쉽지 않게 됐다.

### 1.3.3 트랜스포머 모델 선택하기

빅테크 기업이 NLP 시장을 장악하고 있다. 구글, 메타, 마이크로소프트는 하루에 수십억 개 NLP 루틴(routines)을 실행하여 막강한 성능을 가진 AI 모델을 개선하고 있다. 이러한 거대 기업은 이제 광범위하게 트랜스포머 모델을 제공하며 최고 수준 파운데이션 모델을 보유하고 있다.

하지만 방대한 NLP 시장에 주목한 소규모 회사들도 이 시장에 뛰어들었다. 허깅페이스는 현재 무료 또는 유료 서비스를 제공한다. 마이크로소프트와 구글 연구소가 수십억 달러를 쏟아부어 달성한 수준에 허깅페이스가 도달하기는 어려운 것이다. 파운데이션 모델의 시작점은 GPT-3 또는 구글 BERT와 같은 슈퍼컴퓨터에서 완전히 학습한 트랜스포머이다.

허깅페이스는 다른 접근 방식을 갖고 있다. 작업에 맞는 다양한 트랜스포머를 제공하는데, 이는 흥미로운 철학이다. 허깅페이스는 유연한(flexible) 모델을 제공한다. 게다가 허깅페이스는 개발자가 제어 가능한 쉬운 API를 제공한다. 이 책의 여러 챕터에서 교육용 도구이자 특정 작업의 솔루션 후보로서 허깅페이스를 살펴볼 것이다.

지금까지 OpenAI는 전 세계적으로 가장 강력한 소수의 트랜스포머 엔진에 집중해 왔으며, 인간 수준의 많은 NLP 작업을 수행할 수 있다. 7장, GPT-3 엔진을 사용한 초인간 트랜스포머 등장에서 OpenAI의 GPT-3 엔진의 강력한 성능을 확인할 수 있다.

이러한 상반되고 종종 상충하는 전략은 우리에게 수많은 구현 방법을 남긴다. 따라서 4차 산업혁명에서 인공지능 전문가의 역할을 정의해야 한다.

### 1.3.4 4차 산업혁명에서 인공지능 전문가의 역할

4차 산업혁명은 모든 것을 모든 곳에서 연결한다. 기계가 다른 기계와 직접 통신한다. AI 기반 IoT 신호는 사람의 개입 없이 자동으로 의사 결정을 내린다. NLP 알고리즘은 자동화된 보고서, 요약, 이메일, 광고 등을 전송한다.

인공지능 전문가가 트랜스포머 모델 구현을 포함하여 점점 더 자동화되는 새로운 시대에 적응해야 한다. 인공지능 전문가는 새로운 능력을 갖게 될 것이다. 인공지능 전문가가 수행해야 하는 트랜스포머 NLP 작업을 나열해 보면 일부 간단한 작업은 인공지능 전문가의 개발이 거의 또는 전혀 필요하지 않은 것으로 보인다. AI 전문가는 설계 아이디어, 설명 및 구현을 제공하는 전문가가 될 수 있다.

#### NOTE

인공지능 전문가에게 트랜스포머가 무엇을 의미하는지에 대한 실용적인 정의는 생태계에 따라 달라질 수 있다.

몇 가지 예를 살펴보자.

- **API:** OpenAI API는 AI 개발자가 필요하지 않다. 웹 디자이너가 입력 양식을 만들면 언어학자나 주제별 전문가(Subject Matter Expert, SME)가 프롬프트 입력 텍스트를 준비할 수 있다. AI 전문가의 주요 역할은 GPT-3 엔진에 작업을 수행하는 방법을 알려주는 것이 아니라 보여줄 수 있는 언어적 기술이 필요하다. 예를 들어, 보여줄 수 있는 언어적 기술로는 입력의 맥락에 대한 작업이 포함된다. 이 새로운 작업을 프롬프트 엔지니어링이라고 부린다. 프롬프트 엔지니어는 AI분야에서 미래가 밝다.
- **라이브러리:** 구글 트랙스 라이브러리는 바로 사용할 수 있는 모델로 시작하기 위해 약간의 개발을 요구한다. 언어학 및 NLP 작업을 숙달한 AI 전문가는 데이터셋과 출력에 대한 작업을 할 수 있다.
- **학습과 미세 조정:** 미세 조정작업 중 일부는 API와 라이브러리를 모두 제공하므로 약간의 개발을 필요로 한다. 그러나 어떤 경우에는 개발을 많이 해야 할 때도 있다. 이 경우 모델을 학습하고, 미세 조정하고, 올바른 하이퍼파라미터를 찾으려면 인공지능 전문 지식이 필요하다.
- **개발 수준의 기술:** 9장, 데이터셋에 적합한 토큰나이저에서 설명하겠지만, 일부 프로젝트에서는 토큰나이저와 데이터셋이 일치하지 않을 수 있다. 이 경우, 예를 들어 언어학자와 함께 일하는 인공지능 개발자가 중요한 역할을 할 수 있다. 따라서 전산언어학(computational linguistics) 학습은 매우 유용할 수 있다.

최근 NLP AI의 진화는 “임베디드 트랜스포머”라 할 수 있다. 이는 AI 개발 생태계를 혼란에 빠뜨리고 있다.

- 예를 들어, GPT-3 트랜스포머는 현재 여러 마이크로소프트 Azure 애플리케이션에 깃허브 코파일럿과 함께 내장되어 있다. 이번 챕터의 파운데이션 모델 섹션에서 소개한 것처럼, 코텍스는 16장, 트랜스포머 기반 코파일럿의 등장에서 살펴볼 예이다.
- 임베디드 트랜스포머는 직접 접근할 수는 없지만 자동 코드 생성과 같은 자동 개발 지원을 제공한다.
- 최종 사용자는 텍스트 완성 지원 기능으로 임베디드 트랜스포머를 원활하게 사용할 수 있다.

**NOTE**

GPT-3 엔진에 직접 접근하려면 먼저 OpenAI 계정을 만들어야 한다. 그런 다음 API를 사용하거나 OpenAI 사용자 인터페이스에서 예제를 직접 실행할 수 있다.

16장에서 이 매력적인 임베디드 트랜스포머의 새로운 세계를 살펴볼 것이다. 16장을 최대한 활용하려면 먼저 이전 장의 개념, 예제 및 프로그램을 숙지해야 한다.

4차 산업혁명 AI 전문가는 유연성, 여러 분야에 걸친 지식, 그리고 무엇보다도 융통성이 필요하다. 이 책은 시장의 인공지능 전문가가 새로운 패러다임에 적응할 수 있도록 다양한 트랜스포머 생태계를 제공한다.

2장에서 초기 트랜스포머 아키텍처를 살펴보기 전에 이번 장의 내용을 정리해 보자.

## 1.4

## 정리하기

4차 산업혁명으로 인해 인공지능은 엄청난 진화를 거듭하고 있다. 3차 산업혁명은 디지털이었다. 4차 산업혁명은 모든 것을 모든 곳에 연결하는 디지털 혁명을 기반으로 한다. 자동화된 프로세스가 자연어 처리 등 중요한 영역에서 인간의 의사 결정을 대체하고 있다.

RNN은 빠르게 변화하는 세상이 요구하는 자동화된 NLP 작업의 발전 속도를 따라가지 못하는 한계가 있었다. 트랜스포머가 그 공백을 메웠다. 기업은 4차 산업혁명의 도전 과제 해결을 위해 요약, 번역과 같은 많은 NLP 도구가 필요하다.

4차 산업혁명은 인공지능 산업화 시대에 박차를 가하고 있다. 플랫폼, 프레임워크, 언어, 모델 개념의 진화는 4차 산업혁명 개발에게는 도전 과제이다. 파운데이션 모델은 추가 학습이나 미세 조정없이 다양한 작업을 수행할 수 있는 단일화된 모델을 제공함으로써 3차 산업혁명과 4차 산업혁명 사이를 이어준다.

예를 들어 AllenNLP와 같은 웹사이트는 설치 없이도 교육용 NLP 작업을 제공할 뿐만 아니라 맞춤형 프로그램으로 트랜스포머 모델을 구현할 수 있는 리소스도 제공한다. OpenAI는 몇 줄의 코드만으로 강력한 GPT-3 엔진을 실행할 수 있는 API를 제공한다. 구글 트랙스는 엔드-투-엔드(end-to-end) 라이브러리를 제공하며, 허깅페이스는 다양한 트랜스포머 모델과 구현을 제공한다. 이 책에서는 이러한 생태계를 살펴볼 것이다.

4차 산업혁명은 광범위한 기술을 갖춘 점에서 이전의 AI와 근본적으로 다르다. 예를 들어, 프로젝트 관리자는 웹 디자이너에게 프론트엔지니어링을 통해 OpenAI의 API용 인터페이스를 만들어 달라고 요청하여 트랜스포머를 구현하기로 결정할 수 있다. 또는 필요한 경우 프로젝트 관리자는 인공지능 전문가에게 구글 트랙스(Trax)나 허깅페이스를 다운로드하여 맞춤형 트랜스포머 모델로 본격적인 프로젝트를 개발하도록 요청할 수 있다.

4차 산업혁명은 개발자의 역할이 확장되고 프로그래밍보다 설계 능력을 필요하도록 판도를 바꿀 것이다. 또한 임베디드 트랜스포머는 코드 개발 및 사용을 지원할 것이다. 이러한 새로운 기술셋은 하나의 도전이지만 새롭고 흥미로운 지평을 열어준다.

**2장, 트랜스포머 모델 아키텍처 살펴보기**에서는 오리지널 트랜스포머 아키텍처에 대해 살펴보자.

## 1.5 문제

01. 우리는 3차 산업혁명 시대에 살고 있다. 참 / 거짓
02. 4차 산업혁명은 모든 것을 연결한다. 참 / 거짓
03. 4차 산업혁명 개발자는 때로는 해야 할 AI 개발이 없다. 참 / 거짓
04. 4차 산업혁명 개발자는 트랜스포머 구현을 처음부터 해야 할 수도 있다. 참 / 거짓
05. 허깅페이스와 같은 트랜스포머 생태계를 두 개 이상 익힐 필요가 없다. 참 / 거짓
06. 바로 사용할 수 있는 트랜스포머 API는 모든 요구 사항을 충족한다. 참 / 거짓
07. 회사는 개발자가 가장 잘 알고 있는 트랜스포머 생태계를 채택할 것이다. 참 / 거짓
08. 클라우드 트랜스포머는 주류가 되고 있다. 참 / 거짓
09. 트랜스포머 프로젝트는 노트북(laptop)에서 실행될 수 있다. 참 / 거짓
10. 4차 산업혁명 인공지능 전문가는 좀 더 유연해질 것이다. 참 / 거짓

## 1.6 참고 문헌

- 2021, 파운데이션 모델의 기회와 위험에 대해, <https://arxiv.org/abs/2108.07258>
- 2021, 소스 코드를 학습한 대규모 언어 모델 평가, <https://arxiv.org/abs/2107.03374>
- 마이크로소프트 AI: <https://innovation.microsoft.com/en-us/ai-at-scale>
- OpenAI: <https://openai.com/>
- 구글 AI: <https://ai.google/>
- 구글 트랙스: <https://github.com/google/trax>
- AllenNLP: <https://allennlp.org/>
- 허깅페이스: <https://huggingface.co/>

the 1990s, the number of people in the world who are under 15 years of age is expected to increase from 1.1 billion to 1.5 billion.

As a result of the demographic changes, the number of people in the world who are aged 65 and over is expected to increase from 300 million in 1990 to 600 million in 2020.

The demographic changes are also expected to have a significant impact on the world's economy. The number of people in the world who are aged 15 and over is expected to increase from 3.5 billion in 1990 to 5.5 billion in 2020.

The demographic changes are also expected to have a significant impact on the world's environment. The number of people in the world who are aged 15 and over is expected to increase from 3.5 billion in 1990 to 5.5 billion in 2020.

The demographic changes are also expected to have a significant impact on the world's culture. The number of people in the world who are aged 15 and over is expected to increase from 3.5 billion in 1990 to 5.5 billion in 2020.

The demographic changes are also expected to have a significant impact on the world's politics. The number of people in the world who are aged 15 and over is expected to increase from 3.5 billion in 1990 to 5.5 billion in 2020.

The demographic changes are also expected to have a significant impact on the world's religion. The number of people in the world who are aged 15 and over is expected to increase from 3.5 billion in 1990 to 5.5 billion in 2020.

The demographic changes are also expected to have a significant impact on the world's science and technology. The number of people in the world who are aged 15 and over is expected to increase from 3.5 billion in 1990 to 5.5 billion in 2020.

The demographic changes are also expected to have a significant impact on the world's art and literature. The number of people in the world who are aged 15 and over is expected to increase from 3.5 billion in 1990 to 5.5 billion in 2020.

The demographic changes are also expected to have a significant impact on the world's sports and recreation. The number of people in the world who are aged 15 and over is expected to increase from 3.5 billion in 1990 to 5.5 billion in 2020.

The demographic changes are also expected to have a significant impact on the world's health and medicine. The number of people in the world who are aged 15 and over is expected to increase from 3.5 billion in 1990 to 5.5 billion in 2020.

The demographic changes are also expected to have a significant impact on the world's education and training. The number of people in the world who are aged 15 and over is expected to increase from 3.5 billion in 1990 to 5.5 billion in 2020.

The demographic changes are also expected to have a significant impact on the world's social and economic development. The number of people in the world who are aged 15 and over is expected to increase from 3.5 billion in 1990 to 5.5 billion in 2020.

The demographic changes are also expected to have a significant impact on the world's environment and natural resources. The number of people in the world who are aged 15 and over is expected to increase from 3.5 billion in 1990 to 5.5 billion in 2020.

The demographic changes are also expected to have a significant impact on the world's culture and heritage. The number of people in the world who are aged 15 and over is expected to increase from 3.5 billion in 1990 to 5.5 billion in 2020.

The demographic changes are also expected to have a significant impact on the world's politics and governance. The number of people in the world who are aged 15 and over is expected to increase from 3.5 billion in 1990 to 5.5 billion in 2020.

The demographic changes are also expected to have a significant impact on the world's religion and spirituality. The number of people in the world who are aged 15 and over is expected to increase from 3.5 billion in 1990 to 5.5 billion in 2020.

The demographic changes are also expected to have a significant impact on the world's science and technology. The number of people in the world who are aged 15 and over is expected to increase from 3.5 billion in 1990 to 5.5 billion in 2020.

# 02장

---

## 트랜스포머 모델 아키텍처 살펴보기

---

- 2.1 트랜스포머의 시작: Attention is All You Need
- 2.2 학습과 성능
- 2.3 허깅페이스의 트랜스포머 모델
- 2.4 정리하기
- 2.5 문제
- 2.6 참고 문헌

언어는 커뮤니케이션의 핵심이다. 언어를 구성하는 단어 시퀀스(sequence)가 없었다면 문명은 탄생하지 못했을 것이다. 이제 우리는 언어를 디지털로 표현하는 세상에 살고 있고 우리의 일상은 디지털화된 언어 기능인 NLP에 의존하고 있다. 검색엔진, 이메일, 소셜 네트워크, 게시물, 트윗, 스마트폰 메시지, 번역, 웹페이지, 스트리밍 사이트의 음성 대본 변환, 자동 응답기의 음성 합성 등 다양한 기능을 일상에서 사용 중이다.

**1장, 트랜스포머란 무엇인가?**에서는 RNN의 한계와 설계 및 개발의 상당 부분을 차지하는 클라우드 AI 트랜스포머에 대해 알아보았다. 4차 산업 시대 개발자의 역할은 트랜스포머의 아키텍처와 그 뒤를 잇는 다양한 트랜스포머 생태계를 이해하는 것이다.

2017년 12월, 바스와니(Vaswani)는 구글 브레인과 구글 리서치 동료와 함께 쓴 기념비적인 논문, 「Attention is All You Need」를 발표했다. 트랜스포머가 탄생한 것이다. 트랜스포머는 기존 최고의 성능(state-of-the-art)을 가진 NLP 모델을 능가했다. 트랜스포머를 사용하면 이전보다 더 빠르게 학습하며 더 좋은 성능을 얻을 수 있었다. 그 결과 트랜스포머는 NLP의 핵심 요소가 되었다.

트랜스포머의 어텐션 헤드(attention head) 아이디어는 순환 신경망을 대체해 나갔다. 이번 장에서는 바스와니가 설명하는 트랜스포머를 들여다보며 구성 요소를 살펴보겠다. 매력적인 어텐션의 세계를 탐험하며 핵심 요소를 설명하고자 한다.

이 장에서는 다음 주제들을 다룬다.

- 트랜스포머 아키텍처
- 트랜스포머의 셀프-어텐션(self-attention) 모델
- 인코딩(encoding) 및 디코딩(decoding) 스택(stack)
- 입력 및 출력 임베딩
- 위치 임베딩(positional embedding)
- 셀프-어텐션
- 멀티-헤드 어텐션(multi-head attention)
- 마스크드 멀티 어텐션(masked multi-attention)
- 잔차 연결(residual connections)
- 정규화
- 순방향 네트워크(feedforward network)
- 출력 확률

이제 트랜스포머의 아키텍처를 살펴보자.

## 2.1

# 트랜스포머의 시작: Attention is All You Need

2017년 12월, 바스와니 등의 기념비적인 논문, 「Attention is All You Need」를 발표한다. 이들은 구글 리서치와 구글 브레인에서 연구를 수행했다. 앞으로 이 책에서는, 「Attention is All You Need」에서 설명한 트랜스포머를 “오리지널 트랜스포머 모델”이라고 부르겠다.

### NOTE

부록 I, 트랜스포머 용어 설명을 참고하면 기존의 딥러닝 용어를 트랜스포머 어휘로 전환하는 데 도움이 될 것이다. 부록 I은 신경망 모델 관련 용어의 변화를 요약하여 보여준다.

이번 섹션에서는 논문에서 소개한 트랜스포머 모델의 구조를 확인해 보고, 이후 이어지는 섹션으로 각 구성 요소를 살펴보겠다.

오리지널 트랜스포머는 층 여섯 개를 쌓아 올린 스택 형태로 되어있다. 마지막 층을 제외하고, N번째 층의 출력은 N+1번째 층의 입력이 된다. 왼쪽에는 여섯 개의 층을 가진 인코더 스택(encoder stack)이 있고, 오른쪽에는 여섯 개의 층을 가진 디코더 스택(decoder stack)이 있다.

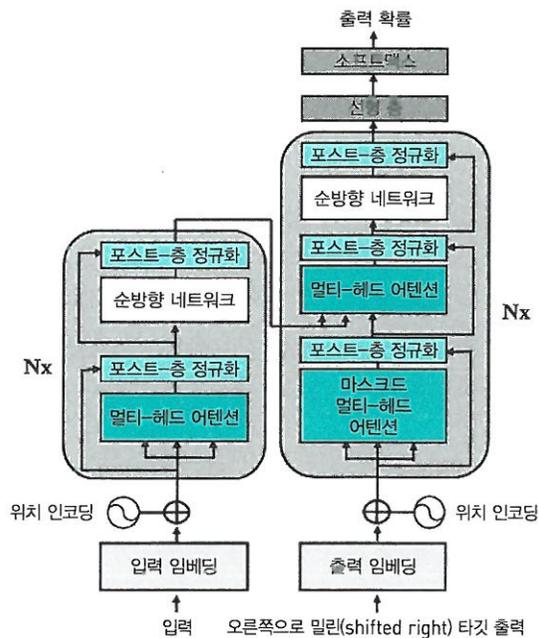


그림 2.1 트랜스포머 아키텍처

그림 2.1의 왼쪽은 트랜스포머의 인코더 부분으로, 입력값이 들어오는 부분이다. 어텐션 층과 순방향 (feedforward) 층으로 이루어져 있다. 오른쪽은 두 어텐션 층과 하나의 순방향 층으로 이루어진 트랜스포머의 디코더 부분으로, 타겟(target) 출력값을 입력받는다. RNN, LSTM, CNN 등은 전혀 사용하지 않았다. 트랜스포머에는 재귀적(recurrence) 구조가 없다.

단어 간 거리가 멀어질수록 더 많은 파라미터가 필요했던 재귀적 구조 대신 어텐션을 사용했다. 어텐션은 “단어-투-단어(word to word)” 연산이다. 실제로는 토큰-투-토큰(token to token) 연산이지만 이해를 돕기 위해 단어 수준으로 설명하겠다. 어텐션 메커니즘은 한 단어가 자신을 포함한 시퀀스 내 모든 단어들과 각각 어떻게 연관되어 있는지 계산한다.

다음 시퀀스(문장)를 살펴보자.

The cat sat on the mat.

어텐션은 단어 벡터 간의 내적(dot product)을 사용하여 한 단어와 가장 밀접한 관계를 가지는 단어를 찾는다. 이때 탐색 대상에는 자기 자신도 포함된다.(“cat”과 “cat” 사이의 관계)

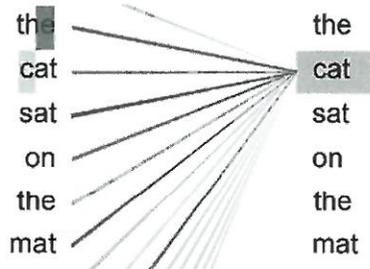


그림 2.2 모든 단어와의 관계 계산

어텐션 메커니즘을 사용하면 단어 간의 관계를 더 깊게 파악하고 더 좋은 성능을 얻을 수 있다.

오리지널 트랜스포머 모델은 계산 속도를 높이기 위해 각 어텐션 층마다 여덟 번의 어텐션 메커니즘을 병렬로 처리한다. 이어지는 “인코더 스택” 섹션에서 더 자세히 알아보겠다. 이 구조는 “멀티-헤드 어텐션”이라고 불리며 다음과 같은 기능을 제공한다.

- 시퀀스에 대한 더 심층적인 분석
- 재귀적 구조를 없애 계산량 감소
- 병렬화로 인한 학습 시간 단축
- 동일한 입력 시퀀스를 다른 관점으로 학습하는 각각의 어텐션 메커니즘

NOTE

어텐션 메커니즘은 재귀적 구조를 대체하였다. 하지만 트랜스포머에는 어텐션 메커니즘만큼 중요하고 창의적인 부분들이 더 있다. 내부 아키텍처를 살펴보면 점차 알 수 있을 것이다.

지금까지는 트랜스포머의 겉모습을 살펴보았다. 이제 트랜스포머 각 구성 요소의 내부를 살펴보자. 인코더부터 시작하겠다.

### 2.1.1 인코더 스택

오리지널 트랜스포머 모델의 인코더와 디코더는 층을 쌓아 올린 스택 형태(stack of layer)로 되어있다. 각 인코더 층의 구조는 다음과 같다.

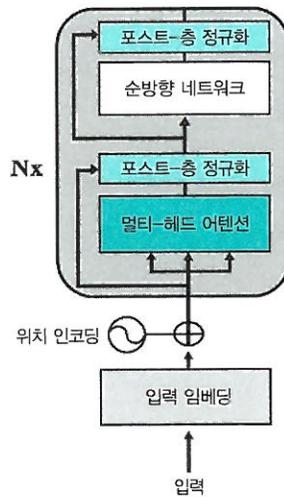


그림 2.3 트랜스포머 인코더 스택의 층

오리지널 트랜스포머 모델의 인코더 층은 총 N=6개이고 모두 동일한 구조이다. 각각의 층에 멀티-헤드 어텐션 메커니즘, 완전 연결 위치별 순방향 네트워크(fully connected position-wise feed-forward network)인 두 서브 층을 가지고 있다.

잔차 연결(residual connection)이 트랜스포머 모델의 각 서브 층을 둘러싸고 있다. 잔차 연결은 서브 층의 입력 x를 층 정규화(layer normalization) 함수에 전달하여, 위치 인코딩(positional encoding)과 같은 중요한 정보가 손실되지 않도록 보장한다. 각 층의 정규화된 출력은 다음과 같다.

$$\text{LayerNormalization}(x + \text{Sublayer}(x))$$

인코더의 N=6개 층이 모두 완전히 동일한 구조일지라도, 각 층은 서로 다른 내용을 담고 있다.

예를 들어, 임베딩 서브 층은 스택의 가장 아래에만 위치한다. 다른 다섯 층은 임베딩 층을 포함하고 있지 않고, 이 덕분에 인코딩된 입력이 모든 층에 걸쳐 안정적으로 유지된다.

멀티-헤드 어텐션 메커니즘 또한 여섯 개의 층에 동일하게 적용되지만 각자 다른 역할을 수행한다. 각 층은 이전 층의 출력을 토대로 시퀀스 내 토큰들의 관계를 파악할 다양한 방법들을 학습한다. 마치 우리가 글자와 단어 사이의 연관성을 찾으며 십자말풀이 퍼즐을 푸는 것처럼, 멀티-헤드 어텐션 메커니즘은 단어 사이의 다양한 연관성을 찾는다.

트랜스포머의 설계자는 매우 효율적인 제약 조건을 도입했다. 임베딩 층과 잔차 연결을 포함해서 모델을 구성하는 모든 서브 층의 출력 차원을 일정하게 했다. 이 차원( $d_{\text{model}}$ )은 목적에 따라 다른 값으로 설정할 수 있으며 오리지널 트랜스포머 모델에서는  $d_{\text{model}}=512$ 로 설정하고 있다.

출력 차원  $d_{\text{model}}$ 을 일정하게 유지하면 강력한 이점이 있다. 사실상 모든 주요 연산은 벡터의 내적이다. 결과적으로 차원을 항상 일정하게 유지할 수 있게 됨에 따라, 연산량과 리소스의 사용량을 줄이고 모델에 흐르는 정보를 쉽게 추적할 수 있다.

이처럼 인코더를 전체적인 관점에서 바라보면 트랜스포머의 아키텍처가 고도로 최적화되어 있음을 알 수 있다. 이어지는 섹션에서는 각 서브 층과 메커니즘을 자세히 들여다보겠다.

임베딩 서브 층부터 시작해 보자.

### 2.1.1.1 입력 임베딩

입력 임베딩 서브 층은 오리지널 트랜스포머 모델의 학습된 임베딩을 사용하여 입력 토큰을  $d_{\text{model}}=512$  차원의 벡터로 변환한다. 입력 임베딩은 고전적인 구조를 가지고 있다.

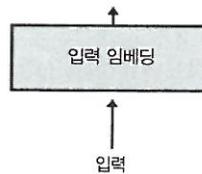


그림 2.4 트랜스포머의 입력 임베딩 서브 층

임베딩 서브 층은 일반적인 트랜스덕션(transduction) 모델과 동일하게 동작한다. 먼저 BPE(Byte-Pair Encoding) 워드 피스(word piece), 센텐스 피스(sentence piece)와 같은 토크나이저가 문장을 토큰으로 분리한다. 오리지널 트랜스포머는 BPE를 사용했지만, 모델마다 다른 토크나이저를 사용하기도 한다. 모델에 따른 선택일 뿐 목적은 동일하다. “the Transformer is an innovative NLP model!”이란 문장에 토크나이저를 적용하면 다음과 같이 토큰을 생성한다.

```
['the', 'transform', 'er', 'is', 'an', 'innovative', 'n', 'l', 'p', 'model', '!']
```

토크나이저가 대문자를 소문자로 변경하고 문장을 하위 부분들로 잘라낸 것을 볼 수 있다. 일반적으로 토크나이저는 다음과 같이 임베딩 과정에 사용될 정수 표현까지 제공한다.

```
text = "The cat slept on the couch. It was too tired to get up."
tokenized text = [1996, 4937, 7771, 2006, 1996, 6411, 1012, 2009, 2001, 2205, 5458, 2000,
2131, 2039, 1012]
```

현재 토큰화된 텍스트에는 활용할 정보가 충분하지 않다. 토큰화된 텍스트를 임베딩해야 한다.

트랜스포머는 학습된 임베딩 서브 층을 포함하는데, 토큰화된 입력에 다양한 임베딩 방법들을 적용할 수 있다.

트랜스포머의 임베딩 서브 층을 묘사하기 위해, 구글이 2013년에 발표한 스킵 그램(skip-gram) 아키텍처를 설명해 보겠다. 스킵 그램은 주어진 단어에 기초하여 주변(context) 단어를 예측하도록 학습하는 모델이다. 스텝 크기가 2인 윈도우(window)의 중심에 단어 word(i)가 있다면 word(i-2), word(i-1), word(i+1), word(i+2)를 학습하고, 윈도우를 한 칸씩 움직이며 과정을 반복한다. 스킵 그램 모델은 일반적으로 입력 층, 가중치, 은닉 층, 토큰화된 입력단어에 대한 임베딩 출력 층으로 구성된다.

다음 문장을 임베딩한다고 가정해 보자.

```
The black cat sat on the couch and the brown dog slept on the rug.
```

‘black’과 ‘brown’ 두 단어를 살펴보면, 두 단어의 임베딩 벡터는 비슷할 것이다.

우리는 각 단어에 대한  $d_{\text{model}}=512$ 차원의 벡터를 생성해야 하므로, 각 단어마다 512차원의 임베딩 벡터를 얻을 것이다.

```
black=[[-0.01206071 0.11632373 0.06206119 0.01403395 0.09541149
0.10695464 0.02560172 0.00185677 -0.04284821 0.06146432 0.09466285
0.04642421 0.08680347 0.05684567 -0.00717266 -0.03163519 0.03292002
-0.11397766 0.01304929 0.01964396 0.01902409 0.02831945 0.05870414
0.03390711 -0.06204525 0.06173197 -0.08613958 -0.04654748 0.02728105
-0.07830904
...
0.04340003 -0.13192849 -0.00945092 -0.00835463 -0.06487109 0.05862355
-0.03407936 -0.00059001 -0.01640179 0.04123065
-0.04756588 0.08812257 0.00200338 -0.0931043 -0.03507337 0.02153351
-0.02621627 -0.02492662 -0.05771535 -0.01164199
-0.03879078 -0.05506947 0.01693138 -0.04124579 -0.03779858
-0.01950983 -0.05398201 0.07582296 0.00038318 -0.04639162
-0.06819214 0.01366171 0.01411388 0.00853774 0.02183574
-0.03016279 -0.03184025 -0.04273562]]
```

단어 'black'을 512차원으로 표현했다. 다른 임베딩 방법을 사용할 수도 있고  $d_{\text{model}}$ 을 더 높은 값으로 설정할 수도 있다.

단어 'brown' 또한 512차원으로 표현할 수 있다.

```
brown=[[ 1.35794589e-02 -2.18823571e-02 1.34526128e-02 6.74355254e-02
1.04376070e-01 1.09921647e-02 -5.46298288e-02 -1.18385479e-02
4.41223830e-02 -1.84863899e-02 -6.84073642e-02 3.21860164e-02
4.09143828e-02 -2.74433400e-02 -2.47369967e-02 7.74542615e-02
9.80964210e-03 2.94299088e-02 2.93895267e-02 -3.29437815e-02
...
7.20389187e-02 1.57317147e-02 -3.10291946e-02 -5.51304631e-02
-7.03861639e-02 7.40829483e-02 1.04319192e-02 -2.01565702e-03
2.43322570e-02 1.92969330e-02 2.57341694e-02 -1.13280728e-01
8.45847875e-02 4.90090018e-03 5.33546880e-02 -2.31553353e-02
3.87288055e-05 3.31782512e-02 -4.00604047e-02 -1.02028981e-01
3.49597558e-02 -1.71501152e-02 3.55573371e-02 -1.77437533e-02
-5.94457164e-02 2.21221056e-02 9.73121971e-02 -4.90022525e-02]]
```

코사인 유사도(cosine similarity)로 'black'과 'brown'의 임베딩이 유사한지 확인하면, 두 단어의 임베딩 결과를 검증할 수 있다.

코사인 유사도는 단위구면(unit sphere)에 벡터를 표현하기 위해 유클리드 노름(Euclidean norm)을 사용한다. 비교하려는 두 벡터의 내적으로 두 벡터 사이의 코사인 값을 나타낼 수 있다. 코사인 유사도에 대한 더 자세한 내용은 사이킷런(scikit-learn)의 문서에서 확인할 수 있다. <https://scikit-learn.org/stable/modules/metrics.html#cosine-similarity>.

예시로 들었던, 'black'과 'brown'에 대한 두  $d_{\text{model}}=512$ 차원 임베딩 벡터의 코사인 유사도를 구해보면 다음과 같다.

```
cosine_similarity(black,brown)=[0.9998901]
```

스킵 그래프가 서로 유사한 두 벡터를 생성했음을 알 수 있다. 스킵 그래프가 'black'과 'brown'이 모두 색상의 하위 집합임을 감지해 낸 것이다.

이제 이어지는 트랜스포머의 층은 유용한 정보를 가지고 시작한다. 단어들이 어떻게 연관되는지 단어 임베딩으로 이미 학습했기 때문이다. 그런데, 아직 중요한 정보가 누락되었다. 단어가 시퀀스의 어디에 위치하는지 전달하지 않았다.

트랜스포머의 설계자는 위치 인코딩이라는 혁신적인 방법을 생각해 냈다.

위치 인코딩이 어떻게 동작하는지 알아보자.

## 위치 인코딩

단어가 어디에 위치하는지 고려하지 않고, 트랜스포머의 위치 인코딩 함수를 다음과 같이 그대로 적용한다.

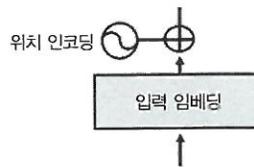


그림 2.5 위치 인코딩

위치 벡터를 별개로 학습하면 트랜스포머의 학습 속도가 매우 느려질 수 있고 어텐션의 서브 층이 너무 복잡해질 위험이 있다. 따라서 추가적인 벡터를 사용하는 대신, 입력 임베딩에 위치 인코딩 값을 더하여 시퀀스 내 토큰의 위치를 표현하였다.

NOTE

4차 산업혁명은 실용적이며 범용적이다. 오리지널 트랜스포머 모델은 하나의 벡터 안에 단어 임베딩과 위치 인코딩 정보를 담았다. 15장, NLP부터 범용 트랜스포머 모델까지에서 별도의 행렬로 위치 인코딩을 분리한 disentangled attention을 알아볼 것이다.

위치 임베딩 함수의 출력 벡터는  $d_{model}=512$ (또는 설정한 다른 상수)의 고정된 크기로 트랜스포머에 전달되어야 한다.

단어 임베딩 서브 층에 사용한 문장을 돌아보면, 'black'과 'brown'이 의미적으로 비슷하지만, 문장 내에서 멀리 떨어져 있음을 볼 수 있다.

The **black** cat sat on the couch and the **brown** dog slept on the rug.

'black'이라는 단어는 두 번째(pos=2) 위치에 있으며 'brown'은 열 번째(pos=10)에 위치한다.

우리는 각각의 단어 임베딩에 적절한 값을 더해 정보를 추가해야 한다. 정보가 더해져야 할 벡터의 크기는  $d_{model}=512$ 차원이므로, 512개의 숫자를 사용해 'black'과 'brown'의 단어 임베딩 벡터에 위치 정보를 주어야 한다.

위치 임베딩을 구현하는 방법에는 여러 가지가 있다. 이번 섹션에서는 단위구와 사인(sin) 및 코사인으로 위치 임베딩을 구현한 트랜스포머 설계자의 방식에 초점을 맞추려 한다.

바스와니(Vaswani et al.)는 사인과 코사인으로 각 위치와, 단어 임베딩의 각 차원  $d_{model}=512$ 개에 서로 다른 주기를 가지는 위치 인코딩(PE) 값을 생성했다.

$$PE_{(pos\ 2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

$$PE_{(pos\ 2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

단어 임베딩의 맨 앞 차원부터 시작해서,  $i=0$ 부터  $i=511$ 까지 순서대로 적용한다. 이때, 짝수 번째는 사인 함수를, 홀수 번째는 코사인 함수를 적용한다. 어떤 구현체에선 사인 함수의 적용 영역을  $i \in [0,255]$ 로 코사인 함수의 영역을  $i \in [256,512]$ 로 다르게 적용하기도 하지만 결과적으론 유사하다.

이번 섹션에서는, 바스와니(Vaswani et al.)가 서술한 방식을 사용한다. 위치 pos에 대한 위치 벡터  $pe[0][i]$ 를 파이썬 의사 코드(pseudo code)로 표현하면 다음과 같다.

```
def positional_encoding(pos, pe):
    for i in range(0, 512, 2):
        pe[0][i] = math.sin(pos / (10000 ** ((2 * i) / d_model)))
        pe[0][i+1] = math.cos(pos / (10000 ** ((2 * i) / d_model)))
    return pe
```

#### NOTE

구글 브레인 트렉스(Google Brain Trax)와 허깅페이스(Hugging Face)는 단어 임베딩과 위치 인코딩 라이브러리를 제공한다. 따라서, 이번 섹션에서 소개한 코드들을 직접 실행할 필요는 없다. 코드를 더 살펴보기 원한다면, 이번 챕터의 깃허브(GitHub) 저장소에 있는 구글 코랩(Google Colaboratory) positional\_encoding.ipynb 노트북과 text.txt에서 확인할 수 있다.

다음으로 넘어가기 전에,  $pos=2$ 에 대한 사인 그래프를 확인해 보자.

구글 검색으로 그래프를 그려볼 수 있다.

```
plot y=sin(2/10000^(2*x/512))
```

그래프 요청을 입력하기만 하면 된다.

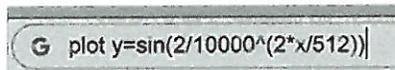


그림 2.6 구글에서 그래프 그리기

그래프 결과는 다음과 같다.

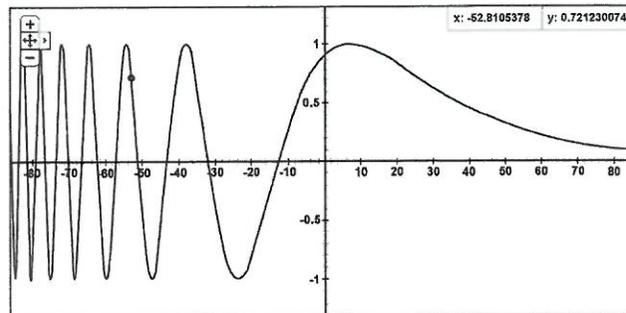


그림 2.7 그래프

이번 섹션에서 분석 중인 문장으로 돌아가면, 'black'은 pos=2에 위치하고 'brown'은 pos=10에 위치했다. 사인과 코사인에 대한 함수를 pos=2로 대입하면, 다음과 같은 512차원의 위치 벡터를 얻는다.

```
PE(2)=
[[ 9.09297407e-01 -4.16146845e-01  9.58144367e-01 -2.86285430e-01
   9.87046242e-01 -1.60435960e-01  9.99164224e-01 -4.08766568e-02
   9.97479975e-01  7.09482506e-02  9.84703004e-01  1.74241230e-01
   9.63226616e-01  2.68690288e-01  9.35118318e-01  3.54335666e-01
   9.02130723e-01  4.31462824e-01  8.65725577e-01  5.00518918e-01
   8.27103794e-01  5.62049210e-01  7.87237823e-01  6.16649508e-01
   7.46903539e-01  6.64932430e-01  7.06710517e-01  7.07502782e-01
   ...
   5.47683925e-08  1.00000000e+00  5.09659337e-08  1.00000000e+00
   4.74274735e-08  1.00000000e+00  4.41346799e-08  1.00000000e+00
   4.10704999e-08  1.00000000e+00  3.82190599e-08  1.00000000e+00
   3.55655878e-08  1.00000000e+00  3.30963417e-08  1.00000000e+00
   3.07985317e-08  1.00000000e+00  2.86602511e-08  1.00000000e+00
   2.66704294e-08  1.00000000e+00  2.48187551e-08  1.00000000e+00
   2.30956392e-08  1.00000000e+00  2.14921574e-08  1.00000000e+00]]
```

pos=10에 대한 512차원의 위치 인코딩 벡터는 다음과 같다.

```
PE(10)=
[[-5.44021130e-01 -8.39071512e-01  1.18776485e-01 -9.92920995e-01
   6.92634165e-01 -7.21289039e-01  9.79174793e-01 -2.03019097e-01
   9.37632740e-01  3.47627431e-01  6.40478015e-01  7.67976522e-01
   2.09077001e-01  9.77899194e-01 -2.37917677e-01  9.71285343e-01
  -6.12936735e-01  7.90131986e-01 -8.67519796e-01  4.97402608e-01
  -9.87655997e-01  1.56638563e-01 -9.83699203e-01 -1.79821849e-01
   ...
   2.73841977e-07  1.00000000e+00  2.54829672e-07  1.00000000e+00
   2.37137371e-07  1.00000000e+00  2.20673414e-07  1.00000000e+00
   2.05352507e-07  1.00000000e+00  1.91095296e-07  1.00000000e+00
   1.77827943e-07  1.00000000e+00  1.65481708e-07  1.00000000e+00
   1.53992659e-07  1.00000000e+00  1.43301250e-07  1.00000000e+00
   1.33352145e-07  1.00000000e+00  1.24093773e-07  1.00000000e+00
   1.15478201e-07  1.00000000e+00  1.07460785e-07  1.00000000e+00]]
```

파이썬으로 바스와니(Vaswani et al.)의 설명을 그대로 적용한 결과가 유의미한지 확인해 보자.

단어 임베딩에 사용했던 코사인 유사도 함수는 위치의 근접성을 시각화할 때도 유용하다.

```
cosine_similarity(PE(2), PE(10))= [[0.8600013]]
```

'black'과 'brown'의 위치적 유사도는 어휘적 유사도(단어가 같은 그룹에 속하는지)와는 차이가 있다.

```
cosine_similarity(black, brown)= [[0.9998901]]
```

두 위치 벡터의 유사도는 단어 임베딩의 유사도보다 낮은 값을 보여준다.

위치 벡터로 두 단어를 서로 떨어뜨릴 수 있는 것이다. 단어 임베딩은 학습에 사용되었던 말뭉치에 따라 달라짐에 유의하자. 이제 위치 인코딩을 단어 임베딩 벡터와 어떻게 합칠지 알아보겠다.

### 임베딩 벡터에 위치 인코딩 더하기

트랜스포머 설계자는 단순히 위치 정보를 단어 임베딩 벡터와 더했다.

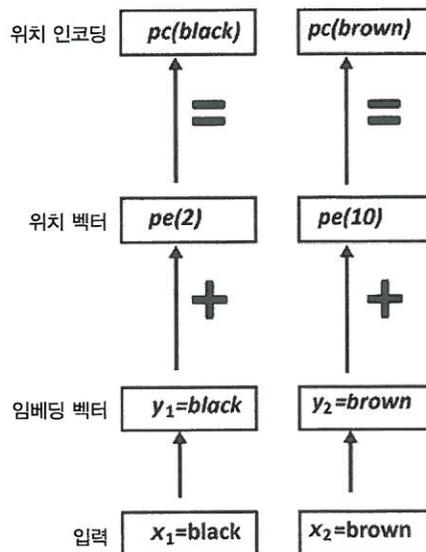


그림 2.8 위치 인코딩

예를 들어 'black'의 단어 임베딩  $y_1 = \text{black}$ 을, 인코딩 함수로 얻은 위치 벡터  $pe(2)$ 와 더한다고 하자. 입력 단어 'black'에 대한 위치 인코딩  $pc(\text{black})$ 은 다음과 같이 얻는다.

$$pc(\text{black}) = y1 + pe(2)$$

매우 간단한 방법이다. 그런데, 이렇게 더하기만 한다면 위치 정보로 인해 단어 임베딩의 정보가 훼손될 위험이 있다.

단어 임베딩 층의 정보를 이어지는 층에 더 확실하게 전달하기 위해,  $y1$ 의 값을 키우는 다양한 방법이 있다.

그중 한 가지는, 'black'의 임베딩  $y1$ 에 임의의 값을 곱하는 것이다.

$$y1 * \text{math.sqrt}(d\_model)$$

이제 동일한 512 크기의 두 벡터, 'black'의 단어 임베딩과 위치 벡터를 더해보자.

```
for i in range(0, 512, 2):
    pe[0][i] = math.sin(pos / (10000 ** ((2 * i)/d_model)))
    pc[0][i] = (y[0][i]*math.sqrt(d_model))+ pe[0][i]

    pe[0][i+1] = math.cos(pos / (10000 ** ((2 * i)/d_model)))
    pc[0][i+1] = (y[0][i+1]*math.sqrt(d_model))+ pe[0][i+1]
```

다음과 같이  $d_{model}=512$ 차원의 최종 위치 인코딩 벡터를 얻을 수 있다.

```
pc(black)=
[[ 9.09297407e-01 -4.16146845e-01 9.58144367e-01 -2.86285430e-01
 9.87046242e-01 -1.60435960e-01 9.99164224e-01 -4.08766568e-02
...
4.74274735e-08 1.00000000e+00 4.41346799e-08 1.00000000e+00
4.10704999e-08 1.00000000e+00 3.82190599e-08 1.00000000e+00
2.66704294e-08 1.00000000e+00 2.48187551e-08 1.00000000e+00
2.30956392e-08 1.00000000e+00 2.14921574e-08 1.00000000e+00]]
```

'brown'을 비롯한 시퀀스 내 모든 단어에 대해서도 동일한 연산이 적용된다.

'black'과 'brown'의 최종 위치 인코딩 벡터를 코사인 유사도 함수로 비교하면 다음과 같다.

$$\text{cosine\_similarity}(pc(\text{black}), pc(\text{brown})) = [[0.9627094]]$$

'black'과 'brown'으로 비교했던 각 단계의 코사인 유사도 결과로, 위치 인코딩 프로세스를 명확하게 볼 수 있다.

```
[[0.99987495]] 단어 유사도
[[0.8600013]] 위치 벡터 유사도
[[0.9627094]] 최종적인 위치 인코딩의 유사도
```

처음에 0.99에 달하는 코사인 유사도로, 두 단어의 임베딩이 매우 유사함을 확인했었다. 그리고 나서 위치 벡터가 두 번째와 열 번째의 두 단어를 0.86의 더 낮은 유사도로 구분하는 것을 확인했다.

마지막으로, 각각의 단어 임베딩 벡터를 해당 위치의 위치 인코딩 벡터와 더한 후 코사인 유사도가 0.96이 되었음을 확인했다.

이제 각 단어의 위치 인코딩은 처음에 보았던 단어 임베딩 정보와 위치 정보를 모두 담고 있다.

위치 인코딩 결과는 멀티-헤드 어텐션으로 전달된다.

### 2.1.1.2 서브 층 1: 멀티-헤드 어텐션

멀티-헤드 어텐션 서브 층에는 여덟 개의 헤드(heads)가 있으며, 포스트-층 정규화(post-layer normalization)와 이어져있다. 포스트-층 정규화는 서브 층의 출력을 잔차 연결과 더한 후 정규화한다.

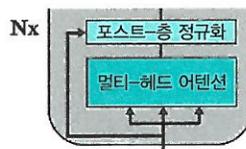


그림 2.9 멀티-헤드 어텐션 서브 층

이번 섹션에서는 어텐션의 아키텍처를 먼저 확인하고 간단한 멀티-헤드 어텐션 파이썬 예제를 살펴본다. 마지막에는 포스트-층 정규화를 설명할 것이다.

이제 멀티-헤드 어텐션의 아키텍처를 확인해 보자.

### 2.1.1.3 멀티-헤드 어텐션 아키텍처

인코더 스택 첫 번째 층의 멀티 어텐션 서브 층으로 각 단어의 임베딩과 위치가 담겨있는 벡터가 입력 된다. 인코더 스택의 이어지는 층에는 이 정보들이 다시 입력되지 않는다.

입력 시퀀스의 각 단어  $x_n$ 을 표현하는 벡터의 크기는  $d_{model}=512$ 이다.

$$pe(x_n) = [d_1 = 9.09297407e01, d_2 = -4.16146845e01, \dots, d_{512} = 1.00000000e + 00]$$

이제 각 단어  $x_n$ 을  $d_{model}=512$ 차원의 벡터로 표현할 수 있다. 각각의 단어를 다른 모든 단어와 비교 하며 시퀀스에 얼마나 적합한지 결정할 것이다.

다음 문장에서, 'it'은 'cat'일 수도 또는 'rug'일 수도 있다.

Sequence =The cat sat on the rug and it was dry-cleaned.

모델은 학습과정에서 'it'이 'cat'과 연관되어 있는지 아니면 'rug'와 연관되어 있는지 찾으려 할 것이다. 설정해 둔  $d_{model}=512$ 차원의 모델로 대규모 학습을 수행하면 된다.

그런데 하나의  $d_{model}$  차원 블록만 사용하면 한 번에 하나의 관점으로만 시퀀스를 분석하게 된다. 만약 다양한 관점으로 분석하려면 상당한 계산 시간이 필요하다.

대신, 시퀀스 내 단어 집합  $x$ 의 각 단어  $x_n$ 을 표현하는  $d_{model}=512$ 차원을 여덟 개로 나누어  $d_k=64$  차원으로 만들면 더 효과적이다.

“헤드” 여덟 개를 병렬로 연산하면 학습 속도를 높이면서 단어 간의 관계를 표현하는 서로 다른 여덟 개의 표현 공간(representation subspace)을 얻게 된다.

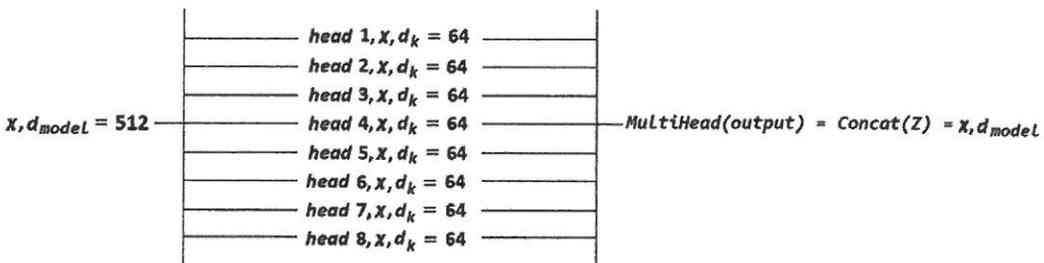


그림 2.10 멀티-헤드에 대한 표현

이제 헤드 여덟 개를 병렬로 실행할 수 있다. 하나의 헤드는 'it'이 'cat'과, 어떤 헤드는 'it'이 'rug'와, 또 다른 헤드는 'rug'가 'dry-cleaned'와 연관되어 있음을 알아낼 것이다.

각 헤드의 출력을  $x * d_k$  모양의 행렬  $Z_i$ 라고 한다면, 멀티-헤드 어텐션의 출력  $Z$ 는 다음과 같다.

$$Z = (Z_0, Z_1, Z_2, Z_3, Z_4, Z_5, Z_6, Z_7)$$

멀티-헤드 서브 층의 출력이 벡터 시퀀스가 아니라  $x_m * d_{model}$  형태의 행렬이 될 수 있도록 연결(concatenate)해야 한다.<sup>8</sup>

멀티-헤드 어텐션 서브 층을 벗어나기 전에,  $Z$ 의 요소들을 연결하면 다음과 같다.

$$\text{MultiHead}(\text{output}) = \text{Concat}(Z_0, Z_1, Z_2, Z_3, Z_4, Z_5, Z_6, Z_7) = x, d_{model}$$

각각의 헤드를 서로 연결하여  $d_{model}=512$ 차원이 되었다. 이제 멀티-헤드 층의 출력은 오리지널 트랜스포머의 제약 조건을 지키고 있다.

어텐션 메커니즘 헤드  $h_n$ 의 내부에서는 “단어”를 세 가지 행렬로 표현한다.

- 다른 “단어” 행렬의 모든 키-밸류 쌍(key-value pair)을 탐색하는,  $d_q=64$ 차원의 쿼리(query) 행렬 ( $Q$ )
- 어텐션 점수를 구하기 위해 학습한,  $d_k=64$ 차원의 키(key) 행렬 ( $K$ )
- 또 다른 어텐션 점수를 구하기 위해 학습한,  $d_v=64$ 차원의 밸류(value) 행렬 ( $V$ )

“스케일드 내적 어텐션(Scaled Dot-Product Attention)”이라고도 불리는 이 어텐션 메커니즘은 다음과 같이  $Q, K, V$ 로 표현할 수 있다.

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

$Q, K, V$ 는 동일한 차원을 가지므로, 내적 연산으로 여덟 개의 헤드에서 어텐션 값을 계산하고 서로 연결하여 출력  $Z$ 를 얻는 과정이 비교적 간단하다.

$Q, K, V$ 를 얻기 위해서는,  $d_{model}=512$ 개의 행과  $d_k=64$ 개의 열을 가진 가중치 행렬  $Q_w, K_w, V_w$ 를 학습해야 한다.  $Q$ 는  $x$ 와  $Q_w$ 의 내적으로 얻을 수 있고, 결과적으로  $Q$ 의 차원은  $d_k=64$ 가 된다.

8 역주. 여기서  $x_m$ 의  $m$ 은 멀티-헤드 층의 출력임을 표시한 것으로,  $x_m$ 을 그냥  $x$ 로 이해해도 무방하다.

#### NOTE

층과 헤드의 개수,  $d_{model}$ ,  $d_k$  등 모든 트랜스포머의 파라미터는 원하는 대로 변경할 수 있다. 이번 장은 바스와니(Vaswani et al.)가 소개한 오리지널 트랜스포머의 파라미터를 그대로 사용했다. 기존 모델을 수정하거나 다른 변형 모델들을 살펴보기 전에 원본 모델을 아는 것이 중요하다.

구글 브레인 트랙스와 OpenAI, 허깅페이스 등에는 이 책이 다루는 내용과 관련한 라이브러리가 준비되어 있다.

하지만, 방금 살펴본 아키텍처를 좀 더 설명하기 위해 파이썬 코드로 시각화하고 중간 과정을 이미지로 보이게 하자.

어텐션 메커니즘의 주요 부분을 실행하기 위해, 기본 파이썬 코드에 넘파이와 소프트맥스(softmax) 함수만 사용하여 10단계로 나타낼 것이다.

#### NOTE

4차 산업혁명 개발자는 동일 알고리즘에 대한 다양한 아키텍처를 사용할 수 있어야 함을 명심하자.

모델의 입력을 표현하는 1단계 과정부터 시작하자.

#### 2.1.1.4 1단계: 입력 표현

Multi\_Head\_Attention\_Sub\_Layer.ipynb 파일을 구글 드라이브에 저장하고(지메일 계정 필요) 구글 코랩에서 열어보자. 해당 노트북 파일은 이번 챕터의 깃허브 저장소에 있다.

최소한의 파이썬 함수를 사용해서 트랜스포머 어텐션 헤드의 내부 동작을 간단하게 알아보자. 기본 코드를 사용해서 멀티-헤드 어텐션 서브 층의 내부를 살펴보겠다.

```
import numpy as np
from scipy.special import softmax
```

어텐션 메커니즘의 입력을  $d_{model}=512$ 에서  $d_{model}=4$ 로 축소하였다. 입력 벡터의 차원을  $d_{model}=4$ 로 만들어 쉽게 시각화하고자 한다.

$x$ 는 512차원 대신 4차원을 가진 입력 세 개로 구성했다.

```

print("Step 1: Input : 3 inputs, d_model=4")
x = nparray([[1.0, 0.0, 1.0, 0.0], # 입력1
             [0.0, 2.0, 0.0, 2.0], # 입력2
             [1.0, 1.0, 1.0, 1.0]]) # 입력3
print(x)

```

출력 결과로  $d_{\text{model}}=4$ 인 벡터 세 개를 확인할 수 있다.

```

Step 1: Input : 3 inputs, d_model=4
[[1. 0. 1. 0.]
 [0. 2. 0. 2.]
 [1. 1. 1. 1.]]

```

모델의 첫 단계를 완성했다.

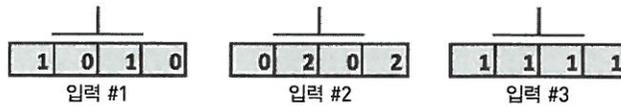


그림 2.11 멀티-헤드 어텐션 서브 층의 입력

이제 모델에 가중치 행렬을 추가하자.

### 2.1.1.5 2단계: 가중치 행렬 초기화

각 입력에는 세 개의 가중치 행렬이 관여한다.

- 쿼리를 얻기 위한  $Q_w$
- 키를 얻기 위한  $K_w$
- 밸류를 얻기 위한  $V_w$

세 가지 가중치 행렬은 모델의 모든 입력에 적용된다.

바스와니(Vaswani et al.)가 설명한 가중치 행렬은  $d_k=64$ 차원이다. 하지만  $d_k=3$ 차원으로 축소해서 표현할 것이다. 가중치 행렬을  $3 \times 4$ 로 축소하여 중간 결과를 더 쉽게 표현하면서 입력  $x$ 와 내적을 수행할 수 있도록 하였다.

## NOTE

사용 중인 교육용 노트북에서 설정한 행렬 크기와 모양은 임의의 값이다. 목표는 어텐션 메커니즘의 전반적인 과정을 수행해 보는 것이다.

세 가지 가중치 행렬 중 쿼리 가중치 행렬부터 초기화한다.

```
print("Step 2: weights 3 dimensions x d_model=4")
print("w_query")
w_query = np.array([[1, 0, 1],
                   [1, 0, 0],
                   [0, 0, 1],
                   [0, 1, 1]])
print(w_query)
```

다음은 가중치 행렬 `w_query`의 출력 결과이다.

```
w_query
[[1 0 1]
 [1 0 0]
 [0 0 1]
 [0 1 1]]
```

이어서 키 가중치 행렬을 초기화한다.

```
print("w_key")
w_key = np.array([[0, 0, 1],
                  [1, 1, 0],
                  [0, 1, 0],
                  [1, 1, 0]])
print(w_key)
```

키 가중치 행렬의 출력 결과는 다음과 같다.

```
w_key
[[0 0 1]
 [1 1 0]
 [0 1 0]
 [1 1 0]]
```

마지막으로 벨류 가중치 행렬을 초기화한다.

```
print("w_value")
w_value = nparray([[0, 2, 0],
 [0, 3, 0],
 [1, 0, 3],
 [1, 1, 0]])
print(w_value)
```

벨류 가중치 행렬은 다음과 같다.

```
w_value
[[0 2 0]
 [0 3 0]
 [1 0 3]
 [1 1 0]]
```

모델의 두 번째 단계를 완성했다.

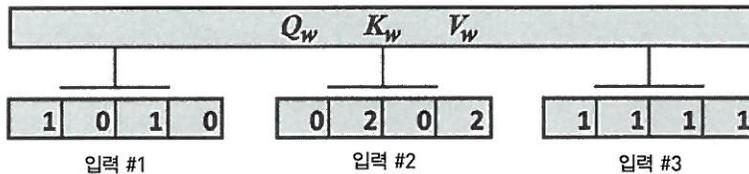


그림 2.12 모델에 세 가지 가중치 행렬을 추가했다

이제 가중치에 입력 벡터를 곱해 Q, K, V를 구해보자.

### 2.1.1.6 3단계: Q, K, V를 얻기 위한 행렬 곱

입력 벡터에 가중치 행렬을 곱해 각 입력에 대한 쿼리, 키, 밸류 벡터를 얻는다.

이 모델에서는 모든 입력에 대해  $w_{\text{query}}$ ,  $w_{\text{key}}$ ,  $w_{\text{value}}$  가중치 행렬을 하나씩만 사용한다고 가정한다. 다른 방식도 있을 수 있다.

먼저 입력 벡터를  $w_{\text{query}}$  가중치 행렬과 곱한다.

```
print("Step 3: Matrix multiplication to obtain Q, K, V")
print("Query: x * w_query")
Q=np.matmul(x, w_query)
print(Q)
```

결과 벡터는  $Q_1=[1, 0, 2]$ ,  $Q_2=[2, 2, 2]$ ,  $Q_3=[2, 1, 3]$ 으로 다음과 같다.

```
Step 3: Matrix multiplication to obtain Q, K, V
Query: x * w_query
[[1. 0. 2.]
 [2. 2. 2.]
 [2. 1. 3.]]
```

다음으로 입력 벡터를  $w_{\text{key}}$  가중치 행렬과 곱한다.

```
print("Key: x * w_key")
K=np.matmul(x, w_key)
print(K)
```

벡터  $K_1=[0, 1, 1]$ ,  $K_2=[4, 4, 0]$ ,  $K_3=[2, 3, 1]$ 을 얻을 수 있다.

```
Key: x * w_key
[[0. 1. 1.]
 [4. 4. 0.]
 [2. 3. 1.]]
```

마지막으로 입력 벡터를 `w_value` 가중치 행렬과 곱한다.

```
print("Value: x * w_value")
V=np.matmul(x, w_value)
print(V)
```

벡터  $V_1=[1, 2, 3]$ ,  $V_2=[2, 8, 0]$ ,  $V_3=[2, 6, 3]$ 을 얻을 수 있다.

```
Value: x * w_value
[[1. 2. 3.]
 [2. 8. 0.]
 [2. 6. 3.]]
```

세 번째 단계를 완료했다.

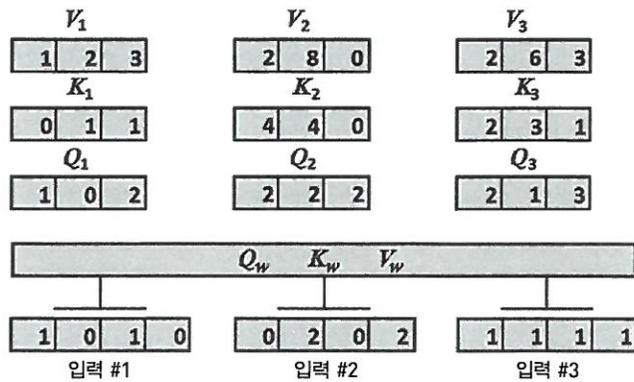


그림 2.13 Q, K, V를 생성했다

어텐션 점수를 계산하기 위한 Q, K, V를 확보했다.

### 2.1.1.7 4단계: 스케일드 어텐션 점수

오리지널 트랜스포머의 어텐션 헤드에 구현된 식은 다음과 같다.

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

4단계는 Q와 K에 집중하겠다.

$$\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

이 모델에선  $\sqrt{d_k} = \sqrt{3} = 1.75$  대신 1을 사용하고 이전에 구한 Q와 K를 대입한다.

```
print("Step 4: Scaled Attention Scores")
k_d=1 # 이번 예제에서는 k_d=3의 제곱근을 내림하여 1을 사용한다
attention_scores = (Q @ K.transpose())/k_d
print(attention_scores)
```

어텐션 점수의 중간 결과는 다음과 같다.

```
Step 4: Scaled Attention Scores
[[ 2.  4.  4.]
 [ 4. 16. 12.]
 [ 4. 12. 10.]]
```

4단계를 완료했다.  $x_1$ 에 대해서 K를 따라 계산한 중간 결과는 [2, 4, 4]이다.

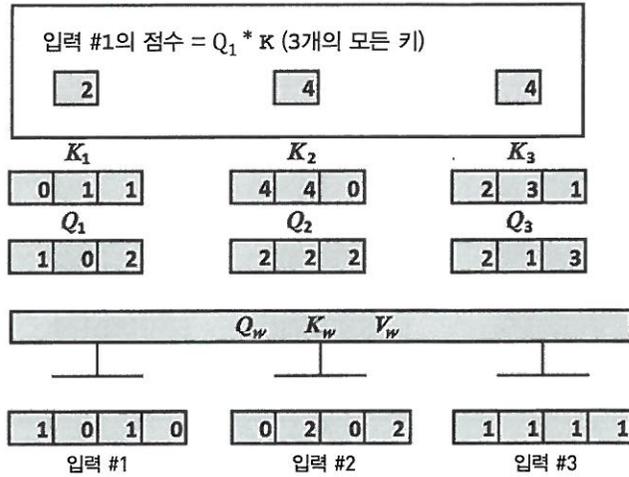


그림 2.14 입력 #1에 대한 스케일드 어텐션 점수

이제, 어텐션 계산식에 따라, 어텐션 점수의 중간 결과에 소프트맥스를 적용하자.

### 2.1.1.8 5단계: 각 벡터의 스케일드 소프트맥스 어텐션 점수

각 어텐션 중간 결과에 소프트맥스 함수를 적용한다. 행렬 곱셈을 수행하는 대신, 개별 벡터를 직접 계산하겠다.

```
print("Step 5: Scaled softmax attention_scores for each vector")
attention_scores[0]=softmax(attention_scores[0])
attention_scores[1]=softmax(attention_scores[1])
attention_scores[2]=softmax(attention_scores[2])
print(attention_scores[0])
print(attention_scores[1])
print(attention_scores[2])
```

다음과 같이 각 벡터에 대한 스케일드 소프트맥스 어텐션 점수를 얻었다.

```
Step 5: Scaled softmax attention_scores for each vector
[0.06337894 0.46831053 0.46831053]
[6.03366485e-06 9.82007865e-01 1.79861014e-02]
[2.95387223e-04 8.80536902e-01 1.19167711e-01]
```

5단계를 완료했다.  $x_1$ 과 모든 키 행렬에 대해 계산한 소프트맥스 점수는 다음과 같다.

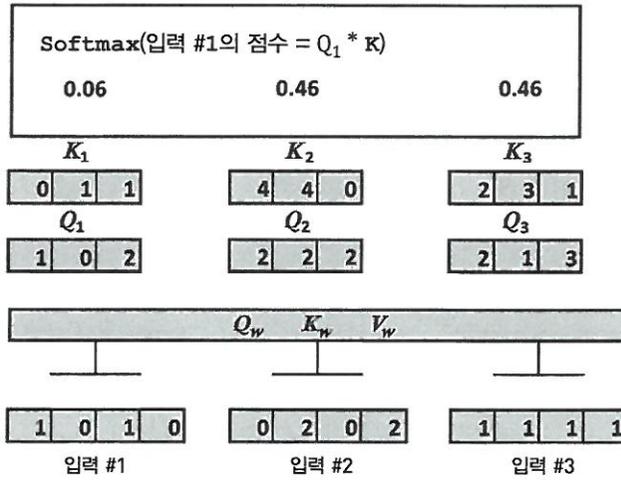


그림 2.15 입력 #1과 모든 키에 대한 소프트맥스 점수

이제, 방정식으로 표현했던, 최종적인 어텐션 점수를 구할 수 있다.

### 2.1.1.9 6단계: 어텐션을 표현하기 위한 마지막 과정

이제 어텐션 방정식에  $V$ 를 대입하기만 하면 된다.

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

6단계와 7단계에선  $x_1$ 에 대한 어텐션 점수를 먼저 계산한다. 하나의 단어 벡터에 대해 하나의 어텐션 값을 구하는 것이다. 8단계에서는 다른 두 입력 벡터에 대해 어텐션 계산 과정을 일반화할 것이다.

$x_1$ 에 대한  $\text{Attention}(Q, K, V)$ 를 구하기 위해, 중간 결과에 밸류 벡터 세 개를 하나씩 곱한다. 내부 과정을 자세히 나타내면 다음과 같다.

```

print("Step 6: attention value obtained by score1/k_d * v")
print(v[0])
print(v[1])
print(v[2])
print("Attention 1")
attention1=attention_scores[0].reshape(-1,1)
attention1=attention_scores[0][0]*v[0]
print(attention1)

print("Attention 2")
attention2=attention_scores[0][1]*v[1]
print(attention2)

print("Attention 3")
attention3=attention_scores[0][2]*v[2]
print(attention3)

```

```

Step 6: attention value obtained by score1/k_d * v
[1. 2. 3.]
[2. 8. 0.]
[2. 6. 3.]

Attention 1
[0.06337894 0.12675788 0.19013681]
Attention 2
[0.93662106 3.74648425 0. ]
Attention 3
[0.93662106 2.80986319 1.40493159]

```

6단계를 완료했다.  $x_1$ 의 세 입력으로부터, 어텐션 값 세 개를 구했다.

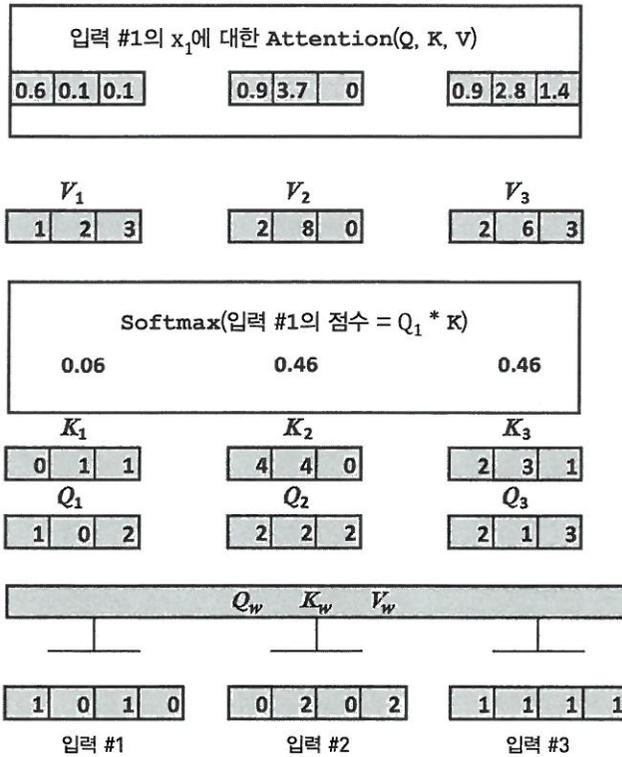


그림 2.16 최종적인 어텐션의 표현

이제 어텐션 결과를 하나로 합칠 것이다.

### 2.1.1.10 7단계: 결과 합산하기

입력 #1로부터 계산한 세 어텐션 값을 합산하면 출력 행렬의 첫 줄을 얻을 수 있다.

```
print("Step 7: summed the results to create the first line of the output matrix")
attention_input1=attention1+attention2+attention3
print(attention_input1)
```

다음은 출력 행렬의 첫 줄이자 입력 #1에 대응하는 결과이다.

```
Step 7: summed the results to create the first line of the output matrix
[1.93662106 6.68310531 1.59506841]]
```

두 번째 줄은, 그다음 입력인, 입력 #2에 대한 결과가 된다.

그림 2.17에서  $x_1$ 의 어텐션을 모두 합산한 것을 볼 수 있다.

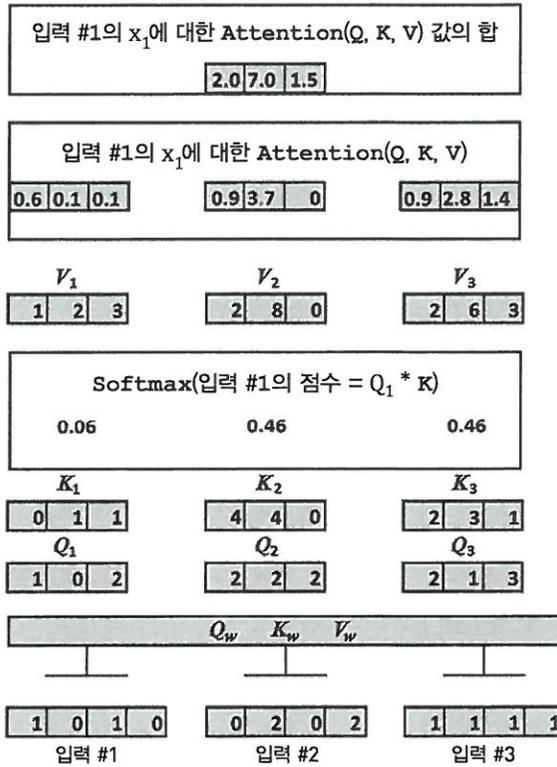


그림 2.17 하나의 입력에 대해 결과를 합산했다

입력 #1에 대한 모든 단계를 완료했다. 이제 모든 입력에 대해서도 결과를 합산해야 한다.

### 2.1.1.11 8단계: 1단계부터 7단계까지의 과정을 모든 입력에 적용

입력 #2와 입력 #3에 대해 “1단계~7단계”의 과정을 적용해서, 하나의 어텐션 헤드에 대해, 트랜스포머 어텐션 값을 구할 수 있다.

이번 단계부터는,  $d_{model}=64$ 로 학습한 가중치로 어텐션 값 세 개를 구했다고 가정하겠다. 이제 오리지널 모델에서 서브 층의 출력이 어떻게 보이는지 확인해 보자.

지금까지 어텐션을 표현하는 과정을 작은 모델로 자세히 확인했다.  $d_{model}=64$ 차원에서 어텐션 표현 세 개를 얻었다고 가정하고 결과를 만들어 보자.

```

print("Step 8: Step 1 to 7 for inputs 1 to 3")
#학습된 가중치로 결과 세 개를 얻었다고 가정한다.(이 예제에서 학습하진 않았다)
#오리지널 트랜스포머 논문대로 구현했다고 가정한다. 64 차원의 결과 세 개를 만든다
attention_head1=np.random.random((3, 64))
print(attention_head1)

```

다음은 헤드 하나로부터  $d_{\text{model}}=64$ 차원의 출력 세 개를 얻는, 시뮬레이션  $Z_0$ 의 결과이다.

```

Step 8: Step 1 to 7 for inputs 1 to 3
[[0.31982626 0.99175996...(61 squeezed values)...0.16233212]
 [0.99584327 0.55528662...(61 squeezed values)...0.70160307]
 [0.14811583 0.50875291...(61 squeezed values)...0.83141355]]

```

랜덤함수로 벡터를 생성했기 때문에 노트북의 실행 결과는 다를 수 있다.

이제 각 입력으로부터 트랜스포머 헤드 한 개의 결과를 얻었다. 다음 단계는 헤드 여덟 개로 결과를 얻고 어텐션 서브 층의 최종 결과를 만드는 것이다.

#### 2.1.1.12 9단계: 어텐션 서브 층 헤드의 출력

어텐션 서브 층의 여덟 헤드를 이미 학습했다고 가정한다. 각 트랜스포머 헤드는(세 개의 단어 또는 워드피스에 대해)  $d_{\text{model}}=64$ 차원의 벡터 세 개를 출력한다.

```

print("Step 9: We assume we have trained the 8 heads of the attention
sublayer")
z0h1=np.random.random((3, 64))
z1h2=np.random.random((3, 64))
z2h3=np.random.random((3, 64))
z3h4=np.random.random((3, 64))
z4h5=np.random.random((3, 64))
z5h6=np.random.random((3, 64))
z6h7=np.random.random((3, 64))
z7h8=np.random.random((3, 64))
print("shape of one head",z0h1.shape,"dimension of 8 heads",64*8)

```

다음 결과는 하나의 헤드에 대한 출력의 모양을 보여준다.

```
Step 9: We assume we have trained the 8 heads of the attention sublayer
shape of one head (3, 64) dimension of 8 heads 512
```

여덟 개의 헤드가 이제  $Z$ 를 형성한다.

$$Z = (Z_0, Z_1, Z_2, Z_3, Z_4, Z_5, Z_6, Z_7)$$

이제 트랜스포머는  $Z$ 의 여덟 개의 요소를 연결해서 멀티-헤드 어텐션 서브 층의 최종 결과를 만들 것이다.

### 2.1.1.13 10단계: 각 헤드의 출력 연결하기

트랜스포머는 다음과 같이  $Z$ 의 여덟 개의 요소를 연결한다.

$$\text{MultiHead}(\text{Output}) = \text{Concat}(Z_0, Z_1, Z_2, Z_3, Z_4, Z_5, Z_6, Z_7)W^0 = x, d_{\text{model}}$$

$Z$ 를 또 다른 학습된 가중치 행렬  $W^0$ 와 곱하고 있음에 유의하자. 이 모델에선  $W^0$ 를 곱하는 과정이 연결 함수에 안에 통합되어 있다고 가정하겠다.

다음으로  $Z_0$ 부터  $Z_7$ 까지 연결한다.

```
print("Step 10: Concatenation of heads 1 to 8 to obtain the original
8x64=512 output dimension of the model")
output_attention=np.hstack((z0h1, z1h2, z2h3, z3h4, z4h5, z5h6, z6h7, z7h8))
print(output_attention)
```

출력값은  $Z$ 를 모두 연결한 형태이다.

```
Step 10: Concatenation of heads 1 to 8 to obtain the original 8x64=512
output dimension of the model
[[0.65218495 0.11961095 0.9555153 ... 0.48399266 0.80186221 0.16486792]
 [0.95510952 0.29918492 0.7010377 ... 0.20682832 0.4123836 0.90879359]
 [0.20211378 0.86541746 0.01557758 ... 0.69449636 0.02458972 0.889699 ]]
```

시각화하면  $Z$ 의 요소들을 나란히 쌓아놓은 모습이 된다.

멀티-헤드 어텐션 출력

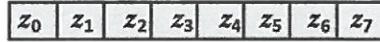


그림 2.19 어텐션 서브 층의 출력

연결 작업으로  $d_{model}=512$ 차원의 표준 출력을 만들었다.

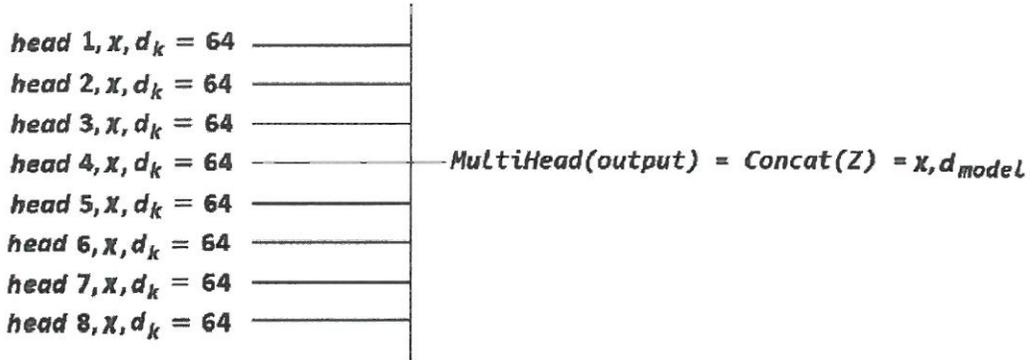


그림 2.19 헤드 출력 여덟 개의 연결

이제 어텐션 서브 층의 결과에 층 정규화를 적용할 것이다.

### 2.1.1.14 포스트-층 정규화

트랜스포머의 각 어텐션 서브 층과, 순방향 서브 층 뒤에는 포스트-층 정규화(Post-LN)가 이어 진다.

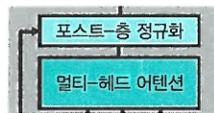


그림 2.20 포스트-층 정규화

포스트-층 정규화는 덧셈 함수와 층 정규화 작업으로 구성된다. 덧셈 함수는 서브 층의 입력과 이어진 잔차 연결을 처리한다. 잔차 연결은 중요한 정보들을 잃어버리지 않도록 방지한다. 포스트-층 정규화를 식으로 표현하면 다음과 같다.

$$\text{LayerNormalization}(x + \text{Sublayer}(x))$$

Sublayer(x)는 서브 층을 의미한다. x는 서브 층에 입력된 정보이다.

$x + \text{Sublayer}(x)$ 의 결과인 벡터  $v$ 가 LayerNormalization의 입력이 된다. 트랜스포머의 모든 입력과 출력은 모든 과정에서  $d_{\text{model}}=512$ 차원으로 표준화(standardize)된다.

층 정규화에는 많은 방법이 있으며, 모델마다 다양하게 변형하여 사용한다. 벡터  $v$ 에 대한 층 정규화 LayerNormalization( $v$ )의 기본 개념은 다음과 같다.

$$\text{LayerNormalization}(v) = \gamma \frac{v - \mu}{\sigma} + \beta$$

변수들을 설명하자면,  $\mu$ 는  $d$ 차원 벡터  $v$ 에 대한 평균이다.

$$\mu = \frac{1}{d} \sum_{k=1}^d v_k$$

$\sigma$ 는  $d$ 차원 벡터  $v$ 에 대한 분산이다.  $\sigma^2 = \frac{1}{d} \sum_{k=1}^d (v_k - \mu)^2$

$\gamma$ 는 스케일링을 위한 파라미터이다.

$\beta$ 는 편향(bias) 벡터이다.

앞서 확인한 LayerNormalization( $v$ )의 내용은 다양한 포스트-층 정규화 방법 중 가장 일반적인 방식이다. 다음 서브 층은 순방향 네트워크로, LayerNormalization( $v$ )의 출력과 이어진다.

### 2.1.1.15 서브 층 2: 순방향 네트워크

순방향 네트워크 (FFN)의 입력은  $d_{\text{model}}=512$ 차원으로, 앞선 서브 층에 대한 포스트-층 정규화의 출력이다.



그림 2.21 순방향 서브 층

다음은 순방향 네트워크 서브 층에 대한 설명이다.

- 인코더와 디코더의 순방향 네트워크는 완전 연결(fully connected) 되어있다.
- 순방향 네트워크는 위치별(position-wise) 네트워크이다. 동일한 연산을 위치별로 각각 수행한다.
- 순방향 네트워크는 층 두 개와 ReLU 활성화 함수로 이루어져 있다.
- 순방향 네트워크 층의 입력과 출력은  $d_{\text{model}}=512$ 차원이지만, 내부 층은  $d_{\text{ff}}=2048$ 로 더 크다
- 순방향 네트워크는 크기가 1인 커널로 두 번의 컨볼루션 연산을 수행하는 것으로 볼 수 있다.

설명들을 바탕으로, 최적화되고 표준화된 순방향 네트워크를 나타내면 다음과 같다.

$$\text{FFN}(\mathbf{x}) = \max(0, \mathbf{x}W_1 + \mathbf{b}_1)W_2 + \mathbf{b}_2$$

이전 섹션에서 설명한 대로, 순방향 네트워크의 출력은 포스트-층 정규화로 이어진다. 그리고 나서, 결과 값은 인코더 스택의 다음 층과 디코더 스택의 멀티-헤드 어텐션 층으로 전달된다.

이제 디코더 스택을 살펴보자.

## 2.1.2 디코더 스택

트랜스포머의 디코더 역시 인코더처럼 층을 쌓아 올린 스택 형태이다. 디코더 스택의 각 층의 구조는 다음과 같다.

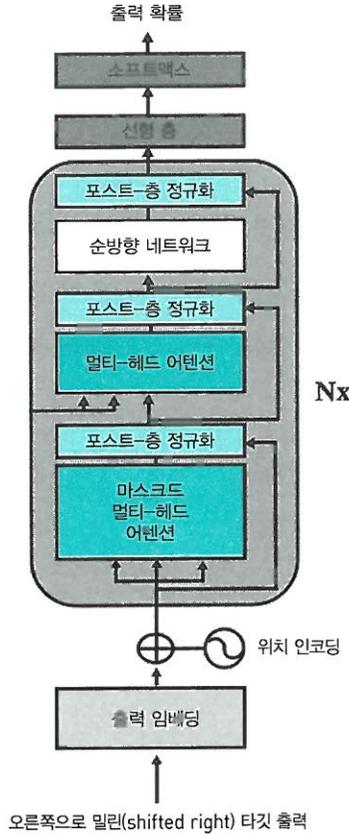


그림 2.22 트랜스포머 디코더 스택의 층

트랜스포머의 인코더처럼,  $N=6$ 개 디코더 층의 구조는 모두 동일하다. 각 층은 3개의 서브 층으로 이루어져 있는데, 멀티-헤드 마스크드 어텐션(multi-head masked attention) 메커니즘, 멀티-헤드 어텐션 메커니즘, 완전 연결 위치별 순방향 네트워크이다.

디코더에는 세 번째 주요 서브 층인 마스크드 멀티-헤드 어텐션이 있다. 마스크드 멀티-헤드 어텐션 서브 층에서는, 주어진 위치 이후의 모든 단어를 마스킹 함으로써, 트랜스포머가 나머지 시퀀스를 보지 않고 스스로의 추론에 근거하여 연산하도록 한다.

인코더 스택처럼, 세 개의 주요 서브 층 각각을 잔차 연결과 Sublayer(x)가 감싸고 있다.

$$\text{LayerNormalization}(x + \text{Sublayer}(x))$$

또한 인코더 스택에서처럼, 임베딩 서브 층은 디코더 스택의 가장 아래층과 연결되어 있으며 임베딩 층과 잔차 연결을 포함한 모든 서브 층의 출력은  $d_{\text{model}}$  차원으로 일정하다.

설계자들이 인코더와 디코더를 대칭적으로 만들기 위해 열심히 노력했음을 알 수 있다.

디코더의 각 서브 층과 함수의 구조는 인코더와 유사하다. 이번 챕터에서는, 디코더와 인코더의 차이점을 위주로 설명하겠다. 동일 기능에 대한 설명이 필요하면 인코더의 내용을 참조하자.

### 2.1.2.1 출력 임베딩과 위치 인코딩

디코더 서브 층의 구조는 인코더 서브 층의 구조와 대부분 동일하다. 출력 임베딩 층과 위치 인코딩 함수는 인코더 스택에서와 같다.

바스와니(Vaswani et al.)의 트랜스포머 활용 사례를 살펴보면, 번역 결과를 출력하도록 학습하고 있다. 다음 프랑스어가 출력이라면,

```
Output=Le chat noir était assis sur le canapé et le chien marron dormait sur le tapis
```

이에 대한 영어 입력은 다음과 같다.

```
Input=The black cat sat on the couch and the brown dog slept on the rug.
```

인코더 스택의 첫 부분처럼, 출력 단어는 단어 임베딩 층과 위치 인코딩 함수를 통과한다. 다음으로, 디코더 스택에 있는 멀티-헤드 어텐션 층의 특징을 확인해 보자.

### 2.1.2.2 어텐션 층

트랜스포머는 자기 회귀(auto-regressive) 모델로, 이전 출력 시퀀스를 추가 입력으로 사용한다. 디코더에 있는 멀티-헤드 어텐션 층의 동작은 인코더와 동일하지만, 마스크드 멀티-헤드 어텐션은 현재 위치를 포함한 모든 앞쪽 단어에만 어텐션을 적용한다. 이렇게 하면 뒤쪽 단어들은 트랜스포머에서 가려지고, 이들을 예측하는 방법을 학습할 수 있다.

인코더에서처럼, 마스크드 멀티-헤드 어텐션 서브 층 이후에는 포스트-층 정규화 과정이 이어진다.

멀티-헤드 어텐션 서브 층 또한, 트랜스포머가 예측 중인 위치까지만 어텐션을 적용해서 예측해야 할 시퀀스를 미리 볼 수 없도록 한다.

멀티-헤드 어텐션 서브 층은 어텐션 내적 연산에 인코더의 K와 V를 활용함으로써 인코더로부터 정보를 얻고, 마스크드 멀티-헤드 어텐션 서브 층의 정보를 가져오기 위해 해당 층(sublayer 1)의 Q를 어텐션 내적에 사용한다. 결과적으로 디코더는 인코더에 학습된 정보를 사용한다. 디코더의 멀티-헤드 어텐션 서브 층은 다음과 같이 정의할 수 있다.

$\text{Input\_Attention} = (\text{Output\_decoder\_sub\_layer} - 1(Q), \text{Output\_encoder\_layer}(K, V))$

마스크드 멀티-헤드 어텐션 서브 층 이후에는, 인코더에서처럼, 포스트-층 정규화 과정이 이어진다.

다음으로 트랜스포머는 순방향 네트워크 서브 층으로 이동하고, 포스트-층 정규화와 선형 층(linear layer)으로 이어진다.

### 2.1.2.3 순방향 서브 층, 포스트-층 정규화 그리고 선형 층

순방향 네트워크 서브 층의 구조는 인코더 스택의 순방향 네트워크와 동일하며, 포스트-층 정규화도 인코더 스택의 층 정규화와 동일하다.

트랜스포머는 한 번에 하나의 출력 시퀀스만 생성한다.

$$\text{Output sequence} = (y_1, y_2, \dots, y_n)$$

선형 층의 출력을 생성하는 선형 함수(linear function)는 모델마다 다르지만, 표준 식은 다음과 같다.

$$y = w * x + b$$

w와 b는 학습된 파라미터이다.

선형 층은 시퀀스의 다음으로 등장할 법한 요소들을 예측하고, 소프트맥스 함수로 가장 가능성 있는 요소가 정해진다. 인코더 층과 마찬가지로 디코더 층 또한 1번째 층에서 1+1번째 층으로 N=6개 중 최상위 층까지 이어진다.

이제 트랜스포머가 어떻게 학습되었고 어떤 성능을 얻었는지 살펴보겠다.

## 2.2

## 학습과 성능

오리지널 트랜스포머는 450만 개의 영어-독일어 문장 쌍과 3,600만 개의 영어-프랑스어 문장 쌍으로 학습되었다.

데이터셋은 **WMT(Workshops on Machine Translation)**에서 가져온 것으로, 자세히 보기 원한다면 다음 링크를 확인하자. (<http://www.statmt.org/wmt14/>)

오리지널 트랜스포머 기본 모델의 학습은 8개의 NVIDIA P100 GPU가 장착된 머신에서 100,000 스텝 동안 12시간이 걸렸다. 빅 모델의 경우에는 300,000 스텝 동안 3.5일이 소요되었다.

오리지널 트랜스포머는 WMT 영어-프랑스어 데이터셋에서 BLEU 점수 41.8점을 달성하며 이전의 모든 기계 번역 모델을 능가했다. BLEU는 Bilingual Evaluation Understudy의 약자로, 기계 번역 결과의 품질을 평가하는 알고리즘이다.

구글 리서치와 구글 브레인 팀은 트랜스포머의 성능을 개선하기 위해 최적화 전략을 사용했다. 예를 들어, 아담 옵티마이저(Adam optimizer)를 사용했지만, 워밍업 단계에서 학습률을 일정하게 증가시키고 이후 조금씩 감소시키는 방식으로 차이를 두었다.

임베딩을 합치는 과정에는 드롭아웃(dropout) 및 잔차 드롭아웃(residual dropout) 같은 다양한 정규화 기법이 적용되었다. 또한, 원-핫(one-hot)에 대한 과신(overconfident)과 오버피팅(overfitting)을 방지하기 위해 라벨 스무딩(label smoothing)이 적용되었다. 라벨 스무딩을 사용하면 평가 과정의 정확도가 낮아지지만, 모델이 더 많이 더 잘 학습하게 된다.

이렇게 트랜스포머와 관련된 여러 시도로 인해 다양한 용도의 다른 모델이 생겨났다. 이어지는 장에서 더 자세히 살펴볼 것이다.

이번 장을 마무리하기 전에, 허깅페이스를 예시로, 바로 사용할 수 있는(ready to use) 트랜스포머의 단순함을 느껴보자.

## 2.3

# 허깅페이스의 트랜스포머 모델

이번 챗터에서 본 모든 내용은 바로 사용할 수 있는 허깅페이스 트랜스포머에 압축되어있다. 허깅페이스를 사용하면, 기계 번역을 단 세 줄로 구현할 수 있다! 구글 코랩에서 `Multi_Head_Attention_Sub_Layer.ipynb` 파일을 열어보자. 노트북을 구글 드라이브에 저장하고(지메일 계정 필요) 마지막 두 셀로 이동하자.

먼저 허깅페이스 트랜스포머를 설치한다.

```
!pip -q install transformers
```

그리고 다양한 트랜스포머 사용법이 포함된 허깅페이스 파이프라인을 임포트한다.

```
#@title pipeline 모듈을 불러오고 영어 프랑스어 번역을 선택한다
from transformers import pipeline
```

다음으로, 바로 사용할 수 있는 다양한 기능들이 있는, 허깅페이스 파이프라인을 생성한다. 이번에는 번역 모델을 활성화하고 영어에서 프랑스어로 번역하기 위한 문장을 입력한다.

```
translator = pipeline("translation_en_to_fr") #한 줄의 코드면 충분하다!
print(translator("It is easy to translate languages with transformers", max_
length=40))
```

그리고 놀랍게도 다음과 같이 번역 내용이 표시된다.

```
{['translation_text': 'Il est facile de traduire des langues à l'aide de
transformateurs.']} #트랜스포머로 언어를 쉽게 번역할 수 있습니다.
```

이번 장에서는 먼저 트랜스포머 아키텍처가 먼 거리의 단어들을 연관짓는 놀라운 방법을 살펴보았다. 트랜스포머는 **자연어 이해(Natural Language Understanding, NLU)** 역사상 전례 없는 방식으로 문자 시퀀스를 의미있는 표현으로 트랜스덕션한다.

트랜스덕션의 확장과 구현의 단순화는 인공지능을 이전에 볼 수 없던 수준으로 끌어올렸다.

RNN, LSTM, CNN 등이 없이도 트랜스덕션과 시퀀스 모델링이 가능하게 한 트랜스포머 아키텍처의 대담한 접근 방식도 살펴보았다. 인코더와 디코더의 크기를 표준화하고 대칭적인 설계를 한 덕분에 여러 서브 층을 매끄럽게 연결할 수 있었다.

트랜스포머가 순환 신경망을 제거하는 것 외에도 병렬화를 적극 도입하여 학습 시간을 단축했음을 확인했고, 위치 인코딩이나 마스크드 멀티-헤드 어텐션 같은 혁신적인 내용도 살펴보았다. 유연하고 독창적인 트랜스포머 아키텍처는 다른 많은 혁신적인 변형 모델들의 기반이 되었고, 이 변형 모델들은 더욱 강력한 트랜스덕션과 언어 모델링을 위한 길을 열었다.

다른 장에서 다양한 변형 모델을 설명할 때 트랜스포머 아키텍처의 여러 측면을 더 자세히 살펴보겠다.

트랜스포머의 출시는 바로 사용할 수 있는 인공지능 모델 시대의 시작을 의미한다. 허깅페이스와 구글 브레인은 몇 줄의 코드만으로 인공지능을 구현할 수 있게 했다.

이어지는 **3장, BERT 모델 미세 조정하기**에서 오리지널 트랜스포머 모델의 강력한 진화를 확인하자.

## 2.5 문제

01. NLP 트랜스덕션은 텍스트 표현을 인코딩 또는 디코딩할 수 있다. 참 / 거짓
02. 자연어 이해는 자연어 처리의 하위 분야이다. 참 / 거짓
03. 언어 모델링 알고리즘은 입력 시퀀스에 기반하여 단어를 생성한다. 참 / 거짓
04. 트랜스포머는 LSTM에 CNN을 추가한 모델이다. 참 / 거짓
05. 트랜스포머에는 LSTM층 또는 CNN 층이 없다. 참 / 거짓
06. 어텐션은 시퀀스의 마지막 토큰뿐만 아니라 모든 토큰을 검사한다. 참 / 거짓
07. 트랜스포머는 위치 인코딩 대신 고정된 위치 벡터를 사용한다. 참 / 거짓
08. 트랜스포머에는 순방향 네트워크를 가지고있다. 참 / 거짓
09. 트랜스포머 디코더의 마스크드 멀티-헤드 어텐션은 처리중인 시퀀스의 나머지 뒷 부분을 알고리즘이 볼 수 없도록 한다. 참 / 거짓
10. 트랜스포머는 멀리 떨어진 단어간의 관계를 LSTM보다 잘 파악한다. 참 / 거짓

## 2.6 참고 문헌

- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin, 2017, Attention Is All YouNeed, <https://arxiv.org/abs/1706.03762>
- 허깅페이스 트랜스포머 사용법: <https://huggingface.co/transformers/usage.html>
- Tensor2Tensor (T2T) 소개: [https://colabresearch.google.com/github/tensorflow/tensor2tensor/blob/master/tensor2tensor/notebooks/hello\\_t2t.ipynb?hl=en](https://colabresearch.google.com/github/tensorflow/tensor2tensor/blob/master/tensor2tensor/notebooks/hello_t2t.ipynb?hl=en)
- 라이미 카림(Raimi Karim)의 설명을 기반으로 마누엘 로메로(Manuel Romero)가 구성한 코랩 노트북: <https://colab.research.google.com/drive/1rPk3ohrmVclqhH7uQ7qys4oznDdAhpzF>
- 구글 언어 리서치: <https://research.google/teams/language/>
- 허깅페이스 트랜스포머: <https://huggingface.co/transformers/index.html>
- 주석이 포함된 트랜스포머 논문: <http://nlp.seas.harvard.edu/2018/04/03/attention.html>
- 제이 알라마르(Jay Alammar)의 그림으로 설명한 트랜스포머(The Illustrated Transformer): <http://jalammargithub.io/illustratedtransformer/>



# 03장

---

## BERT 모델 미세 조정하기

---

- 3.1 BERT 아키텍처
- 3.2 BERT 미세 조정하기
- 3.3 정리하기
- 3.4 문제
- 3.5 참고 문헌

**2장, 트랜스포머 모델 아키텍처 살펴보기**에서 오리지널 트랜스포머 아키텍처의 구성 요소에 대해 알아보았다. 오리지널 트랜스포머를 레고 조각으로 조립된 모델이라고 생각해 보자. 구성품으로는 인코더(encoders), 디코더(decoders), 임베딩 층(embedding layers), 위치 인코딩 메서드(positional encoding methods), 멀티 헤드 어텐션 층(multi-head attention layers), 마스크드 멀티 헤드 어텐션 층(masked multi-head attention layers), 포스트-층 정규화(post-layer normalization), 순방향 서브 층(feed-forward sub-layers), 선형 출력 층(linear output layers) 등이 있다.

조각은 다양한 크기와 형태로 출시되므로 같은 조립 키트를 이용해 몇 시간이 고 다양한 모델을 만들어 볼 수 있다! 그중 어떤 모델은 전체 부품의 일부만 필요할 수도 있고 반대로 추가 부품이 필요한 경우 새로운 레고 조각을 구매할 수도 있다.

BERT는 트랜스포머 조립 키트에 양방향 멀티 헤드 어텐션 서브 층(bidirectional multi-head attention sub-layer)이라는 새로운 부품을 추가했다. 인간은 문장을 이해하는 데 어려움을 겪을 때 단순히 앞의 단어만 보지 않는다. BERT는 우리와 마찬가지로 같은 문장의 모든 단어를 동시에 살펴본다.

이 장에서는 먼저 **Bidirectional Encoder Representations from Transformers(BERT)**의 아키텍처를 살펴본다. BERT는 트랜스포머의 인코더 블록만 새로운 방식으로 사용하고 디코더 스택은 사용하지 않는다.

서드 파티(third party)에 의해 사전 학습되고 허깅페이스(Hugging Face)에 업로드된 BERT 모델을 미세 조정(fine-tune)할 것이다. 사전 학습된 BERT는 미세 조정하여 여러 NLP 작업에 사용하게 된다. 마지막으로는 허깅페이스 모델을 사용한 매력적인 다운스트림(downstream) 트랜스포머의 예시들을 살펴볼 것이다.

이 장에서는 다음 주제들을 다룬다.

- Bidirectional Encoder Representations from Transformers (BERT)
- BERT 아키텍처
- 두 단계 BERT 프레임워크
- 사전 학습 환경 준비하기
- 사전 학습 인코더 층 정의하기
- 미세 조정정의하기
- 다운스트림 멀티태스킹
- 미세 조정된 BERT 모델 구축하기
- 적합성 판단(acceptability judgment) 데이터셋(dataset) 불러오기
- 어텐션 마스크(attention masks) 생성하기
- BERT 모델 설정하기
- 미세 조정된 모델의 성능 측정하기

첫 번째 단계로 BERT 모델의 배경을 살펴보자.

## 3.1

# BERT 아키텍처

BERT는 트랜스포머 모델에 양방향 어텐션을 적용했다. 기존 트랜스포머 모델에 양방향 어텐션을 적용하려면 많은 변경이 필요하다.

여기서는 2장, **트랜스포머 모델 아키텍처 살펴보기**에서 설명한 트랜스포머의 구성 요소는 다루지 않는다. 트랜스포머의 구성 요소에 대해서는 2장을 복습하기 바란다. 이 섹션에서는 BERT 모델의 구체적인 측면에 초점을 맞춘다.

먼저 인코더 스택을 설명하는 2018년 데블린(Devlin et al.)의 설계가 진화하는 과정을 살펴본 다음 사전 학습 입력 환경을 준비해 본다. 그런 다음 BERT의 두 단계 프레임워크인 사전 학습과 미세 조정 에 대해 알아보도록 하자.

첫 번째로 인코더 스택을 살펴보자.

### 3.1.1 인코더 스택

오리지널 트랜스포머 모델에서 가져올 첫 번째 구성 요소는 인코더 층이다. 2장, **트랜스포머 모델 아키텍처 살펴보기**에서 설명한 대로 인코더 층은 그림 3.1과 같다.

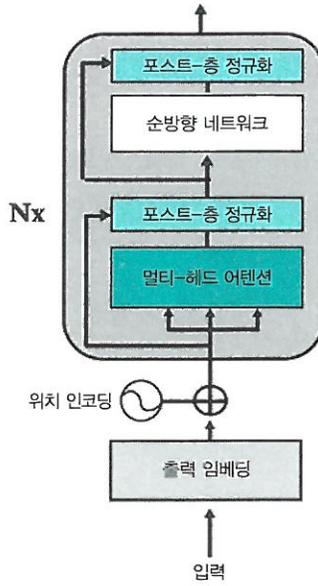


그림 3.1 인코더 층

BERT 모델은 디코더 층을 사용하지 않는다. 마스크드 토큰(예측할 토큰을 숨김)은 인코더의 어텐션 층에 있고 이는 다음 섹션에서 BERT 인코더 층을 보며 자세히 살펴보도록 한다.

오리지널 트랜스포머의 스택은  $N=6$ 층이다. 오리지널 트랜스포머의 차원(dimensions) 수는  $d_{\text{model}}=512$ 이다. 오리지널 트랜스포머의 어텐션 헤드(attention heads) 수는  $A=8$ 이다. 오리지널 트랜스포머 헤드의 차원은 다음과 같다.

$$d_k = \frac{d_{\text{model}}}{A} = \frac{512}{8} = 64$$

BERT의 인코더 층은 오리지널 트랜스포머 모델보다 더 크다.

인코더 층으로 두 가지 BERT 모델을 구축할 수 있다.

- BERT<sub>BASE</sub>의 인코더 스택은  $N=12$ 개 층으로 이루어져 있다.  $d_{\text{model}}=768$ 이며 BERT 논문에서와 같이  $H=768$ 로 표현할 수 있다. 멀티 헤드 어텐션 서브 층의 헤드는  $A=12$ 개다. 각 헤드  $z_A$ 의 차원은 오리지널 트랜스포머 모델과 같이 64로 유지된다.

$$d_k = \frac{d_{\text{model}}}{A} = \frac{768}{12} = 64$$

- 연결(concatenate)하기 이전 각 멀티 헤드 어텐션 서브 층의 출력은 헤드 12개의 출력과 같다.

$$\text{output\_multi-head\_attention} = \{z_0, z_1, z_2, \dots, z_{11}\}$$

- BERT<sub>LARGE</sub>의 인코더 스택은 N=24개 층으로 이루어져 있다.  $d_{\text{model}}=1024$ 이며 멀티 헤드 어텐션 서브 층에는 헤드 A=16개가 포함된다. 각 헤드  $z_A$ 의 차원도 오리지널 트랜스포머 모델과 같이 64로 유지된다.

$$d_k = \frac{d_{\text{model}}}{A} = \frac{768}{12} = 64$$

- 연결하기 전의 각 멀티 헤드 어텐션 서브 층의 출력은 헤드 16개의 출력과 같다.

$$\text{output\_multi-head\_attention} = \{z_0, z_1, z_2, \dots, z_{15}\}$$

모델 크기는 다음과 같이 요약할 수 있다.

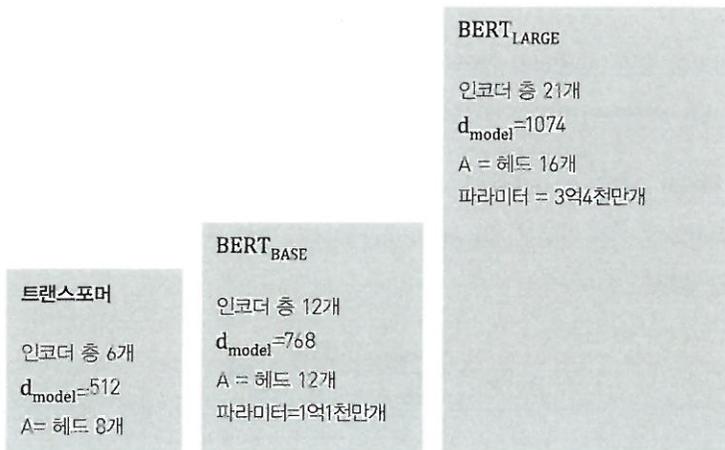


그림 3.2 트랜스포머 모델

#### NOTE

BERT 모델은 앞의 세 가지 설정에 국한되지 않는다. 앞의 세 가지 설정은 BERT 모델의 주요 측면을 나타낸 것이고 많은 변형이 가능하다.

크기와 차원은 BERT 스타일의 사전 학습의 본질이다. BERT 모델은 인간과 같다. BERT 모델은 더 많은 작업 메모리(차원)와 더 많은 지식(데이터)으로 더 나은 결과를 생성한다. 즉, 대량의 데이터를 사전 학습한 대형 트랜스포머 모델이 다운스트림 NLP 작업을 더 잘한다.

이제 첫 번째 서브 층을 보며 BERT 모델의 입력 임베딩과 위치 인코딩의 기본에 대해 살펴보자.

### 3.1.2 사전 학습 입력 환경 준비하기

BERT 모델에는 디코더 스택 층이 없다. 따라서 마스크드 멀티 헤드 어텐션 서브 층이 없다. BERT 설계자들은 나머지 시퀀스(sequence)를 마스크하는 마스크드 멀티 헤드 어텐션 층이 어텐션 프로세스를 방해한다고 말한다.

마스크드 멀티 헤드 어텐션 층은 현재 위치 이후의 모든 토큰을 마스크한다. 다음 예시 문장을 살펴보자.

```
The cat sat on it because it was a nice rug.
```

'it'라는 단어에 도달했을 때 인코더의 입력은 다음처럼 될 수 있다.

```
The cat sat on it<masked sequence>
```

이 접근 방식의 동기(motivation)는 모델이 예측해야 하는 출력을 보지 못하도록 하는 것이다. 이렇게 왼쪽에서 오른쪽으로 접근하는 방식(left-to-right approach)은 비교적 좋은 결과를 만들 수 있다.

하지만 이런 방식으로는 모델이 많은 것을 학습할 수 없다. 'it'이 무엇을 가리키는지 알기 위해서는 전체 문장을 보고 'rug'에 도달하여 'it'이 'rug'라는 것을 알아내야 한다.

이를 위해 BERT 엔지니어(authors)는 모델을 다른 방식으로 사전 학습시켜 보면 어떨까? 라는 아이디어를 떠올렸다.

BERT 엔지니어는 어텐션 헤드가 왼쪽에서 오른쪽으로, 오른쪽에서 왼쪽으로 모든 단어에 어텐션하는 양방향 어텐션 방식을 생각해냈다. 이로써 인코더의 셀프 어텐션 마스크는 디코더의 마스크드 멀티 헤드 어텐션 서브 층의 방해받지 않고도 예측할 수 있게 되었다.

이 모델은 두 가지 방법으로 학습된다. 첫 번째 방법은 **Masked Language Modeling(MLM, 마스크드 언어 모델링)**이고, 두 번째 방법은 **Next Sentence Prediction(NSP, 다음 문장 예측하기)**이다.

마스크드 언어 모델링부터 살펴보자

### 3.1.2.1 마스크드 언어 모델링

마스크드 언어 모델링은 예측하기 위해 가시(visible) 단어 시퀀스와 마스크된 시퀀스로 이어진 시퀀스로 모델을 학습할 필요가 없다.

BERT는 무작위로 문장의 한 단어에 마스크를 씌운 후 양방향으로 분석하는 방식을 사용했다.

BERT는 입력에 단어 분할(subword segmentation) 토큰화 방법인 워드피스(WordPiece)를 적용하고, 사인-코사인(sine-cosine) 방식이 아닌 학습된 위치 인코딩(learned positional encoding)을 사용한다.

다음 입력 시퀀스 예시를 보자.

```
The cat sat on it because it was a nice rug.
```

디코더는 모델이 'it'에 도달한 이후의 어텐션 시퀀스를 다음처럼 마스크한다.

```
The cat sat on it(masked sequence)
```

하지만 BERT 인코더는 예측을 하기 위한 무작위 토큰 하나를 마스크한다.

```
The cat sat on it because it was a nice rug.
```

이제 멀티 어텐션 서브 층은 전체 시퀀스를 보고 셀프 어텐션 프로세스를 실행하고 마스크드 토큰을 예측한다.

이에 더해 입력 토큰을 세 가지 까다로운 방식으로 마스크하여 모델이 더 오래 학습하지만 더 나은 결과를 생성하게 한다.

데이터셋의 10%에 토큰 마스크를 하지 않는다.

```
The cat sat on it [because] it was a nice rug.
```

데이터셋의 10%에 단일 토큰을 무작위 토큰으로 교체한다.

```
The cat sat on it [often] it was a nice rug.
```

데이터셋의 80%에서 단일 토큰을 [MASK] 토큰으로 바꾼다.

The cat sat on it [MASK] it was a nice rug.

엔지니어들의 이 대담한 방식은 모델의 과적합(overfitting)을 피하고 더 효율적으로 학습시킨다.

또한 BERT는 다음 문장을 예측하는 방식으로 학습한다.

### 3.1.2.2 다음 문장 예측하기

BERT를 학습하기 위해 고안된 두 번째 방법은 다음 문장 예측(Next Sentence Prediction, NSP)이다. 다음 문장 예측의 입력은 문장 2개다. 이때 입력 데이터셋의 절반은 두 번째 문장이 문서의 실제 두 번째 문장이고, 나머지 절반은 두 번째 문장이 첫 번째 문장과 상관없는 무작위 문장이다.

새로운 토큰 두 가지가 추가된다.

- [CLS]는 첫 번째 시퀀스의 시작 부분에 추가되는 이진 분류(binary classification) 토큰으로, 두 번째 시퀀스가 첫 번째 시퀀스와 연속되는지 예측하기 위해 사용된다. 양성 샘플은 일반적인 한 쌍의 연속된 문장이고, 음성 샘플은 서로 다른 문서에서 가져온 문장이다.
- [SEP]는 시퀀스의 끝을 알리는 분리(separation) 토큰이다.

다음 예시를 보자.

The cat slept on the rug. It likes sleeping all day.

이 두 문장은 다음처럼 하나의 완전한 입력 시퀀스로 바꿀 수 있다.

[CLS] the cat slept on the rug [SEP] it likes sleep ##ing all day[SEP]

이 방식은 시퀀스 A와 시퀀스 B를 구분하기 위한 추가 인코딩 정보가 필요하다. 전체 임베딩 프로세스를 합쳐보면 다음과 같은 결과를 얻을 수 있다.

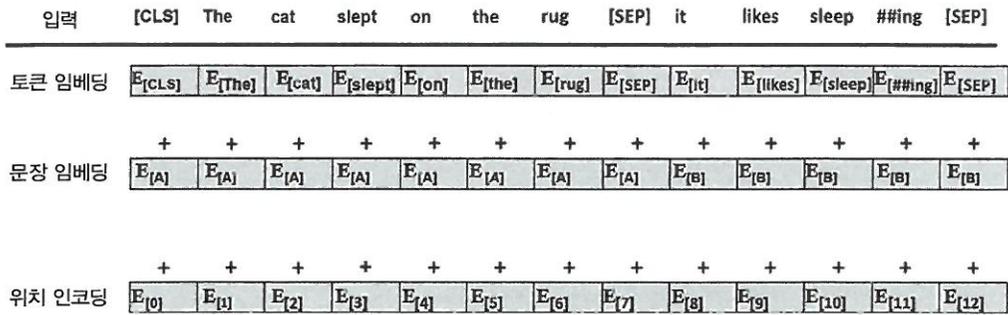


그림 3.3 입력 임베딩

입력 임베딩은 토큰 임베딩, 문장(문장, 구문, 단어) 임베딩, 위치 인코딩 임베딩을 합하여 얻는다.

BERT 모델의 입력 임베딩 및 위치 인코딩 서브 층은 다음과 같이 요약할 수 있다.

- 단어 시퀀스를 워드피스 토큰으로 분해한다.
- 마스크드 언어 모델링 학습을 위해 무작위 단어 토큰을 [MASK] 토큰으로 대체한다.
- [CLS] 분류 토큰을 입력 시퀀스의 시작 부분에 추가한다.
- [SEP] 토큰을 NSP 학습을 위해 두 문장(세그먼트, 구문) 사이에 추가한다.
- 토큰 임베딩에 더해지는 문장 임베딩은 문장 A와 문장 B가 다른 임베딩 값을 갖도록 한다.
- 위치 인코딩이 학습될 때, 오리지널 트랜스포머의 사인-코사인 위치 인코딩 방식은 적용되지 않는다.

몇 가지 추가 특징은 다음과 같다.

- BERT는 멀티 헤드 어텐션 서브 층에서 양방향 어텐션을 사용하여 토큰 간의 관계를 학습하고 이해하는 데 방대한 지평을 열어준다.
- BERT는 라벨링되지 않은 텍스트를 비지도 방식으로 사전 학습한다. 비지도 방식은 모델이 멀티 헤드 어텐션 학습 과정에서 더 열심히 생각하도록 한다. BERT는 언어가 어떻게 구축되는지 학습하게 되고 매번 사전 학습할 필요 없이 이 지식을 다운스트림 작업에 적용한다.
- 또한 BERT는 사전 학습 과정의 모든 단계를 포괄하는 지도 학습도 사용한다.

지금까지 BERT가 개선한 트랜스포머의 학습환경을 알아보았다. 이제 BERT를 사전 학습하게 된 동기와 BERT가 미세 조정과정에 어떤 도움이 되는지 살펴보자.

### 3.1.3 BERT 모델 사전 학습 및 미세 조정하기

BERT는 두 단계 프레임워크이다. 첫 번째 단계는 사전 학습이고 두 번째 단계는 그림 3.4와 같은 미세 조정이다.

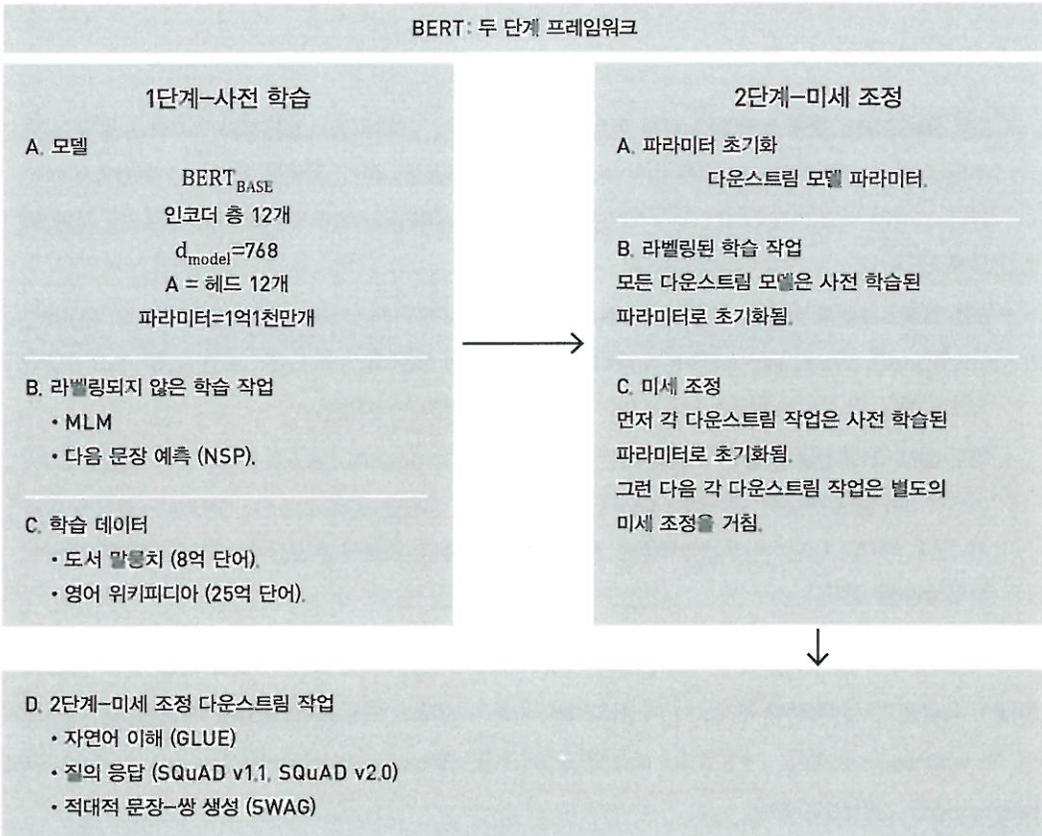


그림 3.4 BERT 프레임워크

트랜스포머 모델을 학습하는 데는 며칠은 아니더라도 몇 시간이 걸릴 수 있다. 아키텍처와 파라미터 (parameter)를 엔지니어링하고 트랜스포머 모델을 학습하기 위한 적절한 데이터셋을 선택하는 데는 상당한 시간이 걸린다.

사전 학습은 BERT 프레임워크의 첫 번째 단계로, 두 가지 서브 단계로 나눌 수 있다.

- 모델 아키텍처 정의하기: 층 수, 헤드 수, 차원 및 기타 모델의 구성 요소 정의
- MLM 및 NSP로 모델 학습하기

BERT 프레임워크의 두 번째 단계는 미세 조정이며, 이 단계도 두 가지 서브 단계로 나눌 수 있다.

- 사전 학습된 BERT 모델의 파라미터로 선택한 다운스트림 모델 초기화하기
- **Recognizing Textual Entailment(RTE)**, question answering(SQuAD v1.1, SQuAD v2.0), **Situations With Adversarial Generations(SWAG)**과 같은 특정 다운스트림 작업을 위해 파라미터 미세 조정하기

이 섹션에서는 BERT 모델을 미세 조정하는 데 필요한 정보만 다루었다. 더 자세한 내용은 다음 장에서 살펴보도록 하자.

- **4장, RoBERTa 모델 처음부터 사전 학습하기**에서 열다섯 단계에 걸쳐 BERT와 유사한 모델을 처음부터 사전 학습해 본다. 데이터를 컴파일하고 토큰라이저를 학습하고 모델을 학습하는 과정까지 살펴본다. 그러기 위해 먼저 BERT 구성 요소를 구체적으로 살펴보고 기존 모델을 미세 조정하는 것을 목표로 한다.
- **5장, 트랜스포머를 사용한 다운스트림 NLP 작업**에서 여러 다운스트림 작업과 함께 GLUE, SQuAD v1.1, SQuAD, SWAG 같은 NLP 평가 데이터셋을 살펴본다. 또한 몇 가지 다운스트림 트랜스포머 모델을 실행해 본다. 이 챕터의 목표는 다운스트림 모델을 미세 조정하는 것이다.
- **7장, GPT-3 엔진을 사용한 초인간 트랜스포머 등장**에서 OpenAI GPT-2 및 GPT-3 트랜스포머의 아키텍처와 활용에 대해 살펴본다. BERT<sub>BASE</sub>의 더 나은 성능을 위해 OpenAI GPT에 가깝게 구성하게 된다. 하지만 OpenAI 트랜스포머는 계속 진화하고 있다. 어떻게 초인간적인 NLP 수준에 도달했는지 살펴볼 것이다.

이번 장에서는 **The Corpus of Linguistic Acceptability(CoLA, 언어 적합성 말뭉치)**를 기반으로 BERT 모델을 미세 조정해 본다. 이 다운스트림 작업은 알렉스 바르슈타트(Alex Warstadt), 아만프리트 싱(Amanpreet Singh), 사무엘 R. 보우먼(Samuel R. Bowman)의 **Neural Network Acceptability Judgments**를 기반으로 한다.

문장이 문법적으로 적합한지 판단하는 BERT 모델을 미세 조정해 볼 것이다. 이 미세 조정된 모델은 상당한 수준의 언어 능력을 갖추게 될 것이다.

지금까지 우리는 BERT 아키텍처와 사전 학습 및 미세 조정프레임워크를 살펴보았다. 이제 BERT 모델을 미세 조정해 보도록 하자.

## 3.2 BERT 미세 조정하기

이 섹션에서는 BERT 모델을 미세 조정하여 적합성 판단 다운스트림 작업을 예측하고, Matthews Correlation Coefficient(MCC, 매튜 상관 계수)로 성능을 측정해 본다. 이에 대해서는 이 챕터의 매튜 상관 계수를 사용하여 평가하기 섹션에서 설명한다.

구글 코랩에서 BERT\_Fine\_Tuning\_Sentence\_Classification\_GPU.ipynb 파일을 연다(이메일 계정이 있는지 확인하자). 노트북은 이 책의 깃허브 저장소에 있는 Chapter03에 있다.

노트북의 각 셀 제목은 이 장의 각 하위 섹션 제목과 같거나 매우 유사하다. 먼저 트랜스포머 모델에서 하드웨어 제약을 고려해야 하는 이유를 살펴보자.

### 3.2.1 하드웨어 제약사항

트랜스포머 모델에는 멀티 프로세싱 하드웨어가 필요하다. 구글 코랩의 런타임(Runtime) 메뉴에서 런타임 유형 변경(Change runtime type)을 선택한 다음 하드웨어 가속기(Hardware accelerator) 드롭다운 목록에서 GPU를 선택한다.

트랜스포머 모델은 하드웨어 성능에 영향을 받는다. 이 장을 계속하기 전에 **부록 II, 트랜스포머 모델의 하드웨어 제약사항**을 읽어보는 것을 권장한다.

이 프로그램에서는 다음에 설치하게 될 허깅페이스의 모듈을 사용한다.

### 3.2.2 BERT를 위한 허깅페이스 파이토치 인터페이스 설치하기

허깅페이스는 사전 학습된 BERT 모델을 제공한다. 허깅페이스는 PreTrainedModel이라는 기본 클래스를 개발했다. 이 클래스를 설치하면 사전 학습된 모델 설정에서 모델을 불러올 수 있다.

허깅페이스는 텐서플로우 또는 파이토치를 위한 모듈을 제공한다. 개발자는 두 가지 환경 모두에 익숙한 것이 좋다. 우수한 AI 연구팀은 이 중 하나 또는 두 가지 모두를 사용한다.

이번 장에서 필요한 모듈을 다음처럼 설치한다.

```
#@title 모델 사용을 위해 허깅페이스 파이토치(PyTorch) 인터페이스 설치하기
```

```
!pip install -q transformers
```

설치가 실행되거나 요구 사항이 충족되었다는 메시지가 표시된다. 이제 프로그램에 필요한 모듈을 가져올 수 있다.

### 3.2.3 모듈 불러오기

사전 학습된 BERT tokenizer 및 BERT 모델 설정과 같이 필요한 사전 학습된 모듈을 불러온다. 시퀀스 분류 모듈과 함께 BERTAdam 옵티마이저가 불러와진다.

```
#@title 모듈 가져오기
```

```
import torch
import torch.nn as nn
from torch.utils.data import TensorDataset, DataLoader, RandomSampler,
SequentialSampler
from sklearn.model_selection import train_test_split
from transformers import BertTokenizer, BertConfig
from transformers import AdamW, BertForSequenceClassification, get_linear_
schedule_with_warmup
```

진행률 표시줄 모듈은 tqdm에서 가져온다.

```
from tqdm import tqdm, trange
```

이제 널리 사용되는 표준 파이썬 모듈을 가져온다.

```
import pandas as pd
import io
import numpy as np
import matplotlib.pyplot as plt
```

모든 것이 정상적으로 진행되면 구글 코랩에서 사용 중인 VM에 아무런 메시지가 표시되지 않을 것이다.

### 3.2.4 토치용 장치로 CUDA 지정하기

이제 토치가 **Compute Unified Device Architecture(CUDA, 컴퓨팅 통합 장치 아키텍처)**를 사용하여 멀티 헤드 어텐션 모델 작업에 NVIDIA 카드의 병렬 컴퓨팅 성능을 활용하도록 지정한다.

```
#@제목 하드웨어 검증 및 장치 속성 확인하기
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
!nvidia-smi
```

출력은 구글 코랩 설정에 따라 다를 수 있다. 자세한 설명과 스크린샷은 **부록 II, 트랜스포머 모델의 하드웨어 제약사항**을 참조하자.

이제 데이터셋을 불러오자.

### 3.2.5 데이터셋 불러오기

이제 2018년 바르슈타트(Warstadt et al.)의 논문을 기반으로 한 CoLA를 불러온다.

**General Language Understanding Evaluation(GLUE, 일반 언어 이해 평가)**는 언어 적합성을 최우선 순위 NLP 작업으로 간주한다. **5장, 트랜스포머를 사용한 다운스트림 NLP 작업**에서는 트랜스포머의 효율성을 입증하기 위해 수행해야 하는 주요 작업을 살펴본다.

노트북의 다음 셀은 필요한 파일을 자동으로 다운로드한다.

```
import os
!curl -L https://raw.githubusercontent.com/Denis2054/Transformers-for-NLP-2nd-Edition/master/Chapter03/in_domain_train.tsv --output "in_domain_train.tsv"
!curl -L https://raw.githubusercontent.com/Denis2054/Transformers-for-NLP-2nd-Edition/master/Chapter03/out_of_domain_dev.tsv --output "out_of_domain_dev.tsv"
```

파일 관리자에 해당 파일이 표시되는 것을 볼 수 있다.

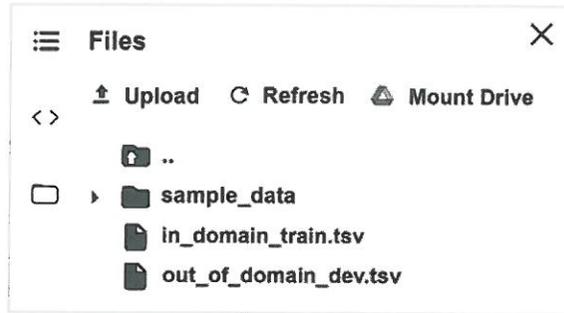


그림 3.5 데이터셋 업로드하기

이제 프로그램이 데이터셋을 불러온다.

```
#@title 데이터셋 불러오기
```

```
#데이터셋 출처: https://nyu-ml1.github.io/CoLA/
```

```
df = pd.read_csv("in_domain_train.tsv", delimiter='\\t', header=None,
names=['sentence_source', 'label', 'label_notes', 'sentence'])
df.shape
```

출력 화면에 불러온 데이터셋의 모양이 표시된다.

```
(8551, 4)
```

적합성 판단 작업 샘플 10개를 시각화하여 시퀀스가 타당한지 확인한다.

```
df.sample(10)
```

라벨링된 데이터셋 10줄을 출력하였으며, 이는 각 실행마다 변경될 수 있다.

	문장 출처	라벨	라벨_비고	문장
1742	r-67	1	NaN	그들은 통이 지불하지 않을 것이라 말했다. 하지만 아마 지불을..
937	bc01	1	NaN	비록 그가 양배추를 좋아할지라도, 프레드는 계란을 좋아한다.
5655	c_13	1	NaN	웬디 엄마의 고향은 아이슬란드다
500	bc01	0	*	존은 이기길 원한다
4596	ks08	1	NaN	나는 침대에서 어떠한 벌레도 찾지 못했다.
7412	sks13	1	NaN	학과 파티에서 그가 만났던 그녀는..
8456	ad03	0	*	피터는 늙은 돼지다
744	bc01	0	*	프랭크는 남자들에게 모두 떠나겠다고 약속했다
5420	b_73	0	*	나는 프랭크만큼 겁쟁이를 많이 봐왔다.
5749	c_13	1	NaN	우리는 부에노스 아이레스까지 차를 몰고갔다.

‘.tsv’ 파일의 각 샘플은 탭으로 구분된 4개의 열을 포함하고 있다.

- 1번째 열: 문장의 소스(코드)
- 2번째 열: 라벨(0=수용 불가, 1=수용 가능)
- 3번째 열: 작성자가 주석을 단 라벨
- 4번째 열: 분류할 문장

로컬에서 ‘.tsv’ 파일을 열어 데이터셋의 몇 가지 샘플을 읽을 수 있다. 이제 프로그램이 BERT 모델을 위한 데이터를 처리한다.

### 3.2.6 문장, 라벨 목록 만들기 및 BERT 토큰 추가하기

이제 프로그램이 이 챕터의 사전 학습 입력 환경 준비하기 섹션에 설명된 대로 문장을 생성한다.

```
#@ 문장, 라벨 리스트 생성하기 및 BERT 토큰 추가하기
sentences = df.sentencevalues
# BERT에 사용하기 위해 CLS, SEP 토큰을 각 문장의 시작과 끝에 추가하기
sentences = ["[CLS] " + sentence + " [SEP]" for sentence in sentences] labels =
df.labelvalues
```

이제 [CLS] 및 [SEP]가 추가되었다. 이제 프로그램이 토큰나이저를 활성화한다.

### 3.2.7 BERT 토큰라이저 활성화하기

이 섹션에서는 사전 학습된 BERT 토큰라이저를 초기화한다. 이렇게 하면 처음부터 학습하는 데 걸리는 시간을 절약할 수 있다.

프로그램은 대소문자가 구분되지 않은(uncased) 토큰라이저를 선택하여 활성화한 후 토큰화된 첫 번째 문장을 표시한다.

```
#@제목 BERT 토큰라이저 활성화하기
```

```
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased', do_lower_
case=True)
tokenized_texts = [tokenizer.tokenize(sent) for sent in sentences]
print ("Tokenize the first sentence:")
print (tokenized_texts[0])
```

출력 화면에는 분류 토큰과 시퀀스 분할 토큰이 출력된다.

```
Tokenize the first sentence:
['[CLS]', 'our', 'friends', 'wo', 'n', "'", 't', 'buy', 'this', 'analysis', '.', 'let', 'alone',
'the', 'next', 'one', 'we', 'propose', '.', '[SEP]']
```

이제 프로그램이 데이터를 처리한다.

### 3.2.8 데이터 처리하기

모델을 위한 데이터의 고정된 최대 길이를 정하고 이를 처리해야 한다. 데이터셋의 문장은 짧다. 따라서 프로그램은 시퀀스의 최대 길이를 128로 설정하고 시퀀스에 패딩(padding)을 적용한다.

```

#@title 데이터 처리하기
# 최대 시퀀스 길이 설정한다. 학습 데이터에서 가장 긴 시퀀스의 길이는 470이지만, 실제로는 최대 길이를
# 여유롭게 잡을 것이다.
# 논문에서 저자는 512 길이를 사용했었다.
MAX_LEN = 128
# 토큰을 BERT 사전의 인덱스 번호로 변환하기 위해서 BERT토큰라이저를 사용하라.
input_ids = [tokenizer.convert_tokens_to_ids(x) for x in tokenized_texts]
# 입력 토큰에 패딩을 추가한다.
input_ids = pad_sequences(input_ids, maxlen=MAX_LEN, dtype="long",
truncating="post", padding="post")

```

시퀀스가 처리되고 이제 프로그램이 어텐션 마스크를 생성한다.

### 3.2.9 어텐션 마스크 만들기

이제 프로세스의 까다로운 부분이 나온다. 이전 셀에서 시퀀스에 패딩을 추가했다. 하지만 모델이 패딩된 토큰에 어텐션을 주지 않도록 해야 한다!

따라서, 각 토큰에 1을, 패딩에는 0을 표시하는 마스크를 다음과 같이 생성했다.

```

#@title 어텐션 마스크 생성하기
attention_masks = []
# 각 토큰에 대해 1 마스크를 생성하고 패딩에 대해 0 마스크를 생성한다.
for seq in input_ids:
    seq_mask = [float(i) for i in seq]
    attention_masks.append(seq_mask)

```

이제 프로그램이 데이터를 분할한다.

### 3.2.10 데이터를 학습 및 검증 셋으로 분할하기

이제 이 프로그램은 데이터를 학습 및 검증 셋으로 분할하는 표준 프로세스를 수행한다.

```
#@title 데이터를 학습 및 검증 데이터셋으로 분리하기
# train_test_split을 사용해 데이터를 학습 및 검증 데이터셋으로 분리한다.
train_inputs, validation_inputs, train_labels, validation_labels = train_
test_split(input_ids, labels, random_state=2018, test_size=0.1)
train_masks, validation_masks, _ = train_test_split(attention_masks,
input_ids, random_state=2018, test_size=0.1)
```

데이터는 학습할 준비가 되었지만 여전히 토치에 맞게 조정해야 한다.

### 3.2.11 모든 데이터를 토치 텐서로 변환하기

미세 조정모델은 토치 텐서를 사용한다. 따라서 프로그램은 데이터를 토치 텐서로 변환해야 한다.

```
#@title 모든 데이터를 토치 텐서로 변환하기
# 모델에 데이터를 입력하기 위해 토치 텐서 타입으로 변환해야 한다.
train_inputs = torch.tensor(train_inputs)
validation_inputs = torch.tensor(validation_inputs)
train_labels = torch.tensor(train_labels)
validation_labels = torch.tensor(validation_labels)
train_masks = torch.tensor(train_masks)
validation_masks = torch.tensor(validation_masks)
```

변환이 끝났다. 이제 이터레이터(iterator)를 만들어야 한다.

### 3.2.12 배치 크기 선택 및 이터레이터 만들기

이 셀에서 프로그램은 배치 크기를 선택하고 이터레이터를 생성한다. 이터레이터는 메모리의 모든 데이터를 불러오는 루프(loop)를 피하는 현명한 방법이다. 이터레이터를 토치 DataLoader와 결합하면 메모리 오류 없이 대규모 데이터셋을 일괄(batch) 학습할 수 있다.

이 모델에서 배치 크기는 32이다.

```
#@title 배치 사이즈 선택하기 및 이터레이터(Iterator) 생성하기
# 학습에 사용될 배치사이즈를 선택하자. BERT를 특정 테스트를 위해 미세 조정하기 위해서는 16 또는 32의
# 배치 사이즈를 추천한다.
batch_size = 32
# 토치 DataLoader를 사용해 데이터 이터레이터를 생성한다. 이렇게 하면 학습 과정에서 루프를 사용하는
# 것보다 메모리 사용을 줄일 수 있다.
# 이터레이터를 사용하면 전체 데이터를 메모리에 한번에 로드할 필요 없다.
train_data = TensorDataset(train_inputs, train_masks, train_labels)
train_sampler = RandomSampler(train_data)
train_dataloader = DataLoader(train_data, sampler=train_sampler, batch_
size=batch_size)
validation_data = TensorDataset(validation_inputs, validation_masks,
validation_labels)
validation_sampler = SequentialSampler(validation_data)
validation_dataloader = DataLoader(validation_data, sampler=validation_
sampler, batch_size=batch_size)
```

데이터가 처리되고 모든 준비가 완료되었다. 이제 프로그램이 BERT 모델을 불러오고 설정할 수 있다.

### 3.2.13 BERT 모델 설정하기

이제 프로그램이 대소문자가 구분되지 않은(uncased) BERT 설정을 초기화한다.

```
#@title BERT 모델 설정하기
#BERT bert-base-uncased 설정 모델을 초기화하기
#@title 트랜스포머 설치하기
try:
    import transformers
except:
    print("Installing transformers")
    !pip -qq install transformers
from transformers import BertModel, BertConfig
configuration = BertConfig()
# bert-base-uncased-style 설정을 사용해 모델을 초기화하기
```

```

model = BertModel(configuration)
# 모델 설정 불러오기
configuration = model.config
print(configuration)

```

출력 화면에는 다음과 유사한 주요 허깅페이스 파라미터가 표시된다(라이브러리는 자주 업데이트된다).

```

BertConfig {
  "attention_probs_dropout_prob": 0.1,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 0,
  "type_vocab_size": 2,
  "vocab_size": 30522
}

```

주요 파라미터를 살펴보자.

- **attention\_probs\_dropout\_prob**: 0.1은 어텐션 확률에 0.1의 드롭아웃 비율을 적용한다.
- **hidden\_act**: "gelu"는 인코더에 비선형 활성화 함수인 Gaussian Error Linear Units를 적용한다. 입력은 크기에 따라 가중치가 부여되므로 비선형적이다.
- **hidden\_dropout\_prob**: 0.1은 완전 연결 층에 적용되는 드롭아웃 비율이다. 완전 연결은 임베딩, 인코더 및 풀러(pooler) 층에서 찾을 수 있다. 출력이 항상 시퀀스의 내용을 잘 반영하는 것은 아니다. 은닉 상태의 시퀀스를 풀링(pooling)하면 출력 시퀀스가 개선된다.
- **hidden\_size**: 768은 인코딩된 층과 풀러 층의 크기다.
- **initializer\_range**: 0.02는 가중치 행렬을 초기화할 때의 표준 편차 값이다.

- `intermediate_size`: 3072는 인코더의 순방향 층의 크기다.
- `layer_norm_eps`:  $1e-12$ 는 층 정규화 층의 엡실론 값이다.
- `max_position_embeddings`: 512는 모델이 사용하는 최대 길이다.
- `model_type`: "bert"는 모델의 이름이다.
- `num_attention_heads`: 12는 헤드의 수이다.
- `num_hidden_layers`: 12는 층의 수이다.
- `pad_token_id`: 0은 학습시키지 않기 위한 패딩 토큰의 ID 이다.
- `type_vocab_size`: 2는 시퀀스를 식별하는 `token_type_ids`의 크기다. 예를 들어, the dog[SEP] The cat.[SEP]는 토큰 ID [0,0,1,1,1]로 나타낼 수 있다.
- `vocab_size`: 30522는 모델에서 `input_ids` 표현하기 위해 사용하는 서로 다른 토큰의 수이다.

이러한 파라미터를 염두에 두고 사전 학습된 모델을 불러오도록 하자.

### 3.2.14 대소문자가 구분되지 않은 허깅페이스 BERT 모델 불러오기

이제 프로그램이 사전 학습된 BERT 모델을 불러온다.

**#@title 허깅페이스 BERT uncased 모델 불러오기**

```
model = BertForSequenceClassification.from_pretrained("bert-base-uncased",
num_labels=2)
model = nn.DataParallel(model)
model.to(device)
```

모델을 정의하고, 병렬 처리를 정의하고, 모델을 디바이스로 보낸다. 자세한 설명은 **부록 II, 트랜스포머 모델의 하드웨어 제약사항**을 참조하자.

이 사전 학습된 모델은 필요한 경우 추가로 학습할 수 있다. 각 서브 층의 파라미터를 시각화하기 위해 아키텍처를 자세히 살펴보도록 하자.

```
BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
```

```

(LayerNorm): BertLayerNorm()
(dropout): Dropout(p=0.1, inplace=False)
)
(encoder): BertEncoder(
(layer): ModuleList(
(0): BertLayer(
(attention): BertAttention(
(self): BertSelfAttention(
(query): Linear(in_features=768, out_features=768, bias=True)
(key): Linear(in_features=768, out_features=768, bias=True)
(value): Linear(in_features=768, out_features=768, bias=True)
(dropout): Dropout(p=0.1, inplace=False)
)
)
(output): BertSelfOutput(
(dense): Linear(in_features=768, out_features=768, bias=True)
(LayerNorm): BertLayerNorm()
(dropout): Dropout(p=0.1, inplace=False)
)
)
(intermediate): BertIntermediate(
(dense): Linear(in_features=768, out_features=3072, bias=True)
)
(output): BertOutput(
(dense): Linear(in_features=3072, out_features=768, bias=True)
(LayerNorm): BertLayerNorm()
(dropout): Dropout(p=0.1, inplace=False)
)
)
(1): BertLayer(
(attention): BertAttention(
(self): BertSelfAttention(
(query): Linear(in_features=768, out_features=768, bias=True)
(key): Linear(in_features=768, out_features=768, bias=True)
(value): Linear(in_features=768, out_features=768, bias=True)
(dropout): Dropout(p=0.1, inplace=False)
)
)
(output): BertSelfOutput(

```

```

(dense): Linear(in_features=768, out_features=768,bias=True)
(LayerNorm): BertLayerNorm()
(dropout): Dropout(p=0.1, inplace=False)
)
)
(intermediate): BertIntermediate(
(dense): Linear(in_features=768, out_features=3072, bias=True)
)
(output): BertOutput(

(dense): Linear(in_features=3072, out_features=768, bias=True)
(LayerNorm): BertLayerNorm()
(dropout): Dropout(p=0.1, inplace=False)
)
)
)

```

이제 옵티마이저의 주요 파라미터를 살펴보자.

### 3.2.15 파라미터 그룹 옵티마이저

이제 프로그램이 모델 파라미터 옵티마이저를 초기화한다. 모델 미세 조정은 사전 학습된 모델 파라미터 값(이름이 아님)을 초기화하는 것으로 시작한다.

파라미터 옵티마이저는 과적합을 방지하기 위한 가중치 감소율을 포함하고 일부 파라미터는 제외된다.

학습 루프를 위한 모델 파라미터를 준비해 보자.

```

##@title 그룹 파라미터 최적화하기
# 이 코드는 아래 링크를 참조했다.
# https://github.com/huggingface/transformers/blob/5bfcd0485ece086ebcbed2d0
08813037968a9e58/examples/run_glue.py#L102
# 다음 토큰을 이름에 포함한 파라미터에 가중치 감쇠(weight decay)를 적용하지 말아라
# (여기서, BERT 는 'gamma' 또는 'beta' 파라미터를 사용하지 않는다. 'bias' 토큰을 포함한 파라미
터만 사용한다)
param_optimizer = list(model.named_parameters())
no_decay = ['bias', 'LayerNormweight']
# 'weight' 파라미터를 'bias' 파라미터와 분리해라

```

```

# - 'weight'파라미터에 대해 'weight_decay_rate'를 0.01 로 설정한다.
# - 'bias'파라미터에 대해 'weight_decay_rate'를 0.0으로 설정한다.
optimizer_grouped_parameters = [
# 'bias', 'gamma', 'beta'를 포함하지 않은 파라미터를 필터링한다
{'params': [p for n, p in param_optimizer if not any(nd in n for nd in
no_decay)], 'weight_decay_rate': 0.1},
# 'bias', 'gamma', 'bea'를 포함한 파라미터를 필터링한다
{'params': [p for n, p in param_optimizer if any(nd in n for nd in no_decay)],
'weight_decay_rate': 0.0}
]
# 참고 - 'optimizer_grouped_parameters'는 파라미터 값만 포함하고 있으며, 이름은 포함하고
있지 않다.

```

파라미터를 학습 루프에 사용할 준비가 됐다.

### 3.2.16 학습 루프의 하이퍼파라미터

학습 루프의 하이퍼파라미터는 사소해 보이지만 매우 중요하다. 예를 들어 아담(Adam)은 가중치 감소(weight decay)를 활성화하고 워밍업 단계를 거친다.

학습률(learning rate, lr)과 워밍업 비율(warmup)은 최적화 단계 초기에 매우 작은 값으로 설정하고 일정 횟수를 반복한 후에 점차적으로 증가해야 한다. 이렇게 함으로써 경사가 커지고 최적화 목표를 지나치는 것을 방지할 수 있다.

일부 연구자들은 층 정규화 이전의 서브 층 출력에서는 워밍업 비율이 필요하지 않다고 주장한다. 이 문제를 해결하려면 많은 실험이 필요하다.

옵티마이저는 아담의 BERT 버전인 BertAdam이다.

```

#@title 학습 과정에 사용할 하이퍼파라미터
optimizer = BertAdam(optimizer_grouped_parameters,
                    lr=2e-5,
                    warmup=1)

```

이 프로그램에 정확도 측정 기능을 추가하여 예측값과 라벨을 비교할 수 있다.

```

# 정확도 측정 함수 만들기
# 라벨 대비 예측값의 정확도를 측정하기 위한 함수
def flat_accuracy(preds, labels):
    pred_flat = np.argmax(preds, axis=1).flatten()
    labels_flat = labels.flatten()
    return np.sum(pred_flat == labels_flat) / len(labels_flat)

```

데이터와 파라미터가 준비되었다. 이제 학습 루프를 활성화하자!

### 3.2.17 학습 루프

학습 루프는 표준 학습 프로세스를 따른다. 에포크(epoch) 수는 4로 설정했으며, 손실 및 정확도를 플롯(plot)할 것이다. 학습 루프는 dataloader를 사용하여 배치를 불러온 후 학습 프로세스를 수행하고 측정 및 평가한다.

이 코드는 먼저 손실과 정확도를 저장하기 위해 train\_loss\_set을 초기화한 후 에포크에 따라 학습을 시작하고 표준 학습 루프를 실행한다.

```

#@title 학습 루프
t = []
# 그래프를 그리기 위해 손실 및 정확도 저장한다
train_loss_set = []
# 학습 에포크 수 (필자는 2~4 사이의 값을 추천한다)
epochs = 4
# trange 는 파이썬의 range 함수에 대한 tqdm 래퍼(wrapper)이다.
for _ in trange(epochs, desc="Epoch"):
    .../...
    tmp_eval_accuracy = flat_accuracy(logits, label_ids)
    eval_accuracy += tmp_eval_accuracy
    nb_eval_steps += 1
    print("Validation Accuracy: {}".format(eval_accuracy/nb_eval_steps))

```

for \_ in trange(epochs, desc="Epoch")처럼 trange 래퍼를 사용하여 각 에포크 정보를 화면에 출력한다.

```

***output***
Epoch: 0%|          | 0/4 [00:00<?, ?it/s]
Train loss: 0.5381132976395461
Epoch: 25%|██████    | 1/4 [07:54<23:43, 474.47s/it]
Validation Accuracy: 0.788966049382716
Train loss: 0.315329696132929
Epoch: 50%|██████████ | 2/4 [15:49<15:49, 474.55s/it]
Validation Accuracy: 0.836033950617284
Train loss: 0.1474070605354314
Epoch: 75%|███████████ | 3/4 [23:43<07:54, 474.53s/it]
Validation Accuracy: 0.814429012345679
Train loss: 0.07655430570461196
Epoch: 100%|███████████ | 4/4 [31:38<00:00, 474.58s/it]
Validation Accuracy: 0.810570987654321

```

트랜스포머 모델은 매우 빠르게 진화하고 있지만, 중단 메시지나 오류가 발생할 수도 있다. 허깅페이 스도 마찬가지로 이런 일이 발생하면 코드 수정이 필요하다.

모델이 학습되었다. 이제 학습을 평가한 내용을 출력해 보자.

### 3.2.18 학습 평가하기

손실 및 정확도 값은 학습 루프를 시작할 때 정의한 대로 `train_loss_set`에 저장되었다.

이제 프로그램이 측정값 그래프를 그릴 것이다.

```

#@title 학습 평가하기
plt.figure(figsize=(15,8))
plt.title("Training loss")
plt.xlabel("Batch")
plt.ylabel("Loss")
plt.plot(train_loss_set)
plt.show()

```

결과 그래프는 학습 과정이 효율적으로 잘 진행된 것을 보여준다.

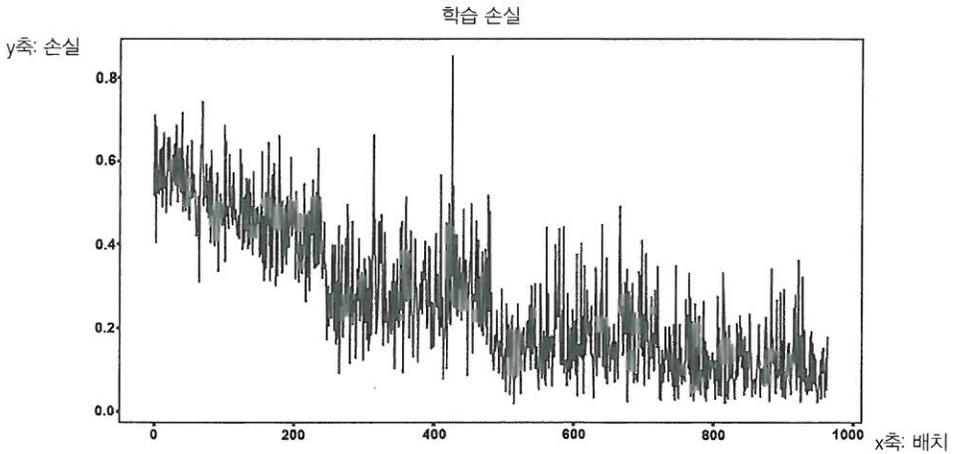


그림 3.6 배치당 학습 손실

모델이 미세 조정되었다. 이제 예측을 수행해 보자.

### 3.2.19 홀드아웃 데이터셋을 사용하여 예측 및 평가하기

BERT 다운스트림 모델은 `in_domain_train.tsv` 데이터셋으로 학습되었다. 이제 프로그램은 `out_of_domain_dev.tsv` 파일에 있는 홀드아웃(테스트) 데이터셋을 사용하여 예측을 수행한다. 예측 목표는 문장이 문법적으로 올바른지 판단하는 것이다.

다음 코드는 학습 데이터에 적용된 데이터 준비 프로세스가 홀드아웃 데이터셋의 코드에도 반복되는 것을 보여준다.

```
#@title Holdout 데이터셋을 사용해 예측 및 평가하기
df = pd.read_csv("out_of_domain_dev.tsv", delimiter='\t', header=None,
names=['sentence_source', 'label', 'label_notes', 'sentence'])
# 문장과 라벨 리스트를 생성하기
sentences = df.sentence.values
# BERT가 정상적으로 동작하게 하기 위해서는 스페셜 토큰을 각 문장의 시작과 끝에 추가해줘야만 한다.
sentences = ["[CLS] " + sentence + " [SEP]" for sentence in sentences] labels =
df.label.values
tokenized_texts = [tokenizer.tokenize(sent) for sent in sentences]
.../...
```

그런 다음 프로그램은 `dataloader`를 사용하여 일괄 예측을 수행한다.

```

# 예측하기
for batch in prediction_dataloader:
    # 배치를 GPU에 추가한다
    batch = tuple(t.to(device) for t in batch)
    # 데이터로더로부터 얻어진 인풋을 분리한다.
    b_input_ids, b_input_mask, b_labels = batch
    # 예측 속도를 높이고 메모리 사용량을 줄이기 위해 모델이 기울기(graident)를 계산하거나 저장하지
    # 않도록한다.
    with torch.no_grad():
        # 순방향(Forward)으로 입력을 통과시키고 로짓(logit) 예측값을 계산한다.
        logits = model(b_input_ids, token_type_ids=None, attention_mask=b_input_
        mask)

```

예측의 로짓(logits)과 라벨이 CPU로 이동된다.

```

# 로짓과 라벨을 CPU로 이동한다.
logits = logits['logits'].detach().cpu().numpy()
label_ids = b_labelsto('cpu').numpy()
# 예측과 정답 라벨이 저장된다.
predictions.append(logits)
true_labels.append(label_ids)

```

이제 프로그램이 예측을 평가한다.

### 3.2.20 매튜 상관 계수를 사용하여 평가하기

Matthews Correlation Coefficient(MCC, 매튜 상관 계수)는 처음엔 이진 분류(binary classification)의 품질을 측정하기 위해 설계되었지만 다중 분류(multi-class) 상관 계수로 수정할 수 있다. 각 예측마다 4가지 확률을 가지는 이중 클래스 분류를 만들 수 있다.

- $T_p$  = 참 양성
- $T_N$  = 참 음성
- $F_p$  = 거짓 양성
- $F_N$  = 거짓 음성

이는 생화학자인 매튜(Brian W. Matthews)가 1975년에 전임자의 phi 함수에서 영감을 받아 고안했다. 그 이후로 다음과 같은 형태로 발전했다.

$$MCC = \frac{T_p \times T_n - F_p \times F_n}{\sqrt{(T_p + F_p)(T_p + F_n)(T_n + F_p)(T_n + F_n)}}$$

MCC는 -1에서 +1 사이의 값을 도출한다. +1은 완벽하게 예측한 것이다. -1은 완전 반대로 예측한 것이고 0은 무작위로 예측한 것이다.

GLUE는 MCC로 언어 적합성을 평가한다. MCC는 `sklearn.metrics`에서 가져온다.

```
#@title 매튜 상관 계수 사용해 평가하기
# 테스트 배치를 가져와 매튜 상관 계수를 사용해 평가한다
from sklearn.metrics import matthews_corrcoef
```

예측셋이 생성되었다.

```
matthews_set = []
```

계산된 MCC 값은 `matthews_set`에 저장된다.

```
for i in range(len(true_labels)):
    matthews = matthews_corrcoef(true_labels[i],
                                np.argmax(predictions[i], axis=1).flatten())
    matthews_set.append(matthews)
```

라이브러리 및 모듈 버전 변경으로 인해 메시지가 표시될 수 있다. 최종 점수는 전체 데이터셋 기준이지만, 개별 배치 점수를 살펴보고 배치 간 지표의 변동성을 파악해 보자.

### 3.2.21 개별 배치의 점수

개별 배치 점수를 확인해 보자.

```
#@title 각 배치의 점수
matthews_set
```

출력은 예상대로 -1에서 +1 사이의 MCC 값이다.

```
[0.049286405809014416,
 -0.2548235957188128,
 0.4732058754737091,
 0.30508307783296046,
 0.3567530340063379,
 0.8050112948805689,
 0.23329882422520506,
 0.47519096331149147,
 0.4364357804719848,
 0.4700159919404217,
 0.7679476477883045,
 0.8320502943378436,
 0.5807564950208268,
 0.5897435897435898,
 0.38461538461538464,
 0.5716350506349809,
 0.0]
```

거의 모든 MCC 값이 양수인 것은 긍정적이다. 이제 전체 데이터셋의 평가 결과를 살펴보자.

### 3.2.22 전체 데이터셋의 매튜 평가

MCC는 분류 모델을 평가하는 실용적인 방법이다. 이제 프로그램은 전체 데이터셋의 실제 값을 집계한다.

```
#@title 전체 데이터셋에 대해 매튜 상관 관계 평가하기
# 전체 데이터셋의 매튜 상관 계수를 수집하기 위해 예측값과 true 값을 펼친다.
flat_predictions = [item for sublist in predictions for item in sublist] flat_
predictions = np.argmax(flat_predictions, axis=1).flatten() flat_true_labels =
[item for sublist in true_labels for item in sublist] matthews_corrcoef
(flat_true_labels, flat_predictions)
```

MCC는 -1에서 +1 사이의 상관관계 값을 도출한다. 0은 평균 예측, -1은 역 예측, 1은 완벽 예측이다. 다음 출력 사례에서 MCC는 양수이고, 이는 모델과 데이터셋이 상관관계가 있음을 나타낸다.

```
0.45439842471680725
```

최종적으로 미세 조정된 BERT 모델이 긍정적인 평가를 받았고, 우리는 BERT 학습 프레임워크의 전반적인 개요를 파악해 보았다.

## 3.3 정리하기

BERT는 트랜스포머에 양방향 어텐션을 도입했다. 왼쪽에서 오른쪽으로 시퀀스를 예측하고 이후의 토큰을 마스킹하여 모델을 학습하는 데는 심각한 한계가 있다. 만약 마스크드 시퀀스에 우리가 찾고자 하는 의미가 포함되면 모델에서 오류가 발생하게 된다. 이를 해결하기 위해 BERT는 시퀀스의 모든 토큰을 동시에 처리한다.

트랜스포머의 인코더 스택만 사용하는 BERT 아키텍처를 살펴보았다. BERT는 두 단계 프레임워크로 설계되었다. 프레임워크의 첫 번째 단계는 모델을 사전 학습하는 것이고, 두 번째 단계는 모델을 미세 조정하는 것이다. 우리는 적합성 판단 다운스트림 작업을 위해 미세 조정된 BERT 모델을 구축했다. 미세 조정프로세스는 다음과 같은 단계를 거쳤다. 먼저 데이터셋을 불러오고 모델에 필요한 사전 학습된 모듈을 불러왔다. 그런 다음 모델을 학습시키고 성능을 측정했다.

사전 학습된 모델을 미세 조정하는 것은 다운스트림 작업을 처음부터 학습하는 것보다 적은 머신 리소스를 필요로 한다. 미세 조정된 모델은 다양한 작업을 수행할 수 있다. BERT는 우리가 단 두 가지 작업으로 사전 학습시킬 수 있다는 것을 증명했는데, 이는 이 자체만으로도 놀라운 성과지만 사전 학습된 BERT 모델의 파라미터를 기반으로 다양한 미세 조정모델을 생성해 내는 것 또한 매우 놀라운 일이다.

**7장, GPT-3 엔진을 사용한 초인간 트랜스포머 등장**에서는 OpenAI가 미세 조정없이도 제로 샷(zero-shot) 수준에 도달했음을 보여준다.

이번 장에서는 BERT 모델을 미세 조정했다. 다음 장인 **4장, RoBERTa 모델** 처음부터 사전 학습하기에서는 BERT 프레임워크를 더 자세히 살펴보고 사전 학습된 BERT와 유사한 모델을 처음부터 구축해 보도록 하자.

## 3.4 문제

01. BERT는 Bidirectional Encoder Representations from Transformers를 축약한 단어이다. (참/거짓) 참 / 거짓
02. BERT는 두 단계 프레임워크이다. 첫 번째 단계는 사전 학습이고 두 번째 단계는 미세 조정이다. 참 / 거짓
03. BERT 모델 미세 조정은 파라미터를 처음부터 학습하는 것을 의미한다. 참 / 거짓
04. BERT는 모든 다운스트림 작업을 사전 학습한다. 참 / 거짓
05. BERT는 Masked Language Modeling (MLM, 마스크드 언어 모델링)을 사용하여 사전 학습한다. 참 / 거짓
06. BERT는 Next Sentence Predictions (NSP, 다음 문장 예측하기)을 사용하여 사전 학습한다. 참 / 거짓
07. BERT는 수학적인 함수를 사전 학습한다. 참 / 거짓
08. 질문-답변(question-answer) 작업은 다운스트림 작업이다. 참 / 거짓
09. BERT 사전 학습 모델에는 토큰화가 필요하지 않다. 참 / 거짓
10. BERT 모델 미세 조정은 사전 학습보다 시간이 덜 걸린다. 참 / 거짓

## 3.5 참고 문헌

- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin, 2017, Attention Is All You Need: <https://arxiv.org/abs/1706.03762>
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, 2018, BERT: Pretraining of Deep Bidirectional Transformers for Language Understanding: <https://arxiv.org/abs/1810.04805>
- Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman, 2018, Neural Network Acceptability Judgments: <https://arxiv.org/abs/1805.12471>
- 언어 적합성 말뭉치(The Corpus of Linguistic Acceptability, CoLA): <https://nyu-ml.github.io/CoLA/>
- 허깅페이스 모델:  
[https://huggingface.co/transformers/pretrained\\_models.html](https://huggingface.co/transformers/pretrained_models.html)  
[https://huggingface.co/transformers/model\\_doc/bert.html](https://huggingface.co/transformers/model_doc/bert.html)  
[https://huggingface.co/transformers/model\\_doc/roberta.html](https://huggingface.co/transformers/model_doc/roberta.html) [https://huggingface.co/transformers/model\\_doc/distilbert.html](https://huggingface.co/transformers/model_doc/distilbert.html)

# 04장

---

## RoBERTa 모델 처음부터 사전 학습하기

---

- 4.1 토큰라이저 학습하기 및 트랜스포머 사전 학습하기
- 4.2 처음부터 KantaiBERT 구축하기
- 4.3 다음 단계
- 4.4 정리하기
- 4.5 문제
- 4.6 참고 문헌

이 장에서는 RoBERTa 모델을 처음부터 만들어 보자. 이 모델을 만들기 위해 BERT 모델을 만들 때 사용하는 구성 요소들을 활용할 것이다. 또한 사전 학습된 토큰라이저나 모델도 사용하지 않을 것이다. 그 대신 이번 장에서 설명하는 15단계에 따라 RoBERTa 모델을 만들어 보게 될 것이다.

이전 장에서 익힌 트랜스포머 지식으로 마스크드 토큰 언어 모델링 모델을 단계별로 만들어볼 것이다. **2장, 트랜스포머 모델 아키텍처 살펴보기**에서는 트랜스포머 모델의 구성 요소를 살펴봤다. **3장, BERT 모델 미세 조정하기**에서는 사전 학습된 BERT 모델을 미세 조정해 봤다.

이 장에서는 허깅페이스의 모듈을 기반으로 하는 노트북을 사용하여 사전 학습된 트랜스포머 모델을 처음부터 다시 구축하는 데 중점을 둔다. 모델 이름은 KantaiBERT이다.

먼저 이 장에서 설명을 위해 만들어진 임마누엘 칸트의 저서 모음집을 로드할 것이다. 이 과정에서 데이터를 어떻게 얻었는지 확인할 수 있을 것이다. 또한 자신만의 데이터셋을 만드는 방법도 볼 수 있을 것이다.

KantaiBERT 학습을 위해 자체 토큰라이저를 처음부터 학습시킬 것이다. 이 과정에서 사전 학습에서 사용할 병합 및 어휘 파일을 구축할 것이다.

그런 다음, KantaiBERT 학습을 위한 데이터셋을 처리하고, 트레이너를 초기화하며, 모델을 학습할 것이다.

마지막으로, KantaiBERT는 학습한 모델로 다운스트림 작업 실험을 하고 마스크 부분을 임마누엘 칸트의 논리로 채워볼 것이다.

이 장이 끝나면 트랜스포머 모델을 처음부터 구축하는 방법을 알게 될 것이다. GPT-3와 같은 모델을 사용하기 위해서는 단순한 구현 이외의 지식이 필요하다. 이번 장으로 강력한 사전 학습된 트랜스포머를 활용하기에 충분한 지식을 갖출 수 있는 것이다. 이 장을 이해하면 **7장, GPT-3 엔진을 사용한 초인간 트랜스포머의 등장**을 이해하는 데 도움이 된다.

이 장에서는 다음 주제들을 다룬다.

- RoBERTa 및 DistilBERT 유사 모델
- 토큰라이저를 처음부터 학습하는 방법
- 바이트 단위 바이트 쌍 인코딩(Byte-level byte-pair encoding)
- 학습한 토큰라이저를 파일에 저장하기
- 사전 학습 프로세스를 위해 토큰라이저를 다시 생성하기
- RoBERTa 모델 초기화하기
- 모델 구성 살펴보기
- 8천만 개의 모델 파라미터 탐색하기
- 학습 데이터셋 구축하기
- 트레이너 초기화하기
- 모델 사전 학습하기
- 모델 저장하기
- MLM(마스크드 언어 모델링)의 다운스트림 작업에 모델 적용하기

먼저, 우리가 구축할 트랜스포머 모델을 설명해 보겠다.

## 4.1

# 토큰나이저 학습하기 및 트랜스포머 사전 학습하기

이번 장에서는 허깅페이스에서 제공하는 구성 요소를 사용하여 KantaiBERT라는 이름의 트랜스포머 모델을 학습한다. 3장, BERT 모델 미세 조정하기에서 우리가 사용할 모델의 구성 요소에 대한 이론을 다뤘었다.

이전 장에서 습득한 지식을 바탕으로 KantaiBERT에 대해 설명하겠다.

KantaiBERT는 BERT의 아키텍처를 기반으로 하는 RoBERTa와 유사한 모델이다.

3장에서 살펴본 것처럼 초기 트랜스포머 모델에 혁신적인 기능을 도입해, BERT 모델이 탄생했다. RoBERTa는 사전 학습 메커니즘을 개선하여 다운스트림 작업에서 트랜스포머 성능을 향상시켰다.

예를 들어, RoBERTa는 워드피스 토큰나이저를 사용하지 않고 바이트 수준의 바이트 쌍 인코딩(Byte-Pair Encoding)을 사용한다. 이 방법은 다양한 BERT 및 BERT 유사 모델에 널리 사용되었다.

이 장에서는 BERT와 마찬가지로 마스크드 언어 모델링(MLM)을 사용하여 KantaiBERT를 학습한다. 마스크드 언어 모델링은 시퀀스에서 단어를 마스크하는 언어 모델링 기법이다. 이 기법을 사용해 트랜스포머 모델은 마스크된 단어를 예측하게 된다.

KantaiBERT는 6개의 층, 12개의 헤드, 84,095,008개의 파라미터로 구성된 작은 모델이다. 8,400만 개 파라미터가 많다고 생각할 수 있지만, 파라미터가 12개의 헤드에 분산되어 있기 때문에 비교적 작은 모델이라고 할 수 있다. 작은 모델은 사전 학습 결과를 확인하기 위해 몇 시간을 기다리지 않고도 각 단계를 실시간으로 볼 수 있다는 장점이 있다.

KantaiBERT는 6개의 층과 12개의 헤드로 구성된 동일한 아키텍처를 가지고 있기 때문에 DistilBERT와 유사한 모델이다. DistilBERT는 BERT의 증류(distillation) 기법을 적용한 모델이다. 이름에서 알 수 있듯이 DistilBERT는 RoBERTa 모델보다 더 적은 수의 파라미터를 가지고 있다. 따라서 실행 속도는 훨씬 빠르지만 결과의 정확도는 RoBERTa 모델에 비해 약간 떨어진다.

대형 모델이 뛰어난 성능을 발휘한다는 것은 잘 알려진 사실이다. 하지만 스마트폰에서 모델을 실행하려면 어떻게 해야 할까? 이를 위해서는 모델 소형화가 필수적이다. 트랜스포머의 구현 방식은 정형화 되어 있다. 따라서 허깅페이스의 경량 버전 BERT를 사용하는 것도 좋은 방법이다. 더 적은 파라미터

를 사용하여 모델을 경량화하면 사전 학습의 이점을 최대한 활용하고, 다양한 다운스트림 작업에 효율적으로 적용할 수 있다.

“스마트폰에서도 실행될 작은 모델”처럼 다양한 모델을 살펴보는 것도 중요하지만, 미래의 트랜스포머는 API가 될 것이다. 7장, GPT-3 엔진을 사용한 초인간 트랜스포머 등장에서 관련 내용을 살펴볼 것이다.

KantaiBERT는 GPT-2에서 사용하는 것과 같은 바이트 수준의 바이트 쌍 인코딩 토큰라이저를 구현할 것이다. 스페셜토큰은 RoBERTa 모델과 동일한 것을 사용할 것이다. BERT 모델은 대부분 워드 피스 토큰라이저를 사용한다.

토큰이 문장의 어느 부분에 속하는지 나타내는 토큰 유형 ID<sup>9</sup>는 없다. 문장은 분리 토큰 </s>으로 구분한다.

커스텀 데이터셋과 MLM 예제 소스 코드를 사용해 토큰라이저와 트랜스포머 모델을 학습시켜 KantaiBERT를 실행시켜 볼 것이다.

자, 이제 트랜스포머 모델을 처음부터 구현해 보자.

9 역주. BERT와 같은 모델에서는 입력으로 2개의 문장이 주어지는 경우가 있다. 입력으로 문장이 2개가 주어진 경우, 두 문장을 구분할 수 있도록 별도의 id를 부여한다. 2개의 문장이 입력으로 주어지면, 첫 번째 문장에 해당하는 토큰들의 인덱스에는 모두 0을, 두 번째 문장에 해당하는 토큰들의 인덱스에는 모두 1을 부여한다. 이러한 0과 1과 같은 값을 토큰 유형 ID라 한다.

## 4.2

# 처음부터 KantaiBERT 구축하기

15단계에 걸쳐 KantaiBERT를 구축하고 마스크드 언어 모델링 예제를 실행해 보자.

구글 코랩(Google Colaboratory)을 열고(Gmail 계정이 필요하다) 이번 장의 디렉터리에 있는 KantaiBERT.ipynb 파일을 업로드해라.

이 섹션의 15개 단계 제목은 노트북 셀의 제목과 같도록 맞춰져 있다.

데이터셋을 먼저 로드하는 것으로 시작해 보자.

### 4.2.1 1단계: 데이터셋로드하기

준비된 데이터셋을 사용해 트랜스포머를 학습하고 객관적으로 비교할 수 있다. 5장, 트랜스포머를 사용한 다운스트림 NLP 작업에서는 다양한 데이터셋을 살펴보겠다. 그러나 이번 챕터의 목표는 몇 시간을 기다리지 않고도 실시간으로 실행할 수 있는 노트북 셀을 사용하여 트랜스포머의 학습 과정을 이해하는 것이다.

계몽 시대의 전형적인 인물이었던 독일 철학자 임마누엘 칸트(1724–1804)의 작품들을 사용해 보자. 모델이 학습한 칸트의 논리를 다운스트림 추론 작업에 적용해 보는 것이 목적이다.

프로젝트 구텐베르크(<https://www.gutenberg.org>)에서 다양한 무료 전자책을 텍스트 형식으로 다운로드할 수 있다. 다른 책을 사용해 맞춤형 데이터셋을 만들 수도 있다.

임마누엘 칸트의 다음 세 권의 책을 kant.txt라는 텍스트 파일로 편집했다.

- 순수이성비판(The Critique of Pure Reason)
- 실천이성비판(The Critique of Practical Reason)
- 도덕론의 기본 원리(Fundamental Principles of the Metaphysic of Morals)

kant.txt는 이번 장의 트랜스포머 모델을 학습하기 위한 작은 데이터셋으로 사용할 것이며, 실험 수준의 결과를 얻게 될 것이다. 실제 프로젝트라면 임마누엘 칸트, 르네 데카르트, 파스칼, 라이프니츠의 모든 작품들을 사용했을 것이다.

텍스트 파일은 다음과 같은 내용으로 구성되어 있다.

...For it is in reality vain to profess `_indifference_` in regard to such inquiries, the object of which cannot be indifferent to humanity.

(...이러한 탐구에 대해 `_무관심_`을 주장하는 것은 사실상 헛된 일이다. 이는 인류에게 무관할 수 없는 대상이기 때문이다.)

데이터셋은 KantaiBERT.ipynb 노트북의 첫 번째 셀을 실행하면 깃허브에서 자동으로 다운로드할 수 있다. 또한 코랩 파일 관리자를 사용하여 깃허브의 이 챕터의 디렉터리에있는 kant.txt를 로드 할 수도 있다. 이 경우 curl을 사용하여 깃허브에서 다운로드한다.

#### #@title 1단계: 데이터셋 로드하기

##### #1. 코랩 파일 관리자를 사용해 kant.txt 로드하기

##### #2. 깃허브에서 파일 다운로드하기

```
!curl -L https://raw.githubusercontent.com/Denis2054/Transformers-for-NLP-2nd-Edition/master/Chapter04/kant.txt --output "kant.txt"
```

로드하거나 다운로드한 후에는 코랩 파일 관리자 창에서 나타난다.

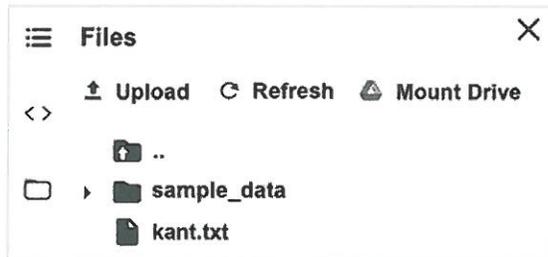


그림 4.1 코랩 파일 관리자

구글 코랩에서 가상 머신을 재시작하면 파일이 삭제되므로 유의해야 한다. 데이터셋이 로드된 것을 확인할 수 있다.

#### TIP

kant.txt 파일 없이 다음 셀들을 실행할 수 없다. 학습 데이터가 반드시 필요하다.

## 4.2.2 2단계: 허깅페이스 트랜스포머 설치하기

허깅페이스 트랜스포머와 토큰라이저를 설치해야 하지만, 이 구글 코랩 VM의 경우 텐서플로우(TensorFlow)는 필요하지 않다.

```
#@제목 2단계: 허깅페이스 트랜스포머 설치하기
# 이 단계에서 텐서플로우는 필요하지 않다
!pip uninstall -y tensorflow
# 마스터 브랜치에서 트랜스포머 설치하기
!pip install git+https://github.com/huggingface/transformers !pip list | grep
-E 'transformers|tokenizers'
# 트랜스포머 버전 --- 2.9.1
# 토큰라이저 버전 --- 0.7.0
```

설치가 완료되면 버전이 다음과 같이 출력될 것이다.

```
Successfully built transformers
tokenizers          0.7.0
transformers        2.10.0
```

트랜스포머 버전은 매우 빠르게 발전하고 있으므로, 실제 실행하는 버전은 달라질 수 있다. 이제 토큰라이저를 학습하는 프로그램을 실행해 보자.

## 4.2.3 3단계: 토큰라이저 학습하기

이 섹션에서는 사전 학습된 토큰라이저를 사용하지 않는다. 미리 학습된 GPT-2 토큰라이저를 사용할 수 있지만, 이 챕터에서는 처음부터 토큰라이저를 학습해 보겠다.

- 허깅페이스의 kant.txt를 사용하여 ByteLevelBPETokenizer()를 학습할 것이다. BPE 토큰라이저는 문자열이나 단어를 부분 문자열 또는 서브 워드로 분해한다. 이 방식은 두 가지 주요 장점을 가지고 있다.
- 토큰라이저는 단어를 최소 구성 요소로 분해할 수 있다. 그런 다음 이러한 작은 구성 요소를 통계적으로 흥미로운 구성 요소로 병합한다. 예를 들어 "smaller"와 "smallest"는 "small", "er", "est"가 될 수 있다. 토큰라이저는 이 구성 요소들을 더 나눌 수 있다. 예를 들어 "sm"과 "all"과 같은 구성 요

소들을 얻을 수 있다. 어떤 경우에도 단어는 서브워드 토큰과 "small" 대신 "sm"과 "all"과 같은 더 작은 단위로 분해된다.

- 워드피스(WordPiece) 인코딩을 사용하면 unk\_token(알 수 없는 토큰을 의미)으로 분류된 문자열은 사실상 사라진다.

이 모델에서는 다음 매개변수로 토큰라이저를 학습한다.

- files = paths는 데이터셋 경로이다.
- vocab\_size = 52\_000은 토큰라이저 모델의 크기다.
- min\_frequency = 2는 최소 빈도 임계값이다.
- special\_tokens=[]은 스페셜 토큰의 목록이다.

스페셜 토큰의 목록은 아래와 같다.

- <s>: 시작 토큰
- <pad>: 패딩 토큰
- </s>: 끝 토큰
- <unk>: 알 수 없는 토큰
- <mask>: 언어 모델링용 마스크 토큰

토큰라이저는 부분 문자열 토큰을 생성하고 빈도를 분석하도록 학습된다.

문장 중간에 있는 두 단어를 예로 들어보겠다.

```
...the tokenizer...
```

첫 단계는 이 문자열을 아래와 같이 분해하는 것이다.

```
'Gthe', 'Gtoken', 'izer'.
```

문자열은 이제 G(공백) 정보가 있는 토큰으로 토큰화된다.

다음 단계는 이를 인덱스로 변환하는 것이다.

'Gthe'	'Gtoken'	'izer'
150	5430	4712

표 4.1 세 가지 토큰과 인덱스

토큰라이저를 아래와 같이 실행시킬 수 있다.

```
#@title 3단계: 토큰라이저 학습하기
%%time
from pathlib import Path
from tokenizers import ByteLevelBPETokenizer
paths = [str(x) for x in Path(".").glob("**/*.txt")]
# 토큰라이저 초기화하기
tokenizer = ByteLevelBPETokenizer()
# 토큰라이저 학습하기

tokenizer.train(files=paths, vocab_size=52_000, min_frequency=2, special_
tokens=[
    "<s>",
    "<pad>",
    "</s>",
    "<unk>",
    "<mask>",
])
```

토큰라이저 학습 소요 시간은 아래와 같이 출력된다.

```
CPU times: user 14.8 s, sys: 14.2 s, total: 29 s
Wall time: 7.72 s
```

이제, 학습된 토큰라이저를 저장해 보겠다.

## 4.2.4 4단계: 디스크에 파일 저장하기

토큰나이저는 학습할 때 두 개의 파일을 생성한다.

- merges.txt에는 병합되고 토큰화된 부분 문자열이 들어있다
- vocab.json에는 토큰화된 부분 문자열의 인덱스가 들어있다.

아래의 코드를 사용해 KantaiBERT 디렉터리를 만든 다음 두 개의 파일을 저장한다.

**#@title 4단계: 파일 저장하기**

```
import os
token_dir = '/content/KantaiBERT'
if not os.path.exists(token_dir):
    os.makedirs(token_dir)
tokenizer.save_model('KantaiBERT')
```

파일이 저장되면서, 아래와 같이 출력될 것이다.

```
['KantaiBERT/vocab.json', 'KantaiBERT/merges.txt']
```

또한 아래와 같이 파일 관리자 창에서 확인할 수 있을 것이다.

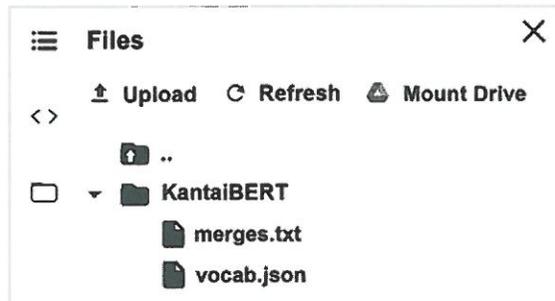


그림 4.2 코랩 파일 관리자

이 예제의 파일들은 사이즈가 작다. 두 번 클릭해 파일의 내용을 확인할 수 있다. 예상대로 merges.txt에는 토큰화된 부분 문자열이 들어있다.

```
#version: 0.2 - Trained by 'huggingface/tokenizers'  
Ġt  
he  
Ġa  
on  
in  
Ġo  
Ġt he  
re  
it  
Ġo f
```

vocab.json에는 인덱스가 들어있다.

```
[...,"Ġthink":955,"preme":956,"ĠE":957,"Ġout":958,"Ġdut":959,"aly":960,"Ġexp":961,...]
```

이제, 토큰화된 데이터 파일을 처리할 차례다.

#### 4.2.5 5단계: 학습된 토큰나이저 파일 로드하기

미리 학습된 토큰나이저 파일을 로드할 수도 있지만, 직접 토큰나이저를 학습했으며 이를 로드할 것이다.

```
#@title 5단계 학습된 토큰나이저 로드하기
```

```
from tokenizers.implementations import ByteLevelBPETokenizer  
from tokenizers.processors import BertProcessing  
tokenizer = ByteLevelBPETokenizer(  
    "/KantaiBERT/vocab.json",  
    "/KantaiBERT/merges.txt",  
)
```

토크나이저를 사용해 다음과 같이 문자열을 인코딩할 수 있다.

```
tokenizer.encode("The Critique of Pure Reason.")[0:6]
```

“The Critique of Pure Reason”은 다음과 같이 변환된다.

```
['The', 'ĠCritique', 'Ġof', 'ĠPure', 'ĠReason', '.']
```

다음과 같이 토큰의 수를 확인하는 것도 가능하다.

```
tokenizer.encode("The Critique of Pure Reason.")[0].num_tokens
```

출력 결과에서 시퀀스에 6개의 토큰이 있다는 것을 볼 수 있다.

```
Encoding(num_tokens=6, attributes=[ids, type_ids, tokens, offsets,
attention_mask, special_tokens_mask, overflowing])
```

이제 토크나이저는 KantaiBERT에 적합하게 토큰을 처리한다. 다음과 같이 시작 토큰과 끝 토큰을 넣어주는 후처리를 추가할 수 있다.

```
tokenizer.tokenizer_post_processor = BertProcessing(
    ("</s>", tokenizer.token_to_id("</s>")),
    ("<s>", tokenizer.token_to_id("<s>")),
)
tokenizer.enable_truncation(max_length=512)
```

후처리된 시퀀스를 인코딩해 보겠다.

```
tokenizer.encode("The Critique of Pure Reason.")[0:6]
```

출력 결과를 보면, 이제 8개의 토큰이 있다는 것을 확인할 수 있다.

```
Encoding(num_tokens=8, attributes=[ids, type_ids, tokens, offsets,
attention_mask, special_tokens_mask, overflowing])
```

추가된 토큰을 보고 싶다면, 다음 셀을 실행하여 토큰라이저가 후처리된 시퀀스를 인코딩하도록 할 수 있다.

```
tokenizer.encode("The Critique of Pure Reason.").tokens
```

출력 결과를 보면, 시작 토큰과 끝 토큰이 추가되어 총 8개의 토큰이 되었다는 것을 확인할 수 있다.

```
['<s>', 'The', 'ĠCritique', 'Ġof', 'ĠPure', 'ĠReason', '.', '</s>']
```

이제 모델 학습을 위한 데이터가 준비되었다. 이 노트북을 실행하는 시스템의 정보를 확인해 보자.

#### 4.2.6 6단계: 자원 제약 확인하기: GPU와 CUDA

KantaiBERT는 그래픽 처리 장치(GPU)에서 최적의 속도로 작동한다. 먼저 NVIDIA GPU 카드가 있는지 확인하기 위해 다음 명령을 실행한다.

```
#@title 6단계: 자원 제약 확인하기: GPU와 NVIDIA
!nvidia-smi
```

NVIDIA GPU 카드의 정보와 버전이 표시된다.

NVIDIA-SMI 440.82      Driver Version: 418.67      CUDA Version: 10.1									
GPU	Name	Persistence-M	Bus-Id	Disp.A	Volatile	Uncorr.	ECC		
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util	Compute	M.		
0	Tesla K80	Off	00000000:00:04.0	Off			0		
N/A	49C	P0	63W / 149W	9707MiB / 11441MiB	0%	Default			

Processes:							GPU Memory
GPU	PID	Type	Process name				Usage

그림 4.3 NVIDIA 카드 정보의 정보

출력 결과는 구글 코랩 VM의 구성에 따라 다를 수 있다. 이제 파이토치(PyTorch)가 CUDA(Compute Unified Device Architecture)를 인식하는지 확인하겠다

```
#@title 파이토치가 CUDA를 인식하는지 확인하기
import torch
torch.cuda.is_available()
```

결과로는 True가 출력되어야 한다.

```
True
```

CUDA는 NVIDIA에서 GPU의 병렬 컴퓨팅 파워를 사용하기 위해 개발한 기술이다. NVIDIA GPU와 CUDA에 대한 자세한 내용은 부록 II, 트랜스포머 모델의 하드웨어 제약사항을 참조하길 바란다. 이제 모델의 구성을 정의할 준비가 되었다.

### 4.2.7 7단계: 모델의 구성 정의하기

우리는 DistilBERT 트랜스포머와 동일한 수의 층과 헤드를 가진 RoBERTa 유형의 트랜스포머 모델을 사전 학습할 것이다. 모델의 사전 크기는 52,000이며, 모델은 12개의 어텐션 헤드와 6개의 층으로 구성되어 있다.

#### #@title 7단계: 모델의 설정 정의하기

```
from transformers import RobertaConfig
config = RobertaConfig(
    vocab_size=52_000,
    max_position_embeddings=514,
    num_attention_heads=12,
    num_hidden_layers=6,
    type_vocab_size=1,
)
```

이 구성에 대해서는 9단계: 모델 초기화하기에서 더 자세히 살펴보겠다. 우선 모델의 토큰라이저를 다시 만들어 보자.

### 4.2.8 8단계: 트랜스포머의 토큰라이저 다시 불러오기

이제 사전 학습된 토큰라이저를 `RobertaTokenizer.from_pretrained()`로 불러올 수 있다.

#### #@title 8단계: 트랜스포머의 토큰라이저를 재생성하기

```
from transformers import RobertaTokenizer
tokenizer = RobertaTokenizer.from_pretrained("./KantaiBERT", max_length=512)
```

이제 학습된 토큰라이저를 불러왔으니, RoBERTa 모델을 초기화해 보겠다.

### 4.2.9 9단계: 모델 초기화하기

이 섹션에서는 모델을 초기화하고 모델의 크기를 살펴보겠다. 먼저 언어 모델링을 위한 RoBERTa masked 모델을 가져오겠다.

#### #@title 9단계: 모델 초기화하기

```
from transformers import RobertaForMaskedLM
```

7단계에서 정의한 설정으로 모델을 초기화한다.

```
model = RobertaForMaskedLM(config=config)
```

모델을 출력하면 6개 층과 12개의 헤드를 가진 BERT 모델임을 확인할 수 있다.

```
print(model)
```

출력을 확인해 보면, 차원 값은 다르지만, 오리지널 트랜스포머의 인코더와 동일한 블록을 확인할 수 있다.

```
RobertaForMaskedLM(
  (roberta): RobertaModel(
    (embeddings): RobertaEmbeddings(
      (word_embeddings): Embedding(52000, 768, padding_idx=1)
      (position_embeddings): Embedding(514, 768, padding_idx=1)
      (token_type_embeddings): Embedding(1, 768)
      (LayerNorm): LayerNorm((768), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          (output): BertSelfOutput(
            (dense): Linear(in_features=768, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
  )
)
```

```

    )
    )
    (intermediate): BertIntermediate(
      (dense): Linear(in_features=768, out_features=3072, bias=True)
    )
    (output): BertOutput(
      (dense): Linear(in_features=3072, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
)
.../...

```

진행하기 전에, 모델 구성의 세부 정보를 자세히 살펴보는 시간을 가져보길 바란다. 모델의 내부를 더욱 잘 이해할 수 있을 것이다.

트랜스포머 모델을 레고 형태로 구성 요소를 분석하다 보면 서브 층 전반에 걸쳐 드롭아웃 정규화가 적용되어 있다는 것을 알 수 있을 것이다.

이제 파라미터를 탐색해 보겠다.

#### 4.2.9.1 파라미터 탐색하기

모델 사이즈는 작은 편이며, 84,095,008개 파라미터를 가지고 있다.

다음과 같이 모델 사이즈를 확인할 수 있다.

```
print(model.num_parameters())
```

대략적인 파라미터 수가 출력된다. 다만, 트랜스포머의 버전에 따라 완전히 똑같지는 않을 수 있다.

```
84095008
```

이제 파라미터를 살펴보겠다. 먼저 파라미터를 LP에 저장하고 파라미터 리스트의 길이를 계산해 보자.

```
#@title 파라미터 탐색하기
LP=list(model.parameters())
lp=len(LP)
print(lp)
```

출력 결과는 대략 108개의 행렬과 벡터가 있다는 것을 보여준다. 이는 트랜스포머 모델에 따라 달라질 수 있다.

```
108
```

이제 텐서(Tensor) 안에 있는 108개의 행렬과 벡터를 표시해 보겠다.

```
for p in range(0,lp):
    print(LP[p])
```

출력 결과는 다음과 같이 모든 파라미터를 보여준다.

```
Parameter containing:
tensor([[ -0.0175, -0.0210, -0.0334, ...,  0.0054, -0.0113,  0.0183],
        [  0.0020, -0.0354, -0.0221, ...,  0.0220, -0.0060, -0.0032],
        [  0.0001, -0.0002,  0.0036, ..., -0.0265, -0.0057, -0.0352],
        ...,
        [-0.0125, -0.0418,  0.0190, ..., -0.0069,  0.0175, -0.0308],
        [  0.0072, -0.0131,  0.0069, ...,  0.0002, -0.0234,  0.0042],
        [  0.0008,  0.0281,  0.0168, ..., -0.0113, -0.0075,  0.0014]],
       requires_grad=True)
```

파라미터를 들여다보고, 트랜스포머가 어떻게 구성되어 있는지 이해해 보자.

파라미터 수는 모델의 모든 파라미터를 합산하여 계산하게 된다. 예를 들어,

- 어휘 크기(52,000) × 차원(768)
- 벡터 크기 1 × 768
- 그 외, 다양한 차원의 파라미터들

여기서  $d_{\text{model}}=768$ 이라는 것을 알 수 있다. 모델에는 12개의 헤드가 있으므로,  $d_k$ 의 차원은  $d_k = d_{\text{model}}/12=64$ 일 것이다. 이것은 트랜스포머의 구성 요소를 레고를 조립하듯이 사용 수 있다는 것을 의미한다.

이제 모델 파라미터 수가 어떻게 계산되며, 84,095,008이라는 수치에 도달하는지 살펴보겠다. 노트북에서 LP 위에 마우스를 올리면, 토치 텐서(torch tensor)의 일부 모양을 볼 수 있다.

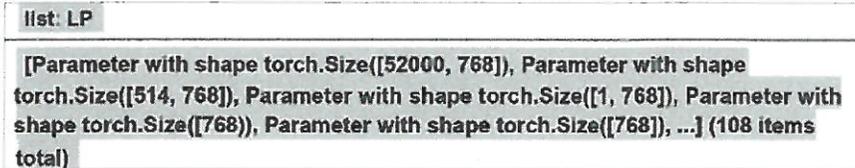


그림 4.4 LP

참고로, 사용하는 트랜스포머 모듈의 버전에 따라 숫자가 달라질 수 있다.

이제, 각 텐서의 매개변수 수를 세어볼 것이다. 먼저, 이 프로그램은 파라미터 카운터인 np(number of parameter)를 초기화하고 매개변수 리스트의 lp (108)개 요소를 순회할 것이다.

```
#@title 파라미터 수 세기
np=0
for p in range(0, lp): # 텐서 개수
```

파라미터는 다음과 같이 다양한 크기를 가지고 있다.

- $768 \times 768$
- $768 \times 1$
- 768

일부는 2차원, 그리고 일부는 1차원의 행렬과 벡터임을 알 수 있다.

리스트 LP의 원소가 이차원인지 아닌지는 다음과 같은 방법으로 확인할 수 있다.

```

PL2=True
try:
    L2=len(LP[p][0]) # 2차원인지 확인하기
except:
    L2=1 #2차원이 아니라면 1차원임을 의미
    PL2=False

```

파라미터가 2차원이면, 두 번째 차원은  $L2 > 0$ 이고  $PL2=True$ (2 dimensions=True)이다.

파라미터가 하나의 차원만 있으면, 두 번째 차원은  $L2=1$ 이고  $PL2=False$ (2 dimensions=False)이다.

$L1$ 은 매개변수의 첫 번째 차원의 크기다.  $L3$ 는 파라미터의 크기를 나타내며 다음과 같이 정의된다.

```

L1=len(LP[p])
L3=L1*L2

```

이제 루프의 각 단계에서 매개변수를 더할 수 있다.

```

np+=L3 # 텐서 당 파라미터 수

```

매개변수의 합을 구할 것이지만, 동시에 트랜스포머 모델의 매개변수 수가 어떻게 계산되는지도 알아 볼 것이다.

```

if PL2==True:
    print(p,L1,L2,L3) # 파라미터 크기 표시하기
if PL2==False:
    print(p,L1,L3) # 파라미터 크기 표시하기
print(np) # 총 파라미터 수

```

참고로, 매개변수가 하나의 차원만 있는 경우  $PL2=False$ 이므로 첫 번째 차원만 표시된다.

다음과 같이 모델의 모든 텐서에 대해 매개변수 수가 어떻게 계산되었는지 확인할 수 있다.

```
0 52000 768 39936000
1 514 768 394752
2 1 768 768
3 768 768
4 768 768
5 768 768 589824
6 768 763
7 768 768 589824
8 768 768
9 768 768 589824
10 768 768
```

RoBERTa 모델의 총 매개변수 수는 목록의 끝에 표시된다.

```
84,095,008
```

라이브러리의 버전에 따라 매개변수 수는 다를 수 있다.

이제 트랜스포머 모델에서 파라미터의 수가 정확히 무엇을 의미하는지 알게 되었다. 출력물의 구성, 매개변수 내용 및 매개변수 크기를 다시 살펴보고 몇 분 동안 이를 이해해 보기를 바란다. 모델의 구성 요소를 정확하게 이해하게 될 것이다.

이제 데이터셋을 구축하기 위한 프로그램을 실행해 보겠다.

## 4.2.10 10단계: 데이터 구축

이제 프로그램은 `block_size=128`로 예제 길이를 제한하여 배치 학습을 위한 샘플을 생성하기 위해 데이터셋을 한 줄씩 로드한다.

```
#@title 10단계: 데이터셋 구축하기
%%time
from transformers import LineByLineTextDataset
dataset = LineByLineTextDataset(
    tokenizer=tokenizer,
```

```

    file_path="/kant.txt",
    block_size=128,
)

```

출력 결과는 허깅페이스가 데이터 처리 시간을 최적화하는 데 상당한 리소스(resource)를 투자했음을 보여준다.

```

CPU times: user 8.48 s, sys: 234 ms, total: 8.71 s
Wall time: 3.88 s

```

프로세서가 실제로 활동한 시간(월 타임, Wall time)이 최적화되어 있음을 확인할 수 있다.

프로그램은 이제 역전파(backpropagation)를 위한 객체를 생성하기 위해 데이터 콜레이터(data collator)를 정의한다.

#### 4.2.11 11단계: 데이터 콜레이터 정의

트레이너를 초기화하기 전에 데이터 콜레이터(data collator)를 실행해야 한다. 데이터 콜레이터는 데이터셋에서 샘플을 가져와 배치(batch)로 정리한다. 사전과 유사한 객체를 결과로 얻게 된다.

MLM을 위한 배치 샘플 프로세스를 준비하기 위해 `mlm=True`로 설정한다.

사전 학습 과정에서 마스킹된 토큰의 비율을 결정하는 파라미터 `mlm_probability`는 0.15로 설정한다.

이제 마스킹된 언어 모델링을 활성화하고 마스킹된 토큰 비율을 0.15로 설정하고 토큰라이저를 주입해 데이터 콜레이터(data collator)를 초기화한다.

```

#@title 11단계: 데이터 콜레이터 정의하기
from transformers import DataCollatorForLanguageModeling
data_collator = DataCollatorForLanguageModeling(
    tokenizer=tokenizer, mlm=True, mlm_probability=0.15
)

```

이제, 트레이너를 초기화할 준비가 되었다.

## 4.2.12 12단계: 트레이너 초기화

이전 단계에서는 트레이너를 초기화하는 데 필요한 정보를 준비했다. 데이터셋은 토큰화되어 로드되었고, 모델을 구축했으며 데이터 콜레이터 또한 준비했다.

이제 트레이너를 초기화할 차례다. 교육 목적을 위해 에포크 수를 한 개로 설정하여 모델 학습이 빠르게 진행되도록 하였다. GPU를 사용하면 배치를 공유하고 모델 학습을 멀티프로세싱으로 수행할 수 있기 때문에 유용하다.

```
#@title 12단계: 트레이너 초기화하기
from transformers import Trainer, TrainingArguments
training_args = TrainingArguments(
    output_dir="/KantaiBERT",
    overwrite_output_dir=True,
    num_train_epochs=1,
    per_device_train_batch_size=64,
    save_steps=10_000,
    save_total_limit=2,
)
trainer = Trainer(
    model=model,
    args=training_args,
    data_collator=data_collator,
    train_dataset=dataset,
)
```

이제 모델을 학습시킬 준비가 되었다.

## 4.2.13 13단계: 모델 사전 학습시키기

모든 것이 준비되었다. 다음 한 줄을 실행시켜 모델 학습을 시작할 수 있다.

```
#@title 13단계: 모델 사전 학습하기
%%time
trainer.Train()
```

출력은 실시간으로 학습 과정을 보여주며, 손실(loss), 학습률(learning rate), 에포크(epoch), 단계(steps) 등이 표시된다.

```
Epoch: 100%
1/1 [17:59<00:00, 1079.91s/it]
Iteration: 100%
2672/2672 [17:59<00:00, 2.47it/s]
{"loss": 5.6455852394104005, "learning_rate": 4.06437125748503e-05,
 "epoch": 0.18712574850299402, "step": 500}
{"loss": 4.940259679794312, "learning_rate": 3.12874251497006e-05,
 "epoch": 0.37425149700598803, "step": 1000}
{"loss": 4.639936000347137, "learning_rate": 2.1931137724550898e-05,
 "epoch": 0.561377245508982, "step": 1500}
{"loss": 4.361462069988251, "learning_rate": 1.2574850299401197e-05,
 "epoch": 0.7485029940119761, "step": 2000}
{"loss": 4.228510192394257, "learning_rate": 3.218562874251497e-06,
 "epoch": 0.9356287425149701, "step": 2500}
CPU times: user 11min 36s, sys: 6min 25s, total: 18min 2s
Wall time: 17min 59s
TrainOutput(global_step=2672, training_loss=4.7226536670130885)
```

모델 학습이 완료되었다. 이제 작업물을 저장할 시간이다.

#### 4.2.14 14단계: 최종 모델(토크나이저 및 설정 파일) 저장하기

이제 모델과 설정 파일을 저장할 것이다.

```
#@title 14단계: 최종 모델(토크나이저 및 설정 파일) 저장하기
trainer.save_model("./KantaiBERT")
```

파일 관리자에서 새로 고침(Refresh)을 클릭하면 파일이 나타날 것이다.

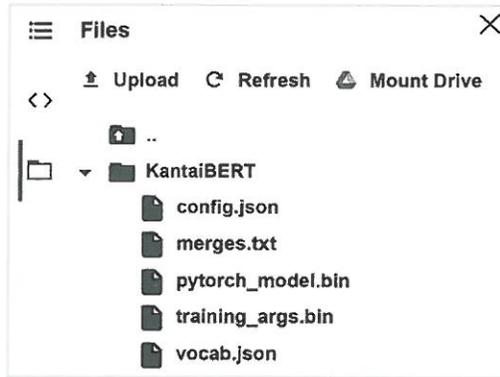


그림 4.5 코랩 파일 관리자

이제 파일 관리자에 config.json, pytorch\_model.bin, training\_args.bin 파일이 나타날 것이다. merges.txt와 vocab.json에는 사전 학습된 토큰나이저로 토큰화한 데이터가 들어있다.

우리는 처음부터 모델을 구축했다. 이제 사전 학습한 모델과 토큰나이저를 사용하여 언어 모델링 작업을 수행하는 파이프라인(pipeline)을 가져와 보겠다.

#### 4.2.15 15단계: FillMaskPipeline을 사용한 언어 모델링

이제 언어 모델링 fill-mask 작업을 가져올 것이다. 학습된 모델과 학습한 토큰나이저를 사용하여 마스크드 언어 모델링 작업을 수행할 것이다.

```
#@title 15단계: FillMaskPipeline을 사용한 언어 모델링
from transformers import pipeline
fill_mask = pipeline(
    "fill-mask",
    model="/KantaiBERT",
    tokenizer="/KantaiBERT"
)
```

모델에 임마누엘 칸트처럼 생각하게 요청해 보자.

```
fill_mask("Human thinking involves human <mask>.")
```

한정된 데이터로 모델을 처음부터 사전 학습하기 때문에 각 실행마다 출력이 달라질 수 있다. 결과는 다를 수 있지만, 이번 실행에서의 출력으로 언어 모델링 개념을 이해할 수 있을 것이다.

```
[{'score': 0.022831793874502182,
  'sequence': '\s Human thinking involves human reason.</s>',
  'token': 393},
 {'score': 0.011635891161859035,
  'sequence': '\s Human thinking involves human object.</s>',
  'token': 394},
 {'score': 0.010641072876751423,
  'sequence': '\s Human thinking involves human priori.</s>',
  'token': 575},
 {'score': 0.009517930448055267,
  'sequence': '\s Human thinking involves human conception.</s>',
  'token': 418},
 {'score': 0.00923212617635727,
  'sequence': '\s Human thinking involves human experience.</s>',
  'token': 531}]
```

예측 결과는 실행할 때마다 또는 히킹페이스가 모델을 업데이트할 때마다 달라질 수 있다. 그러나 다음 출력은 쉽게 확인할 수 있을 것이다.

```
Human thinking involves human reason
```

이번 장의 목표는 트랜스포머 모델을 학습하는 방법을 살펴보는 것이었다. 우리는 트랜스포머 모델이 인간과 유사한 흥미로운 예측이 가능함을 확인할 수 있다.

결과는 실험 수준이며 학습 과정에 따라 달라질 수 있다. 모델을 다시 학습할 때마다 결과는 달라질 것이다.

모델을 제대로 학습시키기 위해서는 다른 계몽시대 인문학자들의 데이터가 훨씬 더 많이 필요할 것이다. 하지만 이번 장을 통해 고성능 트랜스포머 학습을 위한 데이터셋을 생성하는 방법을 배울 수 있었다.

트랜스포머 덕분에 우리는 인공지능의 새로운 시대를 막 시작한 것이다!

## 4.3

## 다음 단계

이번 장에서는 트랜스포머를 처음부터 학습시켰다. 개인적으로 또는 기업에서 할 수 있는 것에 대해 생각해 보자. 특정 작업을 위한 데이터셋을 만들고 처음부터 학습시킬 수 있을 것이다. 여러분의 관심 분야나 회사 프로젝트를 활용하여 흥미로운 트랜스포머 구축 키트를 실험해 볼 수 있다.

원하는 모델을 만들었다면, 허깅페이스 커뮤니티와 공유할 수 있다. 공유한 모델은 허깅페이스 모델 페이지에 표시될 것이다.( <https://huggingface.co/models>)

다음 페이지에 설명된 지침을 사용하여 몇 단계만 거치면 모델을 업로드할 수 있다.([https://huggingface.co/transformers/model\\_sharing.html](https://huggingface.co/transformers/model_sharing.html))

또한 허깅페이스 커뮤니티가 공유한 모델을 다운로드하여 개인 및 전문 프로젝트에 대한 새로운 아이디어를 얻을 수 있다.

이번 장에서는 허깅페이스에서 제공하는 구성 요소를 사용하여 RoBERTa와 유사한 모델인 KantaiBERT를 처음부터 구축했다.

먼저, 임마누엘 칸트의 작업과 관련된 커스텀 데이터셋을 로드하는 것으로 시작했다. 목표에 따라 기존 데이터셋을 로드하거나 직접 생성할 수 있다. 커스텀 데이터셋을 사용하면 트랜스포머 모델의 작동 방식에 대한 인사이트를 얻을 수 있음을 확인했다. 그러나 이 실험에는 한계가 있었다. 교육 목적 이상으로 모델을 학습하려면 훨씬 큰 데이터셋이 필요할 것이다.

KantaiBERT 프로젝트에서 kant.txt의 데이터셋에 대한 토큰라이저를 학습할 수 있었다. 학습한 merges.txt와 vocab.json 파일이 저장되었음을 확인할 수 있었으며, 사전 학습된 파일로 토큰라이저를 만들 수 있었다. 또한, 커스텀 데이터셋을 구축하고 역전파(backpropagation)를 위해 학습 배치를 처리하는 데이터 콜레이터를 정의했다. 더 나아가, 트레이너를 초기화하고 RoBERTa 모델의 파라미터를 자세히 살펴보았다. 또한, 모델을 학습시키고 저장까지 해보았다.

마지막으로 저장된 모델을 다운스트림 언어 모델링 작업에 사용했다. 이 모델을 사용해 임마누엘 칸트의 논리를 채우는 작업을 해보았다.

더 나아가 기존 데이터셋이나 새로운 커스텀 데이터셋을 실험하여 어떤 결과를 얻을 수 있는지 확인해 볼 수 있다. 모델을 허깅페이스 커뮤니티에 공유할 수도 있다. 트랜스포머의 성능은 데이터에 기반한다. 다양한 데이터로 트랜스포머를 사용하는 새로운 방법을 발견할 수도 있다.

사전 학습이나 미세 조정없이 API 만으로 트랜스포머 엔진을 실행하는 방법을 배우게 될 것이다. 7장, **GPT-3 엔진을 사용한 초인간 트랜스포머 등장**에서는 인공지능의 미래를 엿볼 수 있다. 이번 장과 이전 장의 지식이면 충분하다.

다음 장, **트랜스포머를 사용한 다운스트림 NLP 작업**에서는 트랜스포머를 구현하기 위한 준비들을 계속 할 것이다.

## 4.5 문제

01. RoBERTa 는 바이트 단위 바이트 쌍 인코딩 토큰라이저를 사용한다. 참 / 거짓
02. 학습된 허깅페이스 토큰라이저는 merges.txt와 vocab.json을 생성한다. 참 / 거짓
03. RoBERTa는 토큰 타입 ID를 사용하지 않는다. 참 / 거짓
04. DistilBERT 는 6개 층과 12개 헤드를 가진다. 참 / 거짓
05. 8천만 개 파라미터를 가진 트랜스포머 모델은 거대한 층에 속한다. 참 / 거짓
06. 토큰라이저는 학습할 수 없다. 참 / 거짓
07. BERT 류(BERT-like)의 모델은 6개의 디코더 층을 가진다. 참 / 거짓
08. 마스크드 언어 모델(Masked Language Model, MLM)은 문장 내의 마스크된 토큰을 예측한다. 참 / 거짓
09. BERT 류의 모델은 셀프-어텐션 서브 층을 가지고 있지 않다. 참 / 거짓
10. 데이터 콜레이터는 역전파에 도움이 된다. 참 / 거짓

## 4.6 참고 문헌

- Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyano, 2019, RoBERTa: A Robustly Optimized BERT Pretraining Approach: <https://arxiv.org/abs/1907.11692>
- 허깅페이스 토큰라이저: [https://huggingface.co/transformers/main\\_classes/tokenizer.html?highlight=tokenizer](https://huggingface.co/transformers/main_classes/tokenizer.html?highlight=tokenizer)
- 허깅페이스 관련 노트북: [https://colab.research.google.com/github/huggingface/blog/blob/master/notebooks/01\\_how\\_to\\_train.ipynb](https://colab.research.google.com/github/huggingface/blog/blob/master/notebooks/01_how_to_train.ipynb)
- 허깅페이스 블로그: <https://huggingface.co/blog>
- BERT에 대해 자세히 알아보기: [https://huggingface.co/transformers/model\\_doc/bert.html](https://huggingface.co/transformers/model_doc/bert.html)
- DistilBERT 관련 논문: <https://arxiv.org/pdf/1910.01108.pdf>
- RoBERTa에 대해 자세히 알아보기: [https://huggingface.co/transformers/model\\_doc/roberta.html](https://huggingface.co/transformers/model_doc/roberta.html)
- DistilBERT에 대해 자세히 알아보기: [https://huggingface.co/transformers/model\\_doc/distilbert.html](https://huggingface.co/transformers/model_doc/distilbert.html)

# 05장

---

## 트랜스포머를 사용한 다운스트림 NLP 작업

---

- 5.1 트랜스포머의 트랜스덕션과 귀납적 상속
- 5.2 트랜스포머 성능 vs 인간 기준값
- 5.3 다운스트림 작업 실행하기
- 5.4 정리하기
- 5.5 문제
- 5.6 참고 문헌

트랜스포머는 사전 학습된 모델을 활용해서 자연어 이해(Natural Language Understanding, NLU)와 관련된 다운스트림 작업(downstream task)을 수행할 때 잠재력을 최대한 발휘한다. 트랜스포머 모델의 사전 학습과 미세 조정에는 많은 시간과 노력이 필요하지만, 수백만 파라미터의 트랜스포머 모델이 다양한 NLU 작업을 수행하는 것을 보면 그럴 가치는 충분하다.

이번 장에서 먼저 인간 기준값(human baseline)을 뛰어넘기 위한 노력을 소개한다. 인간 기준값은 NLU 작업에 대한 인간의 수행 능력을 나타낸다. 인간은 어릴 때부터 트랜스덕션을 배우고 빠르게 귀납적 사고를 발전시킨다. 인간은 감각을 통해 직접적으로 세상을 인식하는 반면, 인공 지능은 우리의 언어를 이해하기 위해, 단어로 표현한 우리의 지각능력에 전적으로 의존한다.

다음으로는 트랜스포머의 성능을 측정하는 방법을 살펴볼 것이다. 다양한 자연어 처리(Natural Language Processing, NLP) 작업의 정확도를 측정하면 모델의 성능을 직접적으로 확인할 수 있다. 성능 측정에는 다양한 벤치마크와 데이터셋이 사용된다. 일례로 SuperGLUE는 구글 딥마인드, 페이스북, 뉴욕 대학교, 워싱턴 대학교 등이 협력하여 NLP 성능 측정을 위한 높은 기준을 마련한 사례이다.

마지막으로, SST-2(Standard Sentiment TreeBank), 언어 적합성(linguistic acceptability), 위노그라드 스키마(Winograd schemas) 등 여러 다운스트림 작업을 살펴볼 것이다.

트랜스포머는 잘 설계된 벤치마크 작업에서 여러 모델을 능가하며 NLP를 한 단계 더 빠르게 발전시키고 있다. 트랜스포머 아키텍처를 활용한 새로운 모델은 계속해서 등장하고 진화할 것이다.

이번 장은 다음 주제들을 다룬다.

- 트랜스덕션과 귀납적 추론에 대한 기계와 인간의 비교
- NLP에서의 트랜스덕션과 귀납적 추론
- 인간 기준값 대비 트랜스포머 성능 측정
- 측정 방법(정확도, F1-점수, MCC)
- 벤치마크 작업과 데이터셋
- SuperGLUE 다운스트림 작업
- 언어 적합성과 CoLA
- 감성 분석과 SST-2
- 위노그라드 스키마

먼저 인간과 기계가 언어를 표현하는 방식을 이해해보자.

## 5.1

# 트랜스포머의 트랜스덕션과 귀납적 상속

모든 AI 전문가의 역할은 클라우드 AI 자동화 플랫폼의 API인 **AutoML(Automated Machine Learning)**의 등장으로 변화했다. 구글 버텍스(Google Vertex)는 머신러닝 구현을 위한 개발 시간을 80% 단축했다. 이는 누구나 준비된 시스템으로 머신러닝을 구현할 수 있음을 시사한다. 이것이 개발 인력의 80%가 감소함을 의미할까? 그렇지 않다. 4차 산업혁명의 AI 전문가는 AI를 조립하며 계속해서 프로젝트에 가치를 더할 것이다.

### NOTE

4차 산업혁명의 NLP AI 전문가는 소스 코드에 점점 적게 투자하고 팀의 AI 전문가로서 지식에 더 투자한다.

트랜스포머는 배우지 않은 업무에도 지식을 적용할 수 있는 특별한 능력이 있다. 예를 들어, BERT (Bidirectional Encoder Representations from Transformers)는 시퀀스-투-시퀀스(sequence-to-sequence)와 마스크드 언어 모델링으로 언어를 배우고나서, 처음에 학습하지 않았던 다운스트림 작업을 위해 미세 조정된다.

이번 섹션에서는 사고 실험을 해보려 한다. 먼저 트랜스포머의 그래프로 인간과 기계가 언어를 사용해 정보를 이해하는 방법을 표현하겠다. 기계는 인간과 다른 방식으로 정보를 이해하지만 매우 효율적인 결과를 얻는다.

그림 5.1은 트랜스포머 아키텍처와 서브 층으로 구성된 사고 실험으로, 인간과 기계 사이의 그럴듯한 유사성을 보여준다. 트랜스포머가 다운스트림 작업을 이해하기 위해 학습하는 과정을 알아보자.

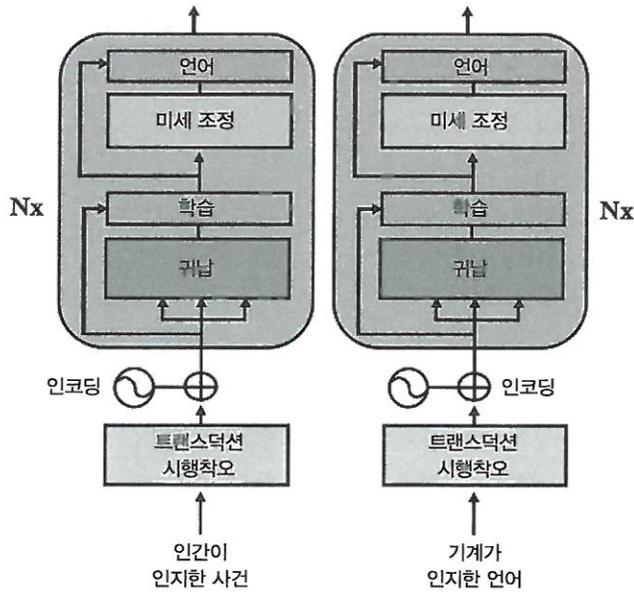


그림 5.1 인간과 기계의 방식

이 예제에선  $N=2$ 라고 가정하겠다. 두 층은 인간이 세대를 거쳐 축적된 지식을 활용함을 보여준다. 기계는 우리가 제공한 것만 처리한다. 우리의 출력을 입력으로 사용하는 것이다.

### 5.1.1 인간 지능 스택

그림 5.1의 왼쪽에서, 인간의 입력, 0번째 층은 사건의 인지이고, 최종 출력은 언어임을 볼 수 있다. 우리는 어렸을 때 우리의 감각으로 사건을 인식한다. 출력은 웅얼거림에서 구조적인 언어로 점차 변화한다.

인간의 트랜스덕션은 시행착오를 거친다. 트랜스덕션은 우리가 인지하는 구조를 패턴 등으로 표현하는 것을 말한다. 우리는 세상에 대한 표현을 만들고 우리의 귀납적 사고에 적용한다. 즉, 귀납적 사고는 트랜스덕션의 품질에 좌우된다.

예를 들어, 어린 시절에는 보통 오후 일찍 낮잠을 자도록 강요받는다. 저명한 아동심리학자 피아제(Piaget)는 어떤 아이들이 “낮잠을 안 잤으니까, 오후가 아니야”라고 말하는 것을 보았다. 이 어린이는 트랜스덕션으로 두 사건의 연결고리를 만들었으며 귀납적으로 사고하여 일반화하였다.

인간은 먼저 트랜스덕션으로 패턴을 발견하고 귀납적 추론으로 일반화한다. 그리고 나서, 시행착오를 통해 여러 사건들이 관련 있음을 배운다.

학습된 관련 사건들 = {일출-빛, 일몰-어둠, 어두운 구름-비, 푸른 하늘-달리기, 음식-좋음, 불-따뜻함, 눈-추움}

시간이 지나며 우리는 수백만 개의 관련 사건들을 학습하며 이해한다. 이때 새로운 세대의 인간은, 이전 세대가 이미 많은 것에 대해 미세 조정해 주었기 때문에, 처음부터 새로 배울 필요가 없다. “불은 뜨겁다” 등을 가르친 것이다. 그 이후, 아이들의 지식은 어떤 형태의 “불”로도 미세 조정될 수 있다. 예를 들면 양초, 산불, 화산 등 모든 “불”의 형태로 지식을 미세 조정할 수 있다.

인간은 마침내 알아내고, 상상하고, 예측한 모든 것을 언어로 표현하면서 0번 층의 출력값이 탄생한 것이다.

인간에게 있어서, 다음 층인 1번째 층의 입력은 학습되고 미세 조정된 방대한 지식이다. 이것에 더해, 인간은 막대한 양의 새로운 사건을 인식하고 기존 지식과 함께 트랜스덕션, 귀납 추론, 학습, 서브 층 미세 조정 과정을 수행한다.

이런 사건들의 상당수는 냄새, 감정, 상황, 경험 등 인간을 특별하게 하는 모든 것에서 비롯된다. 기계는 이런 개별 정체성이 없는 반면에 인간은 개인마다 특화된 방식으로 단어를 인식한다.

기계는 우리가 제공하는 다양한 대량의 데이터를 필터링되거나 특화되지 않은 상태로 받는다. 기계의 목표는 객관적이고 효율적인 작업 수행인 반면, 인간의 목표는 개인적인 행복이다.

인간은 0번 층에서 1번 층으로 이동하고, 더 많은 원시 데이터와 가공된 데이터와 함께, 다시 0번 층으로 되돌아가는 과정을 무한히 반복한다.

결과는 환상적이다! 우리는 요약 능력을 얻기 위해 언어를 처음부터 다시 배우지(학습하지) 않는다. 미리 학습된 지식을 요약 작업에 적용(미세 조정)하면 된다.

트랜스포머도 동일한 과정을 거치지만 그 방식은 다르다.

## 5.1.2 기계 지능 스택

기계는 앞에서 설명한 것과 같이 시퀀스 예측 방식 등을 배우고, 학습에 사용한 시퀀스를 활용해 수백 가지의 작업을 수행한다.

그림 5.1의 오른쪽에서, 기계의 입력이 언어 형태의 간접 정보임을 알 수 있다. 기계가 언어 분석에 사용하는 유일한 입력은 우리 인간의 출력이다.

현재까지의 기계와 인간의 역사에서, 컴퓨터 비전은 이미지를 식별하지만 언어의 문법 구조를 알진 못한다. 음성 인식은 소리를 단어로 바꿀 뿐이다. 음악 패턴 인식은 단어로 표현된 객관적인 개념을 알지 못한다.<sup>10</sup>

기계는 핸디캡을 가지고 시작한다. 인공적인 불이익을 강요받는 것이다. 기계는 고르지 못한 품질의 인간 출력에 의존하여 다음을 수행한다.

- 언어 시퀀스에서 동시에 등장하는 모든 토큰을 연결하며 트랜스덕션 수행
- 트랜스덕션 결과로부터 귀납적 추론 생성
- 귀납적 추론을 학습하여 토큰의 패턴 탐색

이때 유효한 귀납적 추론을 얻기 위해 노력 중인 어텐션 서브 층을 들여다보자.

- 트랜스포머 모델은 이전의 순환 기반(recurrent-based) 연산 대신 셀프 어텐션(self-attention)을 사용하여 모델의 시야를 넓혔다.
- 이때 어텐션 서브 층은 인간보다 우위에 있다. 귀납적 사고를 위해 수백만 개의 예제를 살펴볼 수 있다.
- 마치 인간처럼, 어텐션 서브 층도 트랜스덕션과 귀납적 추론으로 시퀀스에서 패턴을 찾는다.
- 이렇게 찾은 패턴은 모델에 파라미터로 저장된다.

기계는 대량의 데이터, 뛰어난 NLP 트랜스포머, 컴퓨터 성능이라는 그들만의 능력으로 언어를 이해했다. 언어를 깊게 이해한 덕분에, 기계는 학습하지 않은 수백 가지의 작업들을 수행할 수 있다.

트랜스포머도 인간처럼 몇 가지 과정으로 언어를 이해한다. 트랜스덕션으로 연결 관계를 찾고 귀납적 추론 과정으로 이를 일반화한다.

모델이 기계 지능을 위해 서브 층을 미세 조정 할 때면 마치 인간처럼 반응한다. 새로운 작업을 처음부터 학습하지 않고, 미세 조정만 필요한 다운스트림 작업으로 간주하는 것이다. 만약 질문에 답을 하는 방법을 학습한다면, 기계는 언어 자체를 처음부터 배우지 않는다. 트랜스포머 모델은 마치 우리처럼 파라미터를 미세 조정할 뿐이다.

10 역주. 음성 인식(speech recognition)은 사람의 음성을 입력받아 언어적 특성과 음성 신호의 시간적 특성, 언어적 특성, 발음 및 억양 등을 기반으로 음성 신호로부터 말의 내용(단어)을 인식하고 해석하는 기술이다. 응용 분야로는 음성 검색, 음성 명령 및 제어, 텍스트 변환(STT, Speech-To-Text) 등이 있다.

음악 패턴 인식(music pattern recognition)은 음악 신호를 입력 받아 음악의 주파수 특성, 시간적 패턴 및 다양한 음악적 특성을 기반으로 음악의 구조, 리듬, 화음(chord) 등을 인식한다. 응용 분야로는 음악 검색, 음악 장르 분류, 음악 추천 시스템 등이 있다.

이번 챕터에서는, 트랜스포머 모델이 우리처럼 학습하기 위해 고군분투함을 보았다. 기계가 인간의 언어 표현에 의존한다는 점부터 핸디캡이다. 하지만 기계는 엄청난 컴퓨팅 파워로 인간대비 무한에 가까운 데이터를 활용한다.

이제 트랜스포머 성능을 인간 기준값과 어떻게 비교하는지 살펴보겠다.

## 5.2

# 트랜스포머 성능 vs 인간 기준값

인간과 마찬가지로 트랜스포머는 사전 학습된 모델을 상속하여 다운스트림 작업을 위해 미세 조정된다. 사전 학습 모델은 파라미터에 언어 표현을 담고있다.

사전 학습 모델은 언어에 대한 일반적 지식을 얻기 위한 핵심 작업을 학습한다. 반면 모델을 미세 조정할때 다운스트림 작업을 학습한다. 모든 트랜스포머 모델이 동일한 작업으로 사전 학습하진 않는다. 잠재적으로는, 어떤 작업으로도 사전 학습이나 미세 조정을 할 수 있다.

모든 NLP 모델은 표준화된 방법으로 평가해야 한다. 이번 섹션에서는 먼저 일부 주요 측정 방법을 살펴보고, 주요 벤치마크 작업과 데이터셋을 살펴보겠다.

먼저 주요 측정 방식을 살펴보자.

### 5.2.1 모델을 평가하는 지표

어떤 트랜스포머 모델을 다른 트랜스포머 모델(또는 다른 NLP 모델)과 비교하려면 지표를 사용하는 보편적인 평가 체계가 필요하다.

이 섹션에서는 GLUE와 SuperGLUE에서 사용하는 세 가지 측정 방식을 소개하겠다.

#### 5.2.1.1 정확도

정확도는 어떻게 활용하더라도 가장 실용적인 평가 방법이다. 정확도 함수는 단순히 각 결과가 참인지 또는 거짓인지 계산한다. 일부 샘플  $samples_i$ 에 대해서 모델의 출력  $\hat{y}_i$ 이 정답  $y$ 와 일치하는지 판단하며, 수식으로 표현하면 다음과 같다.

$$Accuracy(y, \hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} 1(\hat{y}_i = y_i)$$

결과가 올바르면 1이 되고, 틀리면 0이 된다. 이제 더 유연한 방법인 F1score도 살펴보자.

### 5.2.1.2 F1score

F1score는 클래스 분포가 불균형한 데이터셋에서도 사용할 수 있는 더 유연한 방식이다.

F1score는 정밀도(precision)와 재현율(recall)의 가중 평균으로, 식으로 표현하면 다음과 같다.

$$\text{F1score} = 2 * (\text{precision} * \text{recall}) / (\text{precision} + \text{recall})$$

앞선 식의 정밀도(P)와 재현율(R)을 참 양성( $T_p$ ), 거짓 양성( $F_p$ ), 거짓 음성( $F_n$ )으로 표현하면 다음과 같다.

$$P = \frac{T_p}{T_p + F_p}$$

$$R = \frac{T_p}{T_p + F_n}$$

F1score는 정밀도(P)와 재현율(R)의 조화 평균(역수의 평균의 역수)으로도 나타낼 수 있다.

$$\text{F1score} = 2 * \frac{P * R}{P + R}$$

이제 MCC를 살펴보겠다.

### 5.2.1.3 매튜 상관 계수

매튜 상관 계수(Matthews Correlation Coefficient, MCC)는 3장, BERT 모델 미세 조정하기의 매튜 상관 계수로 평가하기에서 설명 및 구현하였다. MCC는 참 양성( $T_p$ ), 참 음성( $T_n$ ), 거짓 양성( $F_p$ ), 거짓 음성( $F_n$ )으로 지표를 측정한다.

MCC를 수식으로 표현하면 다음과 같다.

$$\text{MCC} = \frac{T_p * T_n - F_p * F_n}{\sqrt{(T_p + F_p)(T_p + F_n)(T_n + F_p)(T_n + F_n)}}$$

MCC는 이진 분류 모델 평가 지표로 두 클래스 크기가 불균형한 경우에도 유용하다.

지금까지 트랜스포머 모델의 결과를 어떻게 평가하고 다른 트랜스포머 모델이나 NLP 모델과 비교할지에 대한 좋은 방법들을 살펴봤다.

측정 방법을 기억하면서, 벤치마크 작업과 데이터셋을 살펴보자.

## 5.2.2 벤치마크 작업과 데이터셋

트랜스포머가 최고 수준의 성능임을 증명하기 위해선 세 가지가 준비되어야 한다.

- 모델
- 데이터셋에 기반한 작업
- 모델을 평가하는 지표 섹션에서 설명한 측정 방법

SuperGLUE 벤치마크를 시작으로 트랜스포머 모델의 평가 과정을 설명하겠다.

### 5.2.2.1 GLUE에서 SuperGLUE까지

SuperGLUE는 2019년 왕(Wang et al.)이 설계하고 공개한 벤치마크다. 왕은 먼저 **일반 언어 이해 평가 (General Language Understanding Evaluation, GLUE)** 벤치마크를 공개했다.

GLUE는 NLU가 유용하기 위해선 다양한 작업에 적용 가능해야 함을 보이기 위해 설계되었다. 상대적으로 작은 GLUE 데이터셋은 NLU 모델이 다양한 작업을 목표하도록 유도했다.

하지만 트랜스포머의 등장으로 NLU 모델의 성능은 GLUE 순위표(2021년 12월)에서 인간의 평균 수준을 뛰어넘었다.<sup>11</sup> GLUE 순위표는 <https://gluebenchmark.com/leaderboard>에서 확인할 수 있다. 혁신적인 트랜스포머 모델이 뛰어난 NLU 역량을 보여주고 있으며, RNN/CNN관련 아이디어도 일부 볼 수 있다.

다음은 순위표의 상위 순위와 GLUE 인간 기준값의 순위이다.

11 역주. 이후 마이크로소프트가 2022년 10월에 Turing ULR v6 모델을 발표하여 최고점수를 갱신하였으며, 현재(2023년)까지 순위를 유지하고 있다. 이 모델 역시 트랜스포머를 기반으로 한다.

Rank	Name	Model	URL	Score
1	Microsoft Alexander v-team	Turing NLR v5		91.2
2	ERNIE Team - Baidu	ERNIE		91.1
3	AliceMind & DURL	StructBERT + CLEVER		91.0
4	liangzhu ge	DEBERTa + CLEVER		90.9
5	DeBERTa Team - Microsoft	DeBERTa / TuringNLR		90.8
6	HFL iFLYTEK	MacALBERT + DKM		90.7
17	GLUE Human Baselines	GLUE Human Baselines		87.1

그림 5.2 GLUE 순위표 - 2021년 12월

새로운 모델과 인간 기준값의 순위는 계속 바뀔 것이다. 이 순위로 고전적인 NLP와 트랜스포머가 얼마나 발전했는지 알 수 있다!

먼저 GLUE의 인간 기준값이 상위권이 아님을 알 수 있다. NLU 모델이 GLUE에서 일반적인 인간을 능가한 것이다. 인간 기준값은 인간이 달성한 점수를 나타낸다. AI는 이제 인간을 능가한다. 2021년 12월에, 인간 기준값은 17위에 불과했다. 이렇게 되면 문제가 하나 발생한다. 뛰어넘기 위한 목표없이 벤치마크 데이터셋을 맹목적으로 사용하며 모델을 개선하기엔 한계가 있기 때문이다.

또한 트랜스포머 모델이 상위에 있음을 알 수 있다.

언어 이해 관점에서, GLUE와 SuperGLUE는 혼돈의 영역에 있는 단어를 질서의 영역으로 이동시키는 것과 같다. '이해'라는 것은 단어들을 함께 맞춰 언어로 만드는 일종의 접착제이다.

NLU가 발전함에 따라 GLUE 순위표도 계속 변화하겠지만, 왕(Wang et al.)은 더 높은 인간 기준값을 제시하기 위해 SuperGLUE를 만들었다.

### 5.2.2.2 더 높은 인간 기준값 제시

왕(Wang et al.)은 GLUE의 한계를 확인하고 더 어려운 NLU 과제인 SuperGLUE를 설계했다.

2020년 12월, SuperGLUE로 인해 인간 기준값은 즉시 1위가 되었다. 순위표는 <https://super.gluebenchmark.com/leaderboard>에서 확인할 수 있다.

Rank	Name	Model
1	SuperGLUE Human Baselines	SuperGLUE Human Baselines
+ 2	T5 Team - Google	T5
+ 3	Huawei Noah's Ark Lab	NEZHA-Plus

그림 5.3 SuperGLUE 순위표 2.0 - 2020년 12월

하지만 NLU 모델이 발전하며 SuperGLUE 순위표 역시 변화했다. 이미 2021년에 트랜스포머가 인간 기준값을 넘어섰다. 2021년 12월, 인간 기준값은 그림 5.4처럼 5위로 내려갔다.

Rank	Name	Model	URL	Score
1	Microsoft Alexander v-team	Turing NLR v5		90.9
2	ERNIE Team - Baidu	ERNIE 3.0		90.6
3	Zirui Wang	T5 + UDG, Single Model (Google Brain)		90.4
4	DeBERTa Team - Microsoft	DeBERTa / TuringNLRv4		90.3
5	SuperGLUE Human Baselines	SuperGLUE Human Baselines		89.8

그림 5.4 SuperGLUE 순위표 2.0 - 2021년 12월

AI 알고리즘의 순위는 새로운 혁신적인 모델이 나올 때마다 바뀔 것이다. 순위표를 보면 NLP 패권 경쟁이 얼마나 치열한지 알 수 있다.

이제 평가 과정이 어떻게 진행되는지 살펴보겠다.

### 5.2.2.3 SuperGLUE 평가 과정

왕(Wang et al.)은 SuperGLUE 벤치마크를 위해 여덟 가지 작업을 선정했다. 작업 선택의 기준은 GLUE에 비해 더 엄격했다. 작업을 수행하려면 텍스트를 이해할 뿐 아니라 추론까지 해야 했다. 인간 전문가 정도의 추론은 아니지만 많은 인간을 대체할 수준은 필요했다.

여덟 가지 작업 목록은 그림 5.5에서 볼 수 있다.

Name	Identifier	Download	More Info	Metric
Broadcoverage Diagnostics	AX-b			Matthew's Corr
CommitmentBank	CB			Avg. F1 / Accuracy
Choice of Plausible Alternatives	COPA			Accuracy
Multi-Sentence Reading Comprehension	MultiRC			F1a / EM
Recognising Textual Entailment	RTE			Accuracy
Words in Context	WIC			Accuracy
The Winograd Schema Challenge	WSC			Accuracy
BoolQ	BoolQ			Accuracy
Reading Comprehension with Commonsense Reasoning	ReCoRD			F1 / Accuracy
Winogender Schema Diagnostics	AX-g			Gender Parity / Accuracy

[DOWNLOAD ALL DATA](#)

그림 5.5 SuperGLUE 작업 목록

<https://super.gluebenchmark.com/tasks>에서 확인할 수 있다.

- 각 작업에는 해당 작업에 필요한 정보가 링크로 제공된다.
- Name(이름)은 사전 학습되고 미세 조정된 모델로 수행할 다운스트림 작업의 이름이다.
- Identifier(식별자)는 이름의 짧은 버전 또는 약어이다.
- Download(다운로드)는 데이터셋을 받을 수 있는 링크이다.
- More Info(추가 정보)는 작업을 설계한 팀의 논문 또는 웹사이트 링크로 자세한 정보를 제공한다.
- Metric(지표)은 모델 평가에 사용하는 측정 방법이다.

SuperGLUE는 작업 지침, 소프트웨어, 데이터셋과 문제를 설명하는 논문 또는 웹사이트를 제공한다. 벤치마크 작업을 수행하고 순위표에 도달하면 다음과 같이 결과가 표시된다.

Score	BoolQ	CB	COPA	MultiRC	ReCoRD	RTE	WIC	WSC	AX-b	AX-g
89.8	89.0	95.8/98.9	100.0	81.8/51.9	91.7/91.3	93.6	80.0	100.0	76.6	99.3/99.7

그림 5.6 SuperGLUE 작업 점수

SuperGLUE는 총 점수와 함께 각 개별 작업의 점수를 표시한다.

2019년 왕이 논문에서 소개한 COPA(Choice of Plausible Answers)의 지시사항을 살펴보자.

먼저 2011년 로멜레(Roemmele et al.)의 논문을 알아야 한다. 요약하자면, NLU 모델의 기계적 사고(인간적 사고가 아니라) 능력을 확인한다는 내용이다. 이 작업에서 트랜스포머는 질문에 대한 가장 그럴듯한 답변을 선택해야한다. 데이터셋은 전제를 제공하고 트랜스포머 모델은 가장 타당한 답을 찾아야 한다.

다음 예시를 보자.

**전제:** 나는 이웃집 문을 두드렸다. 그 결과 어떤 일이 일어날까요?

**대안 1:** 이웃이 나를 안으로 초대했다.

**대안 2:** 이웃이 집을 나갔다.

질문에 대답하려면 인간은 1~2초가 필요하며, 이는 상식수준의 기계적 사고력이 필요함을 의미한다. SuperGLUE의 작업 페이지에서 COPA.zip 데이터셋을 바로 확인할 수 있다.<sup>12</sup> 공정과 신뢰를 위해 벤치마크 경쟁에 참가하는 모든 참가자들에게 평가 방법 또한 공개했다.

예제가 어려워 보일 수 있다. 하지만 트랜스포머는 COPA의 인간 기준값에 근접하고 있으며(그림 5.7), 전체 작업을 다 고려하면 인간 기준값은 5위에 해당한다.

Rank	Name	Model	URL	Score	COPA
1	Microsoft Alexander v-team	Turing NLR v5		90.9	98.2
2	ERNIE Team - Baidu	ERNIE 3.0		90.6	97.4
3	Zirui Wang	T5 + UDG, Single Model (Google Brain)		90.4	98.0
4	DeBERTa Team - Microsoft	DeBERTa / TuringNLRv4		90.3	98.4
5	SuperGLUE Human Baselines	SuperGLUE Human Baselines		89.8	100.0

그림 5.7 SuperGLUE에서 COPA 결과

12 역주. SuperGLUE 작업 페이지에서 실제 데이터셋을 확인하면 한글로 적혀있지 않다. 본 책에서는 SuperGLUE 작업 이해를 돕기 위해 영어 원문 대신 한글로 번역 결과를 실었다.

트랜스포머는 믿기 어려울 정도로 짧은 시간만에 순위표의 상위권에 올랐다! 이걸 단지 시작일 뿐이다. 새로운 아이디어가 거의 매달 등장하고 있다!

COPA에 대해 알아보았고, 이제 다른 7가지 SuperGLUE 벤치마크 작업을 알아보겠다.

### 5.2.3 SuperGLUE 벤치마크 작업 알아보기

작업은 사전 학습에 사용될 수도, 또는 미세 조정을 위한 다운스트림 작업으로 활용될 수도 있다. 하지만 SuperGLUE의 목적은 NLU 모델이 미세 조정으로 다양한 다운스트림 작업을 수행할 수 있음을 보이는 것이다. 다양한 작업을 수행하면 트랜스포머 모델의 사고력을 증명할 수 있다.

트랜스포머의 힘은 사전 학습된 모델로 여러 작업을 수행하고, 미세 조정된 다운스트림 작업에 적용하는 능력에서 나온다. 오리지널 트랜스포머를 비롯한 다양한 응용 모델이 GLUE와 SuperGLUE에서 선두를 달리고 있다. 계속해서 인간 기준값을 능가하기 쉽지 않은 SuperGLUE 다운스트림 작업을 확인해 보겠다.

이전 섹션에서는 COPA를 살펴보았다. 이번 섹션에서는, 왕(Wang et al.)이 그림 5.5에서 정의한 나머지 7가지 작업을 살펴보자.

불리언 질문(Boolean question, BoolQ) 작업부터 시작하겠다.

#### 5.2.3.1 BoolQ

BoolQ는 예-아니오를 답하는 작업이다. SuperGLUE의 BoolQ 데이터셋은 자연스러운 상황 예시 15,942개로 구성되어 있다. 다음은 데이터셋 train.jsonl 파일의 3번째 줄에 있는 샘플로 지문 (passage), 질문(question), 답(answer)으로 이루어져 있다.

{"질문": "윈도우 무비 메이커는 윈도우 필수 패키지에 포함되어 있나요?"

"지문": "윈도우 무비 메이커 -- 윈도우 무비 메이커(이전 윈도우7에서는 윈도우 라이브 무비 메이커로 알려진)는 마이크로소프트의 단종된 영상 편집 소프트웨어다. 윈도우 필수 패키지의 일부이며 영상을 만들고 편집하여 원드라이브, 페이스북, 비메오, 유튜브, 플리커에 게시할 수 있다.", "idx": 2, "라벨": true}

시간이 지나면서 데이터셋의 내용이 변경될 순 있어도, 개념은 동일할 것이다.

이제 인간과 기계 모두에게 집중력을 요하는 CB를 살펴보자.

### 5.2.3.2 커밋먼트 뱅크(CB)

커밋먼트 뱅크(Commitment Bank, CB)는 함의(entailment) 능력이 필요한 어려운 작업이다. 트랜스포머에게 전제(premise)를 읽도록 한 후, 가설(hypothesis)이 전제를 충족하는지 확인하게 한다. 가설은 전제를 충족하거나 반박할 수 있다. 트랜스포머 모델은 전제를 기반으로 가설에 대해 중립(neutral), 함의, 반박(contradiction) 중 하나를 선택한다.

데이터셋은 자연스러운 대화로 이루어져있다.

다음은 train.jsonl의 77번 샘플로, CB 작업의 어려움을 보여준다.

```
{"전제": "수스웨카. 수족어로 '잠자리'라는 뜻이죠. 풀과 제가 거기서 만났다고 말했던가요?"  
"가설": "수스웨카는 그녀와 풀이 만난 곳입니다."  
"레이블": "함의", "IDX": 77}
```

이제 다중 문장 독해 과제를 살펴보자.

### 5.2.3.3 다중 문장 독해(MultiRC)

다중 문장 독해(Multi-Sentence Reading Comprehension, MultiRC)는 모델에게 텍스트를 읽고 여러 선택지 중 고르게 하는 작업으로, 인간과 기계 모두에게 어려운 작업이다. 하나의 본문(text)과 여러 개의 질문(questions), 각 질문에 대한 답변(answers)과 0(거짓), 1(참) 형태의 라벨이 모델에 주어진다.

다음은 train.jsonl의 두 번째 샘플이다.

"텍스트": "10월 17일에 집회가 열렸고, 사건은 2월 29일에 발생했다. 다시 말하지만, 일반적인 영화 제작 기법들은 자연스러운 왜곡을 허용한다. '무어는 당신의 맥락을 박탈하고 공백을 매우는 방식으로 작업한다. 그것은 뛰어나지만 비윤리적이다.' 앞에서 언급한 것처럼, '나의 차가운 죽은 손에서부터' 부분은 그저 무어가 헤스톤을 소개하기 위한 방법일 뿐이다. 무어의 비평가들 이외에 다른 사람들은 그것을 다른 의도로 해석했을까? 무어가 자신을 비판하는 사람들이 주장하는 것처럼 교묘한 속임수의 주인공이었다면, 이를 두 번이나 반복하는 것은 전혀 이치에 맞지 않을 것이다. 헤스톤이 롤랜드에 대해 물었던 조지타운 호야 인터뷰에 대해 당신은 '그가 케일라 롤랜드의 사건을 알아챘다는 어떠한 표시도 없다'고 썼다. 이는 극도로 순진한 주장이다. 헤스톤은 가장 주목받는 총기 폭력 사건에 대한 최신 정보를 얻지 못하면 NRA의 회장이 될 수 없었을 것이다. 그가 인터뷰의 그 부분에 답하지 않았다 하더라도, 그는 그 시점에서 그 사건에 대해 알고 있었을 것이다. 케일라 롤랜드 사건에 대한 NRA 웹사이트 발췌문과 "케일라 롤랜드 사망 후 46시간"이라는 구절을 강조한 점에 대해서는 여기서 제기한 비판이 타당하다. 그러나 이는 의도적인 왜곡으로 묘사하는 것과는 거리가 멀며, 오히려 무어의 빠른 편집으로 인해 사실을 놓치기 쉬운 예시다. 문장이 강조된 이유는 헤스톤이 급히 플린트로 이동하여 집회를 개최한 것으로 시청자를 속이기 위함이 아니라, 이 단락에서 처음으로 '케일라 롤랜드'라는 이름이 언급되었기 때문이다."

샘플엔 질문 4개가 들어있다. 작업을 설명하기 위해, 2개만 확인하겠다. 모델은 라벨을 올바르게 예측해야 한다. 모델에 요청하는 정보의 모습을 확인해 보자.

```
"question": "카일라 롤랜드(Kayla Rolland)는 언제 총에 맞았는가?"
"answers":
[{"text": "2월 17일", "idx": 168, "label": 0},
{"text": "2월 29일", "idx": 169, "label": 1},
{"text": "10월 29일", "idx": 170, "label": 0},
{"text": "10월 17일", "idx": 171, "label": 0},
{"text": "2월 17일", "idx": 172, "label": 0}], "idx": 26},
{"question": "2월 29일에 NRA의 회장은 누구였나?"
"answers": [{"text": "찰턴 헤스톤(Charleton Heston)", "idx": 173, "label": 1},
{"text": "무어(Moore)", "idx": 174, "label": 0},
{"text": "조지 호야(George Hoya)", "idx": 175, "label": 0},
{"text": "롤랜드(Rolland)", "idx": 176, "label": 0},
{"text": "호야(Hoya)", "idx": 177, "label": 0}, {"text": "카일라(Kayla)", "idx": 178, "label": 0}], "idx": 27},
```

사전 학습과 미세 조정을 거친 단일모델이 이런 어려운 다운스트림 작업을 수행하는 것을 보면 그 성능에 감탄할 수밖에 없다.

이제 상식 추론 독해 과제를 알아보자.

### 5.2.3.4 상식 추론 독해 데이터셋(ReCoRD)

상식 추론 독해 데이터셋(Reading Comprehension with Commonsense Reasoning Dataset, ReCoRD)

또한 매우 어려운 작업이다. 데이터셋은 70,000개 이상의 뉴스에서 추출한 120,000개 이상의 쿼리를 포함하고 있다. 트랜스포머는 문제를 해결하기 위해 상식추론 능력을 사용해야 한다.

train.jsonl의 다음 예제를 살펴보자.

```

"source": "데일리 메일"
지문에는 텍스트와 함께 개체의 위치가 제공된다.
먼저 텍스트가 주어진다.
"passage": {
"text": "한때 잉카의 존경을 받았던 페루의 한 부족이 치열한 사냥 기술과 강력한 전사로 남아메리카의
코카 재배 계곡에서 마약 밀매업자, 반군, 불법 벌목꾼과 땅을 공유하며 전통적 삶을 이어가고 있다. 아샤닌
카 인디언은 산악 국가인 페루의 아마존 지역에서 가장 큰 원주민 집단이지만, 현재 페루 인구 3천만 명 중
1% 미만에 불과할 정도로 거주지가 매우 드물다. 남아메리카 열대우림에서 원주민 통치 시절 영토와 식량을
놓고 라이벌 부족과 싸웠던 아샤닌카 부족은 평화를 누린 적이 거의 없다.Wn@highlightWn아샤닌카
부족은 한때 다음과 같이 말했다.
수백 년 전 잉카인처럼 아마존에 살던 원주민들은 수년간의 분쟁으로 반군과 마약상이 숲으로 들어온 후 땅
을 강제로 빼앗겼다Wn@highlightWn. 코카가 풍부한 계곡에 정착했지만, 그들은 산업화 이전의 가난
한 삶을 살고 있다".

```

개체(entities) 목록은 다음과 같이 주어진다.

```

"entities": [{"start": 2,"end": 9}, ..., {"start": 711,"end": 715}]

```

이제, 모델은 쿼리의 지정위치(placeholder)에 해당하는 답을 찾아야 한다.

```

{"query": "젊음의 순수함: 많은 @placeholder의 젊은 세대가 부족 생활에서 등을 돌리고
생활 조건이 더 좋은 도시로 이동했다","answers":[{"start":263,"end":271,"text":":아샤닌카(Asha
ninka)"}, {"start":601,"end":609,"text":":아샤닌카(Ashaninka)"}, {"start":651,"end":659,"te
xt":":아샤닌카(Ashaninka)"}],"idx":9},"idx":3}

```

이 문제를 수행한 트랜스포머는, 다음으로 함의 작업을 수행한다.

### 5.2.3.5 텍스트 함의 인식(RTE, Recognizing Textual Entailment)

텍스트 함의 인식 작업에서, 트랜스포머는 전제를 읽고 가설을 검토한 후 함의 여부에 대한 라벨을 예측해야 한다.

다음은 train.jsonl 데이터셋 파일의 19번째 예제이다.

```
{"premise": "미국 원유는 배럴당 $ 1.32 하락한 42.83 달러에 마감했다.", "hypothesis": "미국산 원유는 배럴당 1.32달러 하락하여 42.83달러에 마감했다.", "label": "NOT_ENTAILMENT", "idx": 19}
```

RTE 작업을 수행하려면 이해와 논리가 필요하다. 이제 단어맥락 작업을 살펴보자.

### 5.2.3.6 단어 맥락(WiC, Words in Context)

단어 맥락 작업과 다음으로 소개할 위노그라드 스키마 작업은 모호한 단어를 처리하는 능력을 평가한다. 단어 맥락에서 트랜스포머는 두 문장을 분석하고 타깃 단어가 같은 의미를 가지는지 판단해야 한다.

train.jsonl의 첫 번째 샘플을 살펴보겠다.

먼저, 타깃 단어가 주어진다.

```
"단어": "장소"
```

타깃 단어를 포함하는 두 문장이 모델에게 주어진다.

```
"sentence1": "나중에 내 집에 오겠습니까?".  
"sentence2": "약자를 위한 자리가 없는 정치 시스템"
```

train.jsonl 파일에는 샘플 번호, 정답라벨, 문장 1과 문장 2에서의 타깃 단어 위치(start1, end1, start2, end2)가 표시되어 있다.

```
"idx": 0,  
"label": false,  
"start1": 31,  
"start2": 27,  
"end1": 36,  
"end2": 32,
```

이 어려운 작업 후에는 위노그라드 작업이 남아있다.

### 5.2.3.7 위노그라드 스키마 챌린지(WSC)

위노그라드 스키마 챌린지(WSC, Winograd schema challenge)라는 명칭은 테리 위노그라드(Terry Winograd)의 이름을 따서 지어졌다. 트랜스포머가 잘 훈련되었다면 증의성 해소 문제도 해결할 수 있어야 한다는 의도를 담고 있다.

데이터셋엔 대명사에 미묘한 차이가 생길 수 있는 문장이 들어있다.

위노그라드 스키마 챌린지는 대명사가 무엇을 의미하는지 찾는 작업이다. 가장 어려운 작업 중 하나이지만 트랜스포머 아키텍처의 셀프 어텐션은 이 작업에 이상적이다.

각 문장에는 두 대상과 대명사가 포함되어 있고, 대명사가 어느 대상을 지칭하는지 찾아내야 한다.

train.jsonl의 샘플을 보자.

먼저, 샘플은 모델에게 문장을 제시한다.

```
{
  "text": "I poured water from the bottle into the cup until it was full.(병에 담긴 물을 컵에 가득 찰 때까지 부었다).",
  "WSC": "모델이 타깃 대명사인 10번째 토큰을 찾으시오.",
  "target": {
    "span2_index": 10,
    "span1_index": 7,
    "span1_text": "the cup",
    "span2_text": "it"
  },
  "idx": 4,
  "label": true
}
```

SuperGLUE의 주요 작업을 몇 가지 살펴보고, 이 외에도 많은 작업이 존재한다.

트랜스포머의 아키텍처를 이해하고 벤치마크 작업의 메커니즘을 이해한다면 어떤 모델이나 벤치마크도 빠르게 적용할 수 있을 것이다.

이제 몇 가지 다운스트림 작업을 실행해보자.

## 5.3

# 다운스트림 작업 실행하기

이번 섹션에서는 직접 트랜스포머라는 자동차를 운전하며 어떻게 동작하는지 살펴보겠다. 다양한 모델과 작업이 있지만, 이 섹션에서는 일부만 실행해보려 한다. 몇 가지 작업의 실행 과정을 이해하고 나면, 나머지도 빠르게 이해할 수 있을 것이다. 어쨌든 모든 작업의 인간 기준값은 바로 우리다!

다운스트림 작업은 사전 학습된 트랜스포머 모델로부터 파라미터를 상속받아 미세 조정된 트랜스포머로 수행한다.

다운스트림 작업은 미세 조정 작업을 수행하는 사전 학습 모델의 관점이다. 즉, 사전 학습에 사용되지 않은 작업이어야 다운스트림 작업이라고 부를 수 있다. 이번 섹션에 등장하는 모든 작업은, 사전 학습에 사용되지 않았으므로, 다운스트림 작업으로 간주하겠다.

모델은 데이터베이스, 벤치마크 방법, 정확도 측정 방법, 순위표 기준 등과 함께 계속 변화한다. 그러나 이 챕터의 다운스트림 작업에 반영된 인간의 사고 구조는 계속 유지될 것이다.

CoLA부터 시작하겠다.

### 5.3.1 언어 적합성 말뭉치(CoLA)

언어 적합성 말뭉치(Corpus of Linguistic Acceptability, CoLA)는 GLUE 작업 중 하나로, 문법적 적합성이 표시된 수천 개의 영어 문장으로 이루어져 있으며, <https://gluebenchmark.com/tasks>에서 확인할 수 있다.

2019년 알렉스 바르슈타트(Alex Warstadt et al.)의 목표는 문장의 언어 적합성에 대한 능력을 측정하는 것이었다. 따라서 NLP 모델로 문장 단위의 판별을 수행한다.

각 문장은 '문법적' 또는 '비문법적' 중 하나로 표기 되어있다. 문장이 문법적으로 적합하지 않으면 0으로, 적합하다면 1로 표기된다.

분류 = 1, '우리는 쉰 목소리로 소리를 질렀다.'

분류 = 0, '우리는 스스로 소리를 질렀다'

3장, BERT 모델 미세 조정하기의 BERT\_Fine\_Tuning\_Sentence\_Classification\_GPU.ipynb를 살펴보면, CoLA 데이터셋으로 미세 조정된 BERT 모델을 볼 수 있다. 다음과 같이 CoLA 데이터를 사용했었다.

```
#@title 데이터셋 불러오기
#데이터셋 출처 : https://nyu-ml1.github.io/CoLA/
df = pd.read_csv("in_domain_traints_v", delimiter='Wt', header=None,
names=['sentence_source', 'label', 'label_notes', 'sentence'])
df.shape
```

사전 학습된 BERT 모델 역시 불러왔었다.

```
#@title 허깅페이스 BERT 기본 모델 (Uncased) 불러오기
model = BertForSequenceClassification.from_pretrained("bert-base-uncased",
num_labels=2)
```

마지막으로, 3장, BERT 모델 미세 조정하기의 매튜 상관 계수로 평가하기 섹션과 이번 장에서 살펴본 MCC를 측정방법으로 사용했었다.

MCC에 대한 수학적 설명이 필요하다면 이전 챕터의 매튜 상관 계수로 평가하기 섹션을 복습하거나 소스 코드를 다시 실행해 볼 수 있다.

문법적으로 틀린 문장일지라도 여전히 감성을 전달할 수 있다. 감성 분석은 기계에게 일종의 공감능력을 더해준다.

### 5.3.2 스탠퍼드 감성 트리뱅크(SST-2)

스탠퍼드 감성 트리뱅크(Stanford Sentiment TreeBank)는 영화 리뷰를 포함하고 있다. 이번 섹션에서는 SST-2(이진 분류)를 소개하려 하지만, 이 데이터셋은 단순한 이진 분류를 넘어 0(부정)부터 n(긍정)까지 감정을 분류할 수 있다.

2013년 소처(Socher et al.)는 긍정-부정의 이진 분류 이상으로 감성을 분석했다. 12장, 고객 감정을 감지해 예측하기에서 트랜스포머 모델로 SST-2 다중 라벨 감성 분류(multi-label sentiment classification)를 살펴볼 것이다.

이번 섹션에서는, SST 샘플의 이진 분류를 위해 허깅페이스 트랜스포머의 파이프라인을 사용한다.

Transformer\_tasks.ipynb를 열고 다음 셀을 실행하여 SST에서 가져온 긍정적, 부정적 영화 리뷰를 분석한다.

```
#@title SST-2 이진 분류
from transformers import pipeline
nlp = pipeline("sentiment-analysis")
print(nlp("If you sometimes like to go to the movies to have fun , Wasabi is a good place to start."),"If you sometimes like to go to the movies to have fun , Wasabi is a good place to start.") # (영화를 볼 때 가끔 재미있는 시간을 보내고 싶다면, 와사비는 좋은 시작점이다.)
print(nlp("Effective but too-tepid biopic."),"Effective but too-tepid biopic.") # (효과적이지만 너무 미지근한 전기영화다.)
```

정확한 출력값이 나온다.

```
[{'label': 'POSITIVE', 'score': 0.999825656414032}] If you sometimes like to go to the movies to have fun , Wasabi is a good place to start . # (영화를 볼 때 가끔 재미있는 시간을 보내고 싶다면, 와사비는 좋은 시작점이다.)
[{'label': 'NEGATIVE', 'score': 0.9974064230918884}] Effective but too-tepid biopic. # (효과적이지만 너무 미지근한 전기영화다.)
```

SST-2 작업은 정확도로 평가한다.

문장의 감성을 분류해 보았다. 이제 두 문장의 패러프레이즈(paraphrase) 여부를 확인해 보자.

### 5.3.3 마이크로소프트 패러프레이즈 말뭉치(MRPC)

마이크로소프트 패러프레이즈 말뭉치(Microsoft Research Paraphrase Corpus, MRPC)는 GLUE 작업으로, 웹상에서 추출한 문장 쌍을 포함한다. 사람이 직접 각각의 문장 쌍에 대해, 두 문장이 동등한 지 두 가지 기준으로 표시했다.

- 패러프레이즈(paraphrase) 동등성
- 의미적 동등성

히킹페이스의 BERT 모델로 샘플을 실행해 보겠다. `Transformer_tasks.ipynb`를 열고 다음 셀로 이동한 후, MRPC에서 발췌한 샘플을 실행한다.

```
#@title 시퀀스 분류하기: 패러프레이즈 분류하기
from transformers import AutoTokenizer,
TFAutoModelForSequenceClassification
import tensorflow as tf
tokenizer = AutoTokenizer.from_pretrained("bert-base-cased-finetuned-
mrpc")
model = TFAutoModelForSequenceClassification.from_pretrained("bert-base-
cased-finetuned-mrpc")
classes = ["not paraphrase", "is paraphrase"]
sequence_A = "The DVD-CCA then appealed to the state Supreme Court."
# (DVD-CCA는 주 최고법원에 항소했다.)
sequence_B = "The DVD CCA appealed that decision to the U.S. Supreme."
# (DVD CCA는 그 결정을 미국 대법원에 항소했다.)
paraphrase = tokenizer.encode_plus(sequence_A, sequence_B, return_
tensors="tf")
paraphrase_classification_logits = model(paraphrase)[0]
paraphrase_results = tf.nn.softmax(paraphrase_classification_logits,
axis=1).numpy()[0]
print(sequence_B, "should be a paraphrase")
for i in range(len(classes)):
    print(f"{classes[i]}: {round(paraphrase_results[i] * 100)}%")
```

정확한 결과가 나오지만, 모델에 더 많은 다운스트림 학습이 필요하다는 경고 메시지가 나올 수 있다.

The DVD CCA appealed that decision to the U.S. Supreme Court, should be a paraphrase

not paraphrase: 8.0%

is paraphrase: 92.0%

(DVD CCA는 그 결정을 미국 대법원에 항소했다. 패러프레이즈 여부.

패러프레이즈 아님: 8%

패러프레이즈이다: 92%)

이제 위노그라드 스키마를 실행해 보겠다.

### 5.3.4 위노그라드 스키마

이 챕터의 위노그라드 스키마 챌린지(WSC) 섹션에서 위노그라드 스키마를 알아보았고, 영어로 된 학습 데이터를 확인했다.

하지만 트랜스포머 모델에게 영어-프랑스어 번역의 대명사 성별 문제를 풀도록 하면 어떻게 될까? 프랑스어는 문법적 성별(여성형, 남성형)에 따라 명사의 철자가 달라진다.

다음 문장은 'car'나 'garage'를 의미할 수 있는 대명사 'it'을 포함하고 있다. 과연 트랜스포머 모델은 대명사를 구분할 수 있을까?

Transformer\_tasks.ipynb를 열고, #위노그라드 셀로 이동하여 다음 예제를 실행한다.

```
#@title 위노그라드
from transformers import pipeline
translator = pipeline("translation_en_to_fr")
print(translator("The car could not go in the garage because it was too
big.", max_length=40)) # (차가 너무 커서 차고에 들어갈 수 없다.)
```

번역 결과는 완벽하다.

```
['translation_text': "La voiture ne pouvait pas aller dans le garage  
parce qu'elle était trop grosse."]]
```

트랜스포머는 'it'이 여성형인 'car'를 의미함을 알아냈다. 'it'과 형용사 'big'에 여성형을 적용했다.

'it'의 번역에 사용된 'elle'는 프랑스어로 'she'를 의미한다. 남성형을 적용했다면 'he'를 의미하는 'il'을 사용했을 것이다.

'grosse'는 'big'의 여성형이다. 남성형이라면 'gros'를 사용했을 것이다.

트랜스포머가 난이도 높은 위노그라드 스키마를 정확하게 풀었다.

데이터 기반 NLU 작업은 매우 다양하다. 트랜스포머 활용 능력을 넓혀나가기 위해, 이 책 전반에 걸쳐 여러 작업을 더 살펴볼 것이다.

## 5.4

## 정리하기

이 장에서는 인간의 언어 표현 방식과 기계 지능의 트랜스덕션 방식의 차이를 분석했다. 트랜스포머는 우리의 매우 복잡한 사고 과정을 표현해 놓은 텍스트에 의존해야 한다. 언어는 다량의 정보를 가장 정확하게 표현하는 방법이다. 하지만 기계는 감각기관이 없어서 원시 데이터가 음성(speech) 데이터인 경우 텍스트로 변환해 사용해야 한다.

그리고 나서, 멀티태스크 트랜스포머의 성능 측정 방법을 확인했다. 다운스트림 작업에서 최고의 성능을 얻는 트랜스포머의 능력은 NLP 역사에서 독보적이다. 난이도 높은 SuperGLUE 작업을 살펴보았다. 트랜스포머는 GLUE와 SuperGLUE의 순위표에서 최고 순위를 차지했다.

BoolQ, CB, WiC 등 여러 작업은 인간에게도 쉽지 않다. 트랜스포머 모델이 효율성을 입증하지 못한 몇몇 다운스트림 작업과 예제를 살펴보았다.

트랜스포머는 이전 NLU 아키텍처를 능가함으로써 그 가치를 입증했다. 구글 코랩 노트북과 허깅페이스 파이프라인으로 여러 작업을 실행하며, 미세 조정된 다운스트림 작업을 얼마나 쉽게 구현할 수 있는지 보았다.

위노그라드 스키마에서, 영어-프랑스어 번역으로 트랜스포머가 난이도 높은 위노그라드 중의성 해소 문제를 해결하도록 했다.

다음 6장, 트랜스포머를 사용한 기계 번역에서는, 번역 작업에서 한 걸음 더 나아가 트렉스(Trax)를 사용하여 번역 모델을 구축할 것이다.

## 5.5 문제

01. 기계 지능은 인간과 동일한 데이터를 사용해 예측을 수행한다. 참 / 거짓
02. SuperGLUE는 GLUE보다 더 어려운 NLP 과제이다. 참 / 거짓
03. BoolQ는 예-아니오를 답하는 작업이다. 참 / 거짓
04. WiC는 “Words in Context”의 약자이다. 참 / 거짓
05. 텍스트 함의 인식(RTE)은 하나의 시퀀스가 다른 한 시퀀스를 함의하는지 판단하는 과제이다. 참 / 거짓
06. 위노그라드 스키마는 동사의 철자가 올바른지 판단하는 과제이다. 참 / 거짓
07. 트랜스포머 모델은 현재 GLUE와 SuperGLUE의 최상위 순위를 차지했다. 참 / 거짓
08. 인간 기준값은 변할 수 있다. SuperGLUE에 의해 더 어려운 기준으로 바뀌었다. 참 / 거짓
09. 트랜스포머 모델은 SuperGLUE의 인간 기준값을 절대 넘지 못할 것이다. 참 / 거짓
10. 트랜스포머를 응용한 다양한 모델이 RNN과 CNN을 뛰어넘었다. 참 / 거짓

## 5.6 참고 문헌

- Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, Samuel R. Bowman, 2019, SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems: <https://w4ngatang.github.io/static/papers/superglue.pdf>
- Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, Samuel R. Bowman, 2019, GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding
- Yu Sun, Shuohuan Wang, Yukun Li, Shikun Feng, Hao Tian, Hua Wu, Haifeng Wang, 2019, ERNIE 2.0: A Continual Pretraining Framework for Language Understanding: <https://arxiv.org/pdf/1907.12412.pdf>
- 허깅페이스의 트랜스포머 사용법: <https://huggingface.co/transformers/usage.html>

# 06장

---

## 트랜스포머를 사용한 기계 번역

---

- 6.1 기계 번역 정의하기
- 6.2 WMT 데이터셋 전처리하기
- 6.3 BLEU로 기계 번역 평가하기
- 6.4 구글 번역으로 번역하기
- 6.5 트렉스로 번역하기
- 6.6 정리하기
- 6.7 문제
- 6.8 참고 문헌

인간은 시퀀스(sequence)를 특정 대상으로 해석(transduction)할 수 있다. 우리는 문자열을 보면 그것이 의미하는 이미지를 머릿속에 쉽게 떠올린다. 누군가 “내 정원에 있는 꽃이 아름답다”라고 말하면 우리는 꽃이 있는 정원을 쉽게 떠올릴 수 있다. 우리는 그 정원을 본 적이 없어도 정원의 이미지를 떠올린다. 지저귀는 새소리와 꽃향기도 상상할 수 있다.

기계는 수학적 개념(numerical representation)을 사용하여 트랜스덕션을 학습한다. 순환(recurrent) 방식이나 컨볼루션(convolutional) 방식은 흥미로운 결과를 가져왔지만 BLEU 번역 평가 점수가 좋지 않았다. 번역을 하려면 언어 A의 개념을 언어 B로 바꾸어야 한다.

트랜스포머 모델의 셀프 어텐션(self-attention)은 기계의 분석 능력을 향상한다. 언어 A 시퀀스를 언어 B로 번역하기 전에 적절하게 표현되어야 한다. 셀프 어텐션은 기계가 더 나은 BLEU 점수를 얻을 수 있도록 더 좋은 지능을 갖추게 한다.

2017년 「Attention Is All You Need」<sup>13</sup>에서 소개한 트랜스포머가 영어-독일어 및 영어-프랑스어 번역 부문에서 최고 점수를 얻었다. 점수는 그 이후로 다른 트랜스포머들에 의해 계속 경신되고 있다.

이 책에서 우리는 트랜스포머 아키텍처, RoBERTa 모델 처음부터 학습하기, BERT(Bidirectional Encoder Representations from Transformers) 미세 조정하기, 미세 조정된 BERT 평가하기, 몇 가지 트랜스포머 예제로 다운스트림 작업 살펴보기 등 트랜스포머의 필수적인 요소들을 다루었다.

이번 장에서는 기계 번역을 세 가지 주제로 나누어 살펴본다. 먼저 기계 번역이 무엇인지 정의한다. 그리고 **WMT(Workshop on Machine Translation)** 데이터셋을 전처리한다. 마지막으로 기계 번역을 구현하는 방법을 살펴본다.

13 역주, 2장, 트랜스포머 모델 아키텍처 살펴보기를 참고하면 오리지널 트랜스포머 아키텍처의 구성 요소에 대해 알아보는 데 도움이 될 것이다.

이 장에서는 다음 주제들을 다룬다.

- 기계 번역 정의하기
- 사람이 하는 번역
- 기계가 하는 번역
- WMT 데이터셋 전처리하기
- BLEU로 기계 번역 평가하기
- 기하학적 평가
- 첸체리 스무딩(Chencherry smoothing)
- 구글 번역 API 소개
- 트랙스(Trax)로 영어-독일어 문제 초기화하기

첫째로 기계 번역을 정의해 보자.

## 6.1

# 기계 번역 정의하기

2017년 바스와니(Vaswani et al.)는 트랜스포머를 설계할 때 가장 어려운 NLP 문제 중 하나를 해결하고자 했다. 기계 번역을 인간 기준 성능(human baseline)으로 하는 것은 인간-기계(human-machine) 지능 설계자에게 쉽지 않아 보인다. 그럼에도 바스와니는 트랜스포머 아키텍처를 발표하고 최상의 BLEU 결과를 달성했다.

이 섹션에서는 기계 번역을 정의한다. 기계 번역은 트랜스덕션과 출력으로 인간의 번역을 재현하는 프로세스이다.

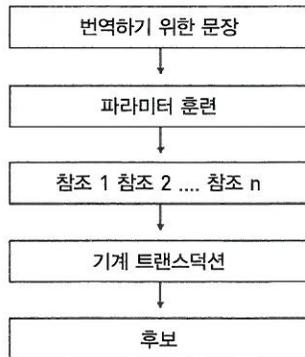


그림 6.1 기계 번역 프로세스

그림 6.1은 기계가 수행할 다음과 같은 작업을 보여준다.

- 번역할 문장을 선택한다.
- 수십억 파라미터를 사용하여 단어가 서로 어떻게 연관되는지 배운다.
- 단어가 서로 참조하는 다양한 방법을 학습한다.
- 기계 트랜스덕션을 사용하여 학습한 파라미터를 새로운 시퀀스로 옮긴다.
- 단어 또는 시퀀스의 번역 후보를 선택한다.

프로세스는 항상 원본(source) 언어 A로 이루어진 번역할 문장으로 시작하여 번역된 문장인 출력 B로 끝나며, 중간 계산에는 트랜스덕션이 포함된다.

## 6.1.1 인간 트랜스덕션과 번역

유럽 의회의 통역사는 문장을 단어 단위로 번역하지 않는다. 단어 단위 번역은 적절한 문법적 구조가 부족하고 각 단어의 문맥이 무시되어 올바른 번역을 생성할 수 없기 때문에 말이 안 되는 경우가 많다.

인간 트랜스덕션은 언어로 된 문장으로 문장의 의미에 대한 인지적 개념을 구축한다. 그런 다음 유럽 의회의 통역사(구두 번역) 또는 번역가(서면 번역)가 해당 트랜스덕션을 B 언어로 된 해석으로 변환한다.

통역사 또는 번역가가 B 언어로 번역한 문장을 레퍼런스(reference) 문장이라고 부르겠다. 그림 6.1의 기계 번역 프로세스에서 여러 레퍼런스 문장을 확인할 수 있다.

한 명의 인간 번역가가 문장 A를 문장 B로 번역할 때, 한 문장을 한 번만 번역한다 그러나 현실에서는 한 명 이상의 번역가가 문장 A를 각자 번역할 수 있다. 예를 들어 「몽테뉴의 수상록」을 프랑스어에서 영어로 번역한 경우에서 종종 찾을 수 있다. 원본의 프랑스어 문장 A 하나에도, 레퍼런스 1부터 n으로 표시된, 문장 B의 여러 버전이 있을 수 있다.

유럽 의회에 가보면 통역사가 제한된 시간(2시간) 동안에만 통역하는 것을 볼 수 있다. 그런 다음 다른 통역사가 통역을 이어받는다. 원본 언어로 된 문장 A는 같은 사람이 하루에 여러 번 말하지만 레퍼런스 문장 B는 여러 버전으로 번역될 수 있다.

$$\text{reference} = \{\text{reference 1, reference 2, \dots, reference n}\}$$

기계도 인간 번역가처럼 생각할 수 있는 방법을 찾아야 한다.

## 6.1.2 기계 트랜스덕션과 번역

오리지널 트랜스포머 아키텍처의 트랜스덕션 프로세스는 인코더 스택, 디코더 스택 및 모델의 모든 파라미터를 사용하여 레퍼런스 시퀀스를 표현한다. 우리는 이 출력 시퀀스를 레퍼런스라고 부른다.

왜 그냥 “출력 예측(prediction)”이라고 하지 않을까? 문제는 출력 예측이 하나가 아니라는 점이다. 트랜스포머는 인간처럼 참조할만한 결과를 생성하긴 하지만, 다른 방식으로 학습시키거나 다른 트랜스포머 모델을 사용하면 결과가 달라질 수 있다!

인간 수준으로 언어 시퀀스를 표현하는 것이 상당히 어렵지만, 지금까지 많은 진전이 있어왔다.

기계 번역 결과를 평가해 보면 NLP가 발전했음을 알 수 있다. 특정 솔루션이 다른 솔루션보다 더 나은지 판단하려면 각 NLP 도전자, 연구소 또는 조직이 동일한 데이터셋을 사용해야 한다.

이제 WMT 데이터셋을 살펴보자.

## 6.2

# WMT 데이터셋 전처리하기

2017년 바스와니는 WMT 2014 영어-독일어 번역 작업과 WMT 2014 영어-프랑스어 번역 작업에서 트랜스포머가 최상의 BLEU 점수를 달성했다고 발표했다. BLEU는 이번 챕터의 BLEU로 기계 번역 평가하기 섹션에서 설명한다.

2014 WMT은 다양한 유럽 국가의 언어 데이터셋을 포함하고, 유로파(Europarl) 말뭉치 버전 7에서 가져온 데이터셋도 포함되어 있다. 우리는 유럽 의회 회의록 병렬(parallel)<sup>14</sup> 말뭉치, 1996-2011 (<https://www.statmt.org/europarl/v7/fr-en.tgz>)의 프랑스어-영어 데이터셋을 사용한다.

먼저 파일을 다운로드하고 압축을 푼 뒤, 두 개의 병렬 파일을 전처리한다.

- europarl-v7.fr-en.en
- europarl-v7.fr-en.fr

이제 말뭉치를 불러오고, 지우고, 크기를 줄이는 등 전처리를 시작해 보자.

### 6.2.1 원시 데이터 전처리하기

이 섹션에서는 europarl-v7.fr-en.en 및 europarl-v7.fr-en.fr을 전처리한다.

깃허브의 이번 챕터 디렉터리에 있는 read.py를 연다. 두 유로파 파일은 read.py와 같은 위치에 있어야 한다.

직렬화(serialized)된 출력 파일을 덤프(dump)하기 위해 표준 파이썬 함수와 pickle을 사용한다.

14 역주. 동일한 내용의 서로 다른 언어로 구성되어 있음을 의미한다.

```
import pickle
from pickle import dump
```

파일을 메모리에 적재하는 함수를 정의한다.

```
# 문서를 메모리에 로드한다
def load_doc(filename):
    # 파일을 읽기 모드로 연다.
    file = open(filename, mode='rt', encoding='utf-8')
    # 모든 텍스트를 읽어들인다.
    text = file.read()
    # 파일을 닫는다
    file.close()
    return text
```

적재된 문서를 문장으로 분할한다.

```
# 로드된 문서를 문장으로 분리한다.
def to_sentences(doc):
    return doc.strip().split('\n')
```

가장 긴 문장의 길이와 가장 짧은 문장의 길이를 찾는다.

```
# 가장 짧은 문장 길이와 가장 짧은 문장 길이
def sentence_lengths(sentences):
    lengths = [len(s.split()) for s in sentences]
    return min(lengths), max(lengths)
```

문장에 있는 노이즈 토큰은 학습되지 않도록 정리해야 한다. 문장을 정규화하고, 공백에 따라 토큰화하며, 소문자로 변환한다. 각 토큰에서 구두점, 출력할 수 없는 문자, 숫자가 포함된 토큰은 제외한다. 정리한 문장은 문자열로 저장한다.

이 프로그램은 함수로 문장을 정리한 후 덧붙여(append) 반환한다.

```

# 문장을 정리한다.
import re
import string
import unicodedata
def clean_lines(lines):
    cleaned = list()
    # 문자 필터링을 위한 정규식을 준비한다.
    re_print = re.compile('[^%s]' % re.escape(string.printable))
    # 구두점 제거를 위해 번역 테이블을 준비한다.
    table = str.maketrans('', '', string.punctuation)
    for line in lines:

        # 유니코드 문자를 정규화한다.
        line = unicodedata.normalize('NFD', line).encode('ascii', 'ignore')
        line = line.decode('UTF-8')
        # 공백 기준으로 토큰나이즈 한다.
        line = line.split()
        # 소문자로 변환한다.
        line = [word.lower() for word in line]
        # 각 토큰에서 구두점을 제거한다.
        line = [word.translate(table) for word in line]
        # 각 토큰에서 프린트할 수 없는 문자를 제거한다.
        line = [re_print.sub('', w) for w in line]
        # 숫자를 포함한 토큰을 제거한다.
        line = [word for word in line if word.isalpha()]
        # 문자열로 저장한다.
        cleaned.append(' '.join(line))
    return cleaned

```

데이터셋 준비 함수를 정의했으니 먼저 영어 데이터부터 불러온 후 정리한다.

```

# 영어 데이터를 로드한다.
filename = 'europarl-v7.fr-en.en'
doc = load_doc(filename)
sentences = to_sentences(doc)
minlen, maxlen = sentence_lengths(sentences)
print('English data: sentences=%d, min=%d, max=%d' % (len(sentences),
minlen, maxlen))
cleanf=clean_lines(sentences)

```

이제 데이터셋이 깔끔해졌으니, pickle로 직렬화된 파일 English.pkl에 덤프한다.

```

filename = 'English.pkl'
outfile = open(filename,'wb')
pickle.dump(cleanf,outfile)
outfile.close()
print(filename," saved")

```

통계치가 출력되고 English.pkl이 저장되었는지 확인한다.

```

English data: sentences=2007723, min=0, max=668
English.pkl saved

```

이제 프랑스어 데이터도 동일한 프로세스를 반복하여 직렬화된 파일 French.pkl에 덤프한다.

```

# 프랑스어 데이터를 로드한다.
filename = 'europarl-v7.fr-en.fr'
doc = load_doc(filename)
sentences = to_sentences(doc)
minlen, maxlen = sentence_lengths(sentences)
print('French data: sentences=%d, min=%d, max=%d' % (len(sentences),
minlen, maxlen))
cleanf=clean_lines(sentences)
filename = 'French.pkl'
outfile = open(filename,'wb')
pickle.dump(cleanf,outfile)
outfile.close()
print(filename," saved")

```

프랑스어 데이터셋의 통계치가 출력되고 French.pkl이 저장되었는지 확인한다.

```

French data: sentences=2007723, min=0, max=693
French.pkl saved

```

전처리가 완료되었다. 하지만 아직 데이터셋에 노이즈 토큰이 있을 수 있으니 확인해야 한다.

## 6.2.2 데이터셋 전처리 마무리하기

이제 `read.py`와 같은 디렉터리에서 `read_clean.py`를 연다. 이전 섹션에서 정리한 데이터셋을 불러온 후 전처리가 마무리되면 저장하는 함수를 정의한다.

```
from pickle import load
from pickle import dump
from collections import Counter
# 정제된 데이터셋을 로드한다.
def load_clean_sentences(filename):
    return load(open(filename, 'rb'))
# 정제된 문장 리스트를 저장한다.
def save_clean_sentences(sentences, filename):
    dump(sentences, open(filename, 'wb'))
    print('Saved: %s' % filename)
```

이제 어휘 카운터(vocabulary counter)를 생성하는 함수를 정의한다. 분석할 시퀀스에서 단어가 몇 번이나 사용되는지 아는 것은 중요하다. 예를 들어, 200만 줄로 구성된 데이터셋에서 한 번만 사용된 단어를 학습하기 위해 귀중한 GPU 리소스를 사용하는 것은 에너지 낭비다. 카운터를 정의해 보자.

```
# 모든 단어에 대한 빈도수 테이블을 생성한다.
def to_vocab(lines):
    vocab = Counter()
    for line in lines:
        tokens = line.split()
        vocab.update(tokens)
    return vocab
```

어휘 카운터는 `min_occurrence`보다 빈도가 낮은 단어를 감지한다.

```
# 기준값 미만의 빈도수를 가진 단어들을 제거한다.
def trim_vocab(vocab, min_occurrence):
    tokens = [k for k,c in vocab.items() if c >= min_occurrence]
    return set(tokens)
```

`min_occurrence=5` 이하인 단어를 제거하여 모델의 학습 시간을 낭비하지 않도록 한다.

이제 OOV(Out-Of-Vocabulary) 단어를 처리한다. OOV 단어는 철자가 틀린 단어, 약어 또는 표준 어휘 표현에 맞지 않는 경우이다. 맞춤법 자동 교정 기능을 사용할 수도 있지만 모든 문제가 해결되지는 않는다. OOV 단어는 unk(알 수 없음) 토큰으로 대체한다.

```
# 모든 OOV 단어를 unk로 표시한다.
def update_dataset(lines, vocab):
    new_lines = list()
    for line in lines:
        new_tokens = list()
        for token in line.split():
            if token in vocab:
                new_tokens.append(token)
            else:
                new_tokens.append('unk')
        new_line = ' '.join(new_tokens)
        new_lines.append(new_line)
    return new_lines
```

이제 영어 데이터셋으로 함수를 실행하고 결과를 저장 및 출력한다.

```
# 영어 데이터셋을 로드한다.
filename = 'English.pkl'
lines = load_clean_sentences(filename)
# 사전을 추출한다.
vocab = to_vocab(lines)
print('English Vocabulary: %d' % len(vocab))
# 사전을 줄인다.
vocab = trim_vocab(vocab, 5)
print('New English Vocabulary: %d' % len(vocab)) # 사전에 없는 단어를 표시한다.
lines = update_dataset(lines, vocab)
# 업데이트된 데이터셋을 저장한다.
filename = 'english_vocab.pkl'
save_clean_sentences(lines, filename)
# 결과를 확인한다.
for i in range(20):
    print("line",i,":",lines[i])
```

먼저 원래 어휘 개수와 추출된 어휘 개수를 보여준다.

```
English Vocabulary: 105357
New English Vocabulary: 41746
Saved: english_vocab.pkl
```

전처리된 데이터셋이 저장되고, 다음 결과가 출력된다.

```
line 0 : resumption of the session
line 1 : i declare resumed the session of the european parliament
adjourned on friday december and i would like once again to wish you a
happy new year in the hope that you enjoyed a pleasant festive period
line 2 : although, as you will have seen, the dreaded millennium bug
failed to materialise still the people in a number of countries suffered a
series of natural disasters that truly were dreadful
line 3 : you have requested a debate on this subject in the course of the
next few days during this partsession
```

이제 프랑스어 데이터셋으로 함수를 실행하고 결과를 저장 및 출력한다.

```
# 프랑스어 데이터를 로드한다.
filename = 'French.pkl'
lines = load_clean_sentences(filename)
# 사전을 추출한다.
vocab = to_vocab(lines)
print('French Vocabulary: %d' % len(vocab))
# 사전을 줄인다.
vocab = trim_vocab(vocab, 5)
print('New French Vocabulary: %d' % len(vocab)) # 사전에 없는 단어를 표시한다.
lines = update_dataset(lines, vocab)
# 업데이트된 데이터셋을 저장한다.
filename = 'french_vocab.pkl'
save_clean_sentences(lines, filename)
# 결과를 확인한다.
for i in range(20):
    print("line",i,":",lines[i])
```

먼저 원래 어휘 개수와 추출된 어휘 개수를 보여준다.

```
French Vocabulary: 141642
New French Vocabulary: 58800
Saved: french_vocab.pkl
```

전처리된 데이터셋이 저장되고, 다음 결과가 출력된다.

```
line 0 : reprise de la session
line 1 : je declare reprise la session du parlement europeen qui avait ete
interrompue le vendredi decembre dernier et je vous renouvelle tous mes
vux en esperant que vous avez passe de bonnes vacances
line 2 : comme vous avez pu le constater le grand bogue de lan ne sest pas
produit en revanche les citoyens dun certain nombre de nos pays ont ete
victimes de catastrophes naturelles qui ont vraiment ete terribles
line 3 : vous avez souhaite un debat a ce sujet dans les prochains jours
au cours de cette periode de session
```

이번 섹션에서 우리는 원시 데이터를 어떻게 전처리하는지 살펴보았고, 이제 데이터셋을 트랜스포머에 연결하여 학습할 준비가 되었다.

프랑스어 데이터셋의 각 줄은 번역할 문장이다. 영어 데이터셋의 각 줄은 기계 번역 모델을 위한 레퍼런스이다. 기계 번역 모델은 레퍼런스와 일치하는 영어 번역 후보를 생성한다.

BLEU는 기계 번역 모델이 생성한 번역 후보를 평가하는 방법이다.

## 6.3

# BLEU로 기계 번역 평가하기

2002년 파피네니(Papineni et al.)는 인간의 번역을 평가하는 효율적인 방법을 고안했다. 인간 기준 성능을 정의하기는 어려웠지만, 인간의 번역과 기계의 번역을 한 단어 한 단어 비교하면 효율적이라는 것을 깨달았다.

파피네니(Papineni et al.)는 이 방법을 BLEU(Bilingual Evaluation Understudy Score)라고 이름 지었다.

NLTK(Natural Language Toolkit)를 사용하여 BLEU를 구현한다.

[http://www.nltk.org/api/nltk.translate.html#nltk.translate.bleu\\_score.sentence\\_bleu](http://www.nltk.org/api/nltk.translate.html#nltk.translate.bleu_score.sentence_bleu)

먼저 기하학적으로 평가해 보자.

### 6.3.1 기하학적 평가

BLEU 방법은 후보 문장의 일부를 레퍼런스 문장과 비교한다.

깃허브 저장소에서 이번 챕터 디렉터리에 있는 BLEU.py를 연다.

프로그램은 nltk 모듈을 가져온다.

```
from nltk.translate.bleu_score import sentence_bleu
from nltk.translate.bleu_score import SmoothingFunction
```

그런 다음 기계 번역 모델이 생성한 후보 문장과 데이터셋의 실제 레퍼런스 문장을 비교하는 시뮬레이션을 수행한다. 동일한 문장이 여러 번역가에 의해 다른 방식으로 번역됐을 수 있으므로 효율적인 평가 전략을 찾기 어렵다는 점을 기억하자.

프로그램은 한 번에 여러 레퍼런스 문장을 평가할 수 있다.

### #예제 1

```
reference = [['the', 'cat', 'likes', 'milk'], ['cat', 'likes', 'milk']]
candidate = ['the', 'cat', 'likes', 'milk']
score = sentence_bleu(reference, candidate)
print('Example 1', score)
```

### #예제 2

```
reference = [['the', 'cat', 'likes', 'milk']]
candidate = ['the', 'cat', 'likes', 'milk']
score = sentence_bleu(reference, candidate)
print('Example 2', score)
```

두 예시의 점수는 모두 1점이다.<sup>15</sup>

```
Example 1 1.0
```

```
Example 2 1.0
```

후보 문장 C에 대한 레퍼런스 문장 R과 C(N)에서 찾은 정답 토큰 개수에 대한 평가 P는 다음 기하학적 함수로 나타낼 수 있다.<sup>16</sup>

$$P(N, C, R) = \prod_{n=1}^N p_n$$

3-그램 중복을 찾는 경우 이 기하학적 접근 방식은 엄격하다. 다음 예시를 보자.

### #예제 3

```
reference = [['the', 'cat', 'likes', 'milk']]
candidate = ['the', 'cat', 'enjoys', 'milk']
score = sentence_bleu(reference, candidate)
print('Example 3', score)
```

<sup>15</sup> 역주. 후보 문장이 레퍼런스 문장 중 하나에만 포함되어도 정답이다.

<sup>16</sup> 역주.  $p_n$ 은 레퍼런스와 후보의 n-그램(gram) 토큰마다 개수를 구한다. 그리고 두 값 중 작은 값을 취한 후 더한 값을 후보 n-그램 토큰 개수로 나눈 값이다.

3-그램 중복을 찾는 경우의 출력은 심각하다.<sup>17</sup>

```
Warning (from warnings module):
  File
"C:\Users\Denis\AppData\Local\Programs\Python\Python37\lib\site-
packages\
nltk\translate\bleu_score.py", line 490
  warnings.warn(msg)
UserWarning:
Corpus/Sentence contains 0 counts of 3-gram overlaps.
BLEU scores might be undesirable; use SmoothingFunction().
Example 3 0.7071067311865475
```

인간이라면 아마 만점인 1점을 줄 것이다. `sentence_bleu`의 파라미터를 추가해 평가 방식에 변경을 줄 수 있지만 여전히 평가가 매우 엄격하다.

앞의 코드에서 출력된 경고 메시지는 다음 챕터에서 살펴보자.

이 프로그램은 확률적인 프로세스이므로 버전과 각 실행마다 경고 메시지가 다를 수 있다.

파피네니는 수정된 유니그램(unigram) 접근법을 고안했다. 이 아이디어는 레퍼런스 문장에서 단어 등장 횟수를 세고 후보 문장에서 단어가 과도하게 평가되지 않도록 하는 것이다.

파피네니가 설명한 다음 예시를 살펴보자.

Reference 1: The cat is on the mat.

Reference 2: There is a cat on the mat.

이제 다음 후보 문장을 살펴보자.

Candidate: the the the the the the the

이제 Reference 1의 단어("the" 2번 등장) 중 후보 문장에 있는 단어의 개수(동일한 "the"가 7번 등장)를 찾는다.

<sup>17</sup> 역주. n-그램에서 일치하는 값이 없는 경우 값이 0이다. 이를 score에 반영할 것인지는 계속 변화하는 nltk 버전에 따라 달라질 수 있다.

표준 유니그램 정밀도는 7/7이다. 수정된 유니그램 정밀도는 2/7이다.

BLEU 함수가 출력한 경고 메시지가 제안했던 스무딩(smoothing) 기법을 추가해 보자.

### 6.3.2 스무딩 기법 적용하기

첸(Chen)과 체리(Cherry)는 표준 BLEU 기법의 기하학적 평가 방식을 개선한 스무딩 기법을 고안했다.

라벨 스무딩은 학습 단계에서 트랜스포머 모델의 성능을 개선하는 매우 효율적인 방법이다. 이 방법은 펄플렉시티(perplexity)<sup>18</sup>에 부정적인 영향을 주지만 모델을 더 불확실하게 만들어서 결과적으로 정확도에 긍정적인 영향을 주게 된다.

예를 들어 다음 시퀀스에서 마스킹된 단어가 무엇인지 예측해야 한다고 가정해 보자.

The cat [mask] milk

출력이 소프트맥스(softmax) 벡터로 나온다고 상상해 보자.

```
candidate_words=[drinks, likes, enjoys, appreciates]
candidate_softmax=[0.7, 0.1, 0.1, 0.1]
candidate_one_hot=[1, 0, 0, 0]
```

‘drinks’만 정답(1), 나머지는 완전한 오답(0)으로 간주하는 것은 꽤나 잔인할 수 있다. 라벨 스무딩은 엡실론 =  $\epsilon$ 을 도입하여 시스템을 보다 유연하게 만들 수 있다.

candidate\_softmax의 크기는  $k=4$ 이다.

라벨 스무딩을 위해  $\epsilon$ 가 0.25인 예시를 살펴보자. 먼저 candidate\_one\_hot의 값을  $1-\epsilon$ 만큼 줄인다. 그리고 0 값을  $0 + \epsilon/(k-1)$ 만큼 증가시킨다. 이를 적용하면 다음 결과를 얻을 수 있다.

candidate\_smoothed=[0.75, 0.083, 0.083, 0.083], 이제 변환 결과가 유연하게 나올 수 있게 만들었다.

트랜스포머는 라벨 스무딩을 변형한 첸체리(chencherry) 스무딩을 사용한다.

18 역주. 언어 모델을 평가하는 지표 중 하나로, 모델이 예측할 때 얼마나 확신하는지를 나타낸다.

### 6.3.2.1 첸체리 스무딩 기법

첸과 체리(2014)는 0이 아닌 값에  $\epsilon$ 를 추가하여 후보 평가를 스무딩하는 흥미로운 방법을 소개했다. 여러 가지 첸체리(복싱 첸(Boxing Chen) + 콜린 체리(Colin Cherry)) 방법이 있으니 다음 링크를 참고하자. <https://www.nltk.org/api/nltk.translate.html>.

먼저 스무딩 기법을 이용하여 프랑스어-영어 예제를 평가해 보자.

#### #예제 4

```
reference = [['je', 'vous', 'invite', 'a', 'vous', 'lever', 'pour', 'cette',  
'minute', 'de', 'silence']]  
candidate = ['levez', 'vous', 'svp', 'pour', 'cette', 'minute', 'de',  
'silence']  
score = sentence_bleu(reference, candidate)  
print("without soothing score", score)
```

후보 문장이 인간이 용납할 수준이긴 하지만 출력 점수는 낮다.

```
without smoothing score 0.37188004246466494
```

이제 평가에 스무딩을 추가해 보자.

```
chencherry = SmoothingFunction()  
r1=list('je vous invite a vous lever pour cette minute de silence')  
candidate=list('levez vous svp pour cette minute de silence')  
#sentence_bleu([reference1, reference2, reference3], hypothesis2, smoothing_  
function=chencherry.method1)  
print("with soothing score", sentence_bleu([r1], candidate, smoothing_  
function=chencherry.method1))
```

점수가 아직 인간이 받아들일 수 있는 수준에는 미치지 못한다.

```
with smoothing score 0.6194291765462159
```

지금까지 데이터셋이 전처리되는 방법과 BLEU가 기계 번역을 평가하는 방법을 살펴보았다.

## 6.4

# 구글 번역으로 번역하기

구글 번역은 번역에 바로 사용할 수 있는 인터페이스를 제공한다. <https://translate.google.com/> 구글은 번역 알고리즘에 트랜스포머 인코더를 점진적으로 도입하고 있다. 다음 섹션에서 구글 트렉스로 번역용 트랜스포머 모델을 구현해 보자.

구현하는데 AI 전문가가 전혀 필요하지 않을 수 있다.

구글 번역에 앞의 챕터에서 분석한 문장 `Levez-vous svp pour cette minute de silence`를 입력하면 실시간으로 영어로 번역한 결과를 얻을 수 있다.



그림 6.2 구글 번역

내용을 확인해 보면 번역이 정확하다는 것을 알 수 있다.

4차 산업 시대에서도 번역 작업에 AI 전문가가 필요할까, 아니면 단순히 웹 인터페이스 개발자가 필요할까?

구글은 번역에 필요한 모든 서비스를 구글 번역 플랫폼에서 제공한다. <https://cloud.google.com/translate>

- **번역 API:** 웹 개발자를 위한 인터페이스
- 스트리밍 콘텐츠 번역을 위한 미디어 번역 API
- 특정 도메인에 맞게 모델 학습 커스터마이징이 가능한 AutoML 번역 서비스

구글 번역 프로젝트에는 인터페이스를 위한 웹 개발자, 주제별 전문가(SME, Subject Matter Expert), 언어학자가 필요하다. 하지만 AI 전문가가 필수 조건은 아니다.

4차 산업 시대는 서비스로서의 AI(AI as a service)로 향하고 있다. 그렇다면 왜 트랜스포머로 AI 개발하는 방법을 공부해야 할까? 4차 산업 시대에서 AI 전문가가 되어야 하는 두 가지 중요한 이유가 있다.

실제 AI 프로젝트에서 간혹 예상치 못한 문제에 부딪히게 된다. 예를 들어, 구글 번역은 아무리 많은 노력을 기울여도 특정 요구 사항에 맞지 않을 수 있다. 이 경우 구글 트렉스가 유용하다!

구글 트렉스를 사용하려면 AI 개발자가 필요하다!

4차 산업 시대는 모든 것을 모든 것에 연결한다. AI 프로젝트는 순조롭게 진행될 때도 있지만, 복잡한 문제를 해결하기 위해 AI 전문 지식이 필요할 때도 있다. **14장, 블랙박스 트랜스포머 모델 해석하기**에서 구글 번역을 구현할 때 AI 개발이 어떻게 필요한지 살펴본다.

이제 트렉스로 번역 모델을 구현할 준비가 되었다.

## 6.5

## 트랙스로 번역하기

구글 브레인은 딥러닝 개발을 더 쉽게 하기 위해 T2T(Tensor2Tensor)를 개발했다. T2T는 텐서플로우의 확장판으로, 다양한 트랜스포머 예제가 포함된 딥러닝 모델 라이브러리가 있다.

T2T에 이어 구글 브레인은 엔드-투-엔드 딥러닝 라이브러리 트랙스를 개발했다. 트랙스에는 번역에 적용할 수 있는 트랜스포머 모델이 있다. 현재 구글 브레인 팀이 트랙스를 유지 및 관리하고 있다.

이번 섹션에서는 트랜스포머의 성능을 설명하기 위해 2017년 바스와니가 설명한 영어-독일어 문제 해결을 위한 함수에 초점을 맞춘다.

트랜스포머 아키텍처가 언어에 구애받지 않는다는 것을 보여주기 위해 전처리된 영어-독일어 데이터 셋을 사용한다.

Trax\_Translation.ipynb를 열고, 필요한 모듈을 설치하는 것부터 시작하자.

### 6.5.1 트랙스 설치하기

구글 브레인은 트랙스를 쉽게 설치하고 실행할 수 있게 만들었다. 한 줄로 트랙스를 설치하고 기본 라이브러리도 가져온다.

```
#@title 트랙스 설치하기
import os
import numpy as np
!pip install -q -U trax
import trax
```

매우 간단하다!

이제 트랜스포머 모델을 만들어 보자.

## 6.5.2 오리지널 트랜스포머 모델 생성하기

2장, 트랜스포머 모델 아키텍처 살펴보기에서 설명한 대로 오리지널 트랜스포머 모델을 생성한다.

트랙스 함수는 코드 몇 줄 만으로 사전 학습된 모델 설정을 검색한다.

```
#@title 오리지널 트랜스포머 모델 생성하기
# 사전 학습된 모델 설정 파일 경로: gs://trax-ml/models/translation/ende_wmt32k.gin
model = trax.models.Transformer(
    input_vocab_size=33300,
    d_model=512, d_ff=2048,
    n_heads=8, n_encoder_layers=6, n_decoder_layers=6,
    max_len=2048, mode='predict')
```

모델은 인코더와 디코더 스택이 있는 트랜스포머이다. 각 스택은 총 6개와 헤드 8개를 가진다. 오리지널 트랜스포머 아키텍처와 같이 `d_model=512`이다.

트랜스포머를 실행하려면 사전 학습된 가중치가 필요하다.

## 6.5.3 사전 학습된 가중치로 모델 초기화하기

트랜스포머의 지능은 사전 학습된 가중치에 있다. 가중치는 트랜스포머의 언어적 표현으로 이루어져 있다. 가중치는 여러 파라미터로 표현할 수 있으며, 이 파라미터는 일종의 기계 지능 IQ를 생성한다.

가중치를 초기화하여 모델에 생명을 불어넣어 보자.

```
#@title 사전 학습된 가중치로 모델 초기화하기
model.init_from_file('gs://trax-ml/models/translation/ende_wmt32k.pkl.gz',
                    weights_only=True)
```

이제 기계 설정과 지능을 실행시킬 준비가 됐다. 문장을 토큰화해 보자.

## 6.5.4 문장 토큰화하기

문장을 토큰화할 준비가 되었다. 노트북은 트랙스에서 전처리된 어휘를 사용한다. 전처리 방법은 이 챕터의 WMT 데이터셋 전처리하기 섹션에서 설명한 것과 비슷하다.

이제 문장을 토큰화해 보자.

```
#@title 문장 토큰화하기
sentence = 'I am only a machine but I have machine intelligence.'
tokenized = list(trax.datatokenize(iter([sentence]), # 스트림으로 동작한다.
                        vocab_dir='gs://trax-ml/vocabs/',
                        vocab_file='ende_32k.subword'))[0]
```

이제 문장을 디코딩하고 번역을 생성해 보자.

## 6.5.5 트랜스포머로 디코딩하기

트랜스포머는 문장을 영어로 인코딩하고 이를 독일어로 디코딩한다. 모델과 가중치가 기능을 구성한다.

트랙스의 디코딩 기능은 직관적이다.

```
#@title 트랜스포머로 디코딩하기
tokenized = tokenized[None, :] # 배치 추가
tokenized_translation = trax.supervised.decoding.autoregressive_sample(
    model, tokenized, temperature=0.0) # 온도(temperature) 파라미터가 높을수록 더
    다양한 결과를 얻을 수 있다
```

이번 챕터의 기계 번역 정의하기 섹션에서 설명한 대로 온도(temperature) 파라미터가 높을수록 인간 번역가와 마찬가지로 다양한 결과를 얻을 수 있다는 점에 유의하자.

마지막으로 토큰화를 해제하고 번역을 출력해 보자.

## 6.5.6 토큰화 해제 및 번역 출력하기

구글 브레인은 트랙스를 통해 대세(mainstream)이자 혁신적이며 직관적인 트랜스포머를 구현했다.

이제 프로그램이 토큰을 문장으로 변환하고 번역을 출력한다.

**#@title** 토큰화 해제 및 번역 출력하기

```
tokenized_translation = tokenized_translation[0][-1] # 배치와 EOS를 제거한다.
translation = traxdata.detokenize(tokenized_translation,
                                  vocab_dir='gs://trax-ml/vocabs/',
                                  vocab_file='ende_32k.subword')
print("The sentence:", sentence)
print("The translation:", translation)
```

번역 결과가 매우 인상적이다.

```
The sentence: I am only a machine but I have machine intelligence.(나는 기계에 불과
하지만 기계 지능(machine intelligence)을 가지고 있다.)
The translation: Ich bin nur eine Maschine, aber ich habe Maschinenübersicht.
(나는 기계에 불과하지만 기계 지능(Maschinenübersicht)을 가지고 있다.)
```

트랜스포머는 machine intelligence를 Maschinenübersicht(기계 개요)로 번역했다.

Maschinenübersicht를 Maschin(기계) + übersicht(지능)로 나뉘고, übersichtf를 분해하면 다음과 같다.

- über는 “위에”라는 뜻이다.
- sicht는 “시력” 또는 “보기”를 의미한다.

트랜스포머가 기계임에도 시각적인 지각 능력이 있다는 것을 보여준다. 트랜스포머로 기계의 지능은 성장하고 있지만, 이는 인간의 지능과는 다르다. 기계는 기계 나름의 지능을 가지고 언어를 학습한다.

이것으로 구글 트랙스에 대한 실험을 마친다.

이번 장에서는 오리지널 트랜스포머의 세 가지 필수 요소를 살펴보았다.

먼저 기계 번역을 정의하는 것으로 시작했다. 인간의 번역 결과로 매우 높은 기준점을 설정했다. 영어-프랑스어 및 영어-독일어 번역에는 해결해야 할 문제가 많다는 것을 알았다. 그리고 트랜스포머는 이러한 문제를 해결하고 최고 수준의 BLEU 기록을 세웠다.

그런 다음 유럽 의회의 WMT 프랑스어-영어 데이터셋을 전처리했다. 데이터셋을 여러 문장으로 변환하고 정리했다. 그리고 일정 횟수 이하로 등장하는 단어를 제거하여 데이터셋의 크기를 줄였다.

기계 번역 NLP 모델은 동일한 평가 방법을 요구한다. WMT 데이터셋으로 모델을 학습하려면 BLEU 평가가 필요하다. 기하학적 평가는 번역 점수를 매기는 데 좋지만 수정된 BLEU에도 한계가 있다는 것을 알았다. 이에 따라 BLEU를 개선하기 위해 스무딩 기법을 추가했다.

구글 번역은 표준 번역 API, 미디어 스트리밍 API, 커스텀 AutoML 모델 학습 서비스를 제공한다. 구글 번역 API 구현 프로젝트가 순조롭게 진행된다면 AI 개발이 필요하지 않을 수 있지만, 그렇지 않다면 예전처럼 직접 개발해야 한다!

구글 브레인의 엔드-투-엔드 딥러닝 라이브러리인 트랙스를 사용하여 영어-독일어 번역 트랜스포머를 구현했다.

지금까지 아키텍처, 사전 학습, 학습, 데이터셋 전처리, 평가 방법 등 트랜스포머를 구축하기 위한 주요 구성 요소에 대해 살펴보았다.

다음 장인 **7장, GPT-3 엔진을 사용한 초인간 트랜스포머 등장**에서 지금까지 살펴본 구성 요소로 트랜스포머를 구현하는 놀라운 방법을 알아보자.

## 6.7 문제

01. 기계 번역은 이제 인간의 수준을 넘어섰다. 참 / 거짓
02. 기계 번역에는 대규모 데이터셋이 필요하다. 참 / 거짓
03. 동일한 데이터셋을 사용하여 트랜스포머 모델을 비교할 필요가 없다. 참 / 거짓
04. BLEU는 파란색을 뜻하는 프랑스어이며 NLP 성능(metric)의 약어이다. 참 / 거짓
05. 스무딩 기법은 BERT의 성능을 향상한다. 참 / 거짓
06. 영어-독일어 기계 번역은 독일어-영어와 동일하다. 참 / 거짓
07. 오리지널 트랜스포머의 멀티-헤드 어텐션 서브 층의 헤드는 2개다. 참 / 거짓
08. 오리지널 트랜스포머 인코더는 층이 6개다. 참 / 거짓
09. 오리지널 트랜스포머 인코더는 층이 6개가 있지만 디코더에는 2개뿐이다. 참 / 거짓
10. 디코더 없이 트랜스포머를 학습할 수 있다. 참 / 거짓

- 영어-독일어의 BLEU 점수를 사용한 논문과 코드: <https://paperswithcode.com/sota/machine-translation-on-wmt2014-english-german>
- 2014 WMT(Workshop on Machine Translation) 데이터셋: <https://www.statmt.org/wmt14/translation-task.html>
- 유럽 의회 회의록 병렬말뭉치 1996-2011, 프랑스어-영어: <https://www.statmt.org/europarl/v7/fr-en.tgz>
- 제이슨 브라운리(Jason Brownlee), Ph.D., 기계번역을 위한 프랑스어-영어 데이터를 준비하는 방법: <https://machinelearningmastery.com/prepare-french-english-dataset-machine-translation/>
- Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu, 2002, BLEU: a Method for Automatic Evaluation of Machine Translation: <https://aclanthology.org/P02-1040.pdf>
- 제이슨 브라운리(Jason Brownlee), Ph.D., 파이썬으로 텍스트에 대한 BLEU 점수를 계산하는 친절한 소개: <https://machinelearningmastery.com/calculate-bleu-score-for-text-python/>
- Boxing Chen and Colin Cherry, 2014, A Systematic Comparison of Smoothing Techniques for Sentence-Level BLEU: <http://acl2014.org/acl2014/W14-33/pdf/W14-3346.pdf>
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin, 2017, Attention Is All You Need: <https://arxiv.org/abs/1706.03762>
- 트랙스 저장소: <https://github.com/google/trax>
- 트랙스 튜토리얼: <https://trax-ml.readthedocs.io/en/latest/>

# 07장

---

## GPT-3 엔진을 사용한 초인간 트랜스포머 등장

---

- 7.1 GPT-3 트랜스포머 모델을 사용한 초인간 NLP
- 7.2 OpenAI GPT 트랜스포머 모델의 아키텍처
- 7.3 GPT-2를 사용한 일반 텍스트 완성
- 7.4 커스텀 GPT-2 언어 모델 학습
- 7.5 OpenAI GPT-3 작업 실행하기
- 7.6 GPT-2와 GPT-3의 출력 비교하기
- 7.7 GPT-3 미세 조정하기
- 7.8 4차 산업혁명 AI 전문가의 역할
- 7.9 정리하기
- 7.10 문제
- 7.11 참고 문헌

2020년 브라운(Brown et al.)은 커먼 크롤(Common Crawl) 데이터에서 추출한 바이트 쌍 인코딩 토큰 4,000억 개와 같은 방대한 데이터셋을 사용해 학습한 파라미터 1,750억 개로 구성된 OpenAI GPT-3 모델 학습을 소개했다. OpenAI는 CPU 285,000개와 GPU 10,000개를 갖춘 마이크로소프트 애저(Microsoft Azure) 슈퍼컴퓨터에서 학습시켰다.

브라운(Brown et al.)은 OpenAI의 GPT-3 엔진과 슈퍼컴퓨터의 기계 지능 덕분에 제로샷(zero-shot) 실험을 할 수 있었다. 제로샷이란, 파라미터를 추가로 학습시키지 않고 학습된 모델을 다운스트림 작업에 사용하는 것이다. 실험 목표는 학습된 모델이 학습하지 않은 작업까지 수행하는 멀티태스킹(multi-task) API를 배포하는 것이다.

초인간(superhuman) 클라우드 AI 엔진의 시대가 열렸다. OpenAI의 API는 높은 수준의 소프트웨어 기술이나 AI 지식을 요구하지 않는다. 왜 “초인간”이라는 용어를 사용했는지 궁금할 것이다. GPT-3 엔진은 인간만큼 많은 작업을 수행할 수 있다는 걸 독자는 알게 될 것이다. 우선, GPT 모델이 어떻게 구축되고 실행되는지를 이해해보자.

이 장에서는 먼저 트랜스포머 모델 크기와 아키텍처의 진화에 대해 살펴본다. 그리고 미세 조정을 거의 하지 않고 학습된 트랜스포머 모델을 다운스트림 작업에 사용하는 제로샷 문제를 살펴본다. 추가로, GPT 트랜스포머 모델의 혁신적인 아키텍처를 살펴본다. OpenAI는 학습된 특정 모델 버전을 엔진이라는 이름으로 제공한다.

이 챕터에서는 OpenAI의 리포지토리에 있는 파라미터 3억 4천5백만 개로 구성된 GPT-2 트랜스포머를 사용할 것이다. GPT 모델을 이해하기 위해선 직접 코드를 작성해봐야 한다. 모델과 상호 작용해 일반적인 조건문으로 텍스트 완성(text completion)을 생성할 것이다.

이어서 1억 7천만 개의 파라미터로 커스터마이징된 GPT-2 모델을 사용할 것이다. **4장, RoBERTa 모델 처음부터 사전 학습하기**에서 RoBERTa 모델 학습에 사용한 칸트 데이터셋을 토큰화할 것이다.

그런 다음, 이 챕터에서는 데이터 과학자, 인공지능 전문가 또는 숙련된 개발자가 아니어도 GPT-3 엔진을 사용하는 방법을 살펴본다. 하지만 그렇다고 해서 데이터 과학자나 AI 전문가가 나중에 필요하지 않다는 의미는 아니다.

GPT-3 엔진은 때때로 미세 조정이 필요할 수 있다. 구글 코랩(Google Colab) 노트북을 사용해 GPT-3 Ada 엔진을 미세 조정해 볼 것이다.

4차 산업혁명 AI 전문가의 새로운 사고방식과 스킬셋으로 이 챕터를 마무리한다.

이 장을 읽으면, 독자는 GPT 모델이 어떻게 구축되는지와 원활한(seamless) GPT-3 API를 사용하는 방법을 알게 될 것이다. 2020년대에 4차 산업혁명 AI 전문가가 해낼 수 있는 보람찬 작업은 무엇이 있는지 알게 될 것이다.

이 장에서는 다음 주제를 다룬다.

- GPT-3 모델 시작하기
- OpenAI GPT 모델 아키텍처
- 제로샷 트랜스포머 모델 정의하기
- 퓨샷(few-shots)에서 원샷(one-shot)으로 가는 길
- 인간에 가까운 GPT-2 텍스트 완성 모델 구축하기
- 3억 4천5백만 개 파라미터 모델 구현 및 실행하기
- 표준 모델을 사용하여 GPT-2와 상호작용하기
- 1억 1천7백만 개 파라미터 GPT-2 언어 모델 학습하기
- 특정 커스텀 데이터셋 불러오기
- 사용자 지정 데이터셋 인코딩하기
- 모델 컨디셔닝하기
- 특정 텍스트 완성 작업용 GPT-2 모델 컨디셔닝하기
- GPT-3 모델 미세조정하기
- 4차 산업혁명 AI 전문가의 역할

GPT-3 트랜스포머 모델을 살펴보자.

## 7.1

# GPT-3 트랜스포머 모델을 사용한 초인간 NLP

GPT-3는 GPT-2 아키텍처에 기반한다. 그러나 완전히 학습된 GPT-3 트랜스포머는 파운데이션 모델이다.<sup>19</sup> 파운데이션 모델은 학습하지 않은 많은 작업을 수행할 수 있다. 완전히 학습된 GPT-3 모델은 모든 NLP 작업과 프로그래밍 작업에도 사용할 수 있다.

### NOTE

GPT-3는 파운데이션 모델 중 몇 안되는 완전히 학습된 트랜스포머 모델이다. GPT-3는 더 강력한 OpenAI 모델을 낳을 것이다. 구글은 슈퍼컴퓨터에 학습한 구글 BERT 그 이상의 파운데이션 모델을 만들 것이다. 파운데이션 모델은 AI에 대한 새로운 사고방식을 대표한다.

머지않아 기업들은 OpenAI API 같은 API를 사용해 NLP 프로젝트를 시작하는 데 데이터 과학자나 AI 전문가가 필요하지 않다는 것을 깨달을 것이다.

다른 도구 사용에 힘을 들이는 대신, 세계에서 가장 강력한 슈퍼컴퓨터 중 하나에서 학습한 가장 효율적인 트랜스포머 모델 중 하나를 사용하는 OpenAI API를 사용할 수 있다.

구글이나 OpenAI와 같이 막대한 자금과 세계 최고 연구팀만이 설계할 수 있는 API가 존재한다면 굳이 도구를 개발하거나 라이브러리를 다운로드해서 사용할 이유가 있을까?

이 질문에 대한 답은 간단하다. GPT-3 엔진을 사용하는 것은 레이싱카 시동을 거는 것만큼 간단하다. 하지만 레이싱카는 운전하기 위해 수개월의 훈련이 필요하다. GPT-3 엔진은 강력한 인공지능 레이싱카다. 클릭 몇 번으로 실행할 수 있다. 하지만 GPT-3 엔진을 운전하기 위해서는 이 책의 시작부터 지금까지 습득한 지식과 다음 장의 지식이 필요하다.

초인간 NLP 모델의 시대에 개발자, AI 전문가, 데이터 과학자가 서있을 자리가 어디인지 파악하려면 먼저 GPT 모델 아키텍처를 이해해야 한다.

<sup>19</sup> 역주. 1장 트랜스포머란 무엇인가?에서 파운데이션 모델에 대해 다루었다.

## 7.2

# OpenAI GPT 트랜스포머 모델의 아키텍처

트랜스포머는 2017년 말부터 2020년 상반기까지 3년이 채 안 되는 기간 동안 학습, 미세 조정, 최종적으로 제로샷 모델에 도달했다. 제로샷 GPT-3 트랜스포머 모델은 미세 조정이 필요하지 않다. 학습된 모델 파라미터는 다운스트림 멀티태스크에 대해 업데이트되지 않으므로 NLP/NLU 작업의 새로운 시대가 열렸다.

이 섹션에서는 먼저 GPT 모델을 설계한 OpenAI 팀의 동기에 대해 알아본다. 제로샷 모델을 미세 조정하는 것부터 시작한다. 그런 다음 놀라운 결과를 주는 텍스트 완성을 생성하기 위해 트랜스포머 모델을 조절하는 방법을 살펴본다.

먼저 OpenAI 팀 구축 과정을 살펴보자.

### 7.2.1 10억 개 파라미터를 가진 트랜스포머 모델 부상

NLP 작업을 위해 학습된 소규모 모델에서 미세 조정이 거의 또는 전혀 필요 없는 모델까지 트랜스포머의 발전 속도는 놀라울 정도다.

2017년 바스와니(Vaswani et al.)는 BLEU(Bilingual Evaluation Understudy Score) 작업에서 CNN과 RNN을 능가하는 트랜스포머를 소개했다. 2018년 래드포드(Radford et al.)는 미세 조정을 통해 다운스트림 작업을 수행할 수 있는 GPT(Generative Pre-Training) 모델을 소개했다. 2019년 데블린(Devlin et al.)은 BERT 모델을 사용하여 미세 조정을 완벽하게 했다. 래드포드(Radford et al.)는 GPT-2 모델까지 나아갔다. 브라운(Brown et al.)은 미세 조정이 필요 없는 트랜스포머에 대한 GPT-3 제로샷 방식을 정의했다!

동시에 왕(Wang et al.)은 2019년에 NLP 모델을 벤치마킹하기 위해 일반 언어 이해 평가(General Language Understanding Evaluation, GLUE)를 만들었다. 그러나 트랜스포머 모델은 너무 빠르게

진화하여 인간의 기준값<sup>20</sup>을 뛰어넘었다!

2019년과 2020년에 왕은 SuperGLUE를 빠르게 만들었고, 인간의 기준선을 더 높게 만들었으며, NLP/NLU 작업을 더 어렵게 만들었다. 트랜스포머는 빠르게 발전하고 있으며, 일부 트랜스포머는 이 글을 쓰는 시점에 이미 SuperGLUE 순위표에서 인간의 기준값을 넘어섰다.

이런 일이 어떻게 이렇게 빨리 일어났을까?

어떻게 이런 발전이 일어났는지 이해하기 위해 먼저 모델의 크기 측면에서 살펴보자.

## 7.2.2 트랜스포머 모델 크기 증가

표 7.1에서 볼 수 있듯, 2017년부터 2020년까지 파라미터 수가 기존 트랜스포머 모델 6500만 개에서 GPT-3 모델 1,750억 개로 증가했다.

트랜스포머 모델	논문	파라미터 수
Transformer Base	바스와니(Vaswani et al.) (2017)	65M
Transformer Big	바스와니(Vaswani et al.) (2017)	213M
BERT-Base	데블린(Devlin et al.) (2019)	110M
BERT-Large	데블린(Devlin et al.) (2019)	340M
GPT-2	래드포드(Radford et al.) (2019)	117M
GPT-2	래드포드(Radford et al.) (2019)	345M
GPT-2	래드포드(Radford et al.) (2019)	1.5B
GPT-3	브라운(Brown et al.) (2020)	175B

표 7.1 트랜스포머 파라미터 수의 변화

표 7.1에는 2017년부터 2020년까지 짧은 기간 동안 설계된 주요 모델만 포함한다. 논문 출판 날짜는 모델이 실제로 설계된 날짜 이후이다. 또한 저자들은 논문을 업데이트했다. 예를 들어, 오리지널 트랜스포머가 시장을 움직이기 시작하자 구글 브레인, OpenAI, 페이스북 AI 각각에서 새로운 트랜스포머 모델을 소개했다.

20 역주. 실제로 인간이 데이터셋의 문제를 풀고 얻은 점수이다.

또한 일부 GPT-2 모델은 더 작은 GPT-3 모델보다 파라미터 개수가 더 크다. 예를 들어, GPT-3 Small 모델은 파라미터 1억 2,500만 개가 있고 파라미터 3억 4,500만 개 파라미터가 있는 GPT-2 모델이 존재한다.

아키텍처 크기도 동시에 커져갔다.

- 모델의 층 수는 오리지널 트랜스포머 6개에서 GPT-3 모델 96개로 증가했다.
- 층의 헤드 수는 오리지널 트랜스포머 8개에서 GPT-3 모델 96개로 증가했다.
- 컨텍스트 크기가 오리지널 트랜스포머 512 토큰에서 GPT-3 모델 12,288개로 증가했다.

96개 층으로 구성된 GPT-3 175B가 40개 층에 불과한 GPT-2 1,542M보다 더 인상적인 결과를 주는 것은 아키텍처 크기의 차이이다. 두 모델의 파라미터 수는 비슷하지만 층 개수는 두 배 차이이다.<sup>21</sup>

이제 컨텍스트 크기에 초점을 맞춰 트랜스포머의 빠른 진화에 대한 또 다른 측면을 이해해 보자.

### 7.2.2.1 컨텍스트 크기 및 최대 경로 길이(maximum path length)

트랜스포머 모델의 초석은 어텐션 서브 층(attention sub-layers)에 있다. 어텐션 서브 층의 핵심 특징은 컨텍스트를 처리하는 데 사용한 방법이다.

컨텍스트는 인간과 기계가 언어를 학습하는 주요 방법 중 하나다. 컨텍스트 크기가 클수록 주어진 시퀀스를 더 많이 이해할 수 있다.

컨텍스트 크기가 컸을 때 단점은 단어가 의미하는 바를 이해하는 데 필요한 단어 간 거리가 길어진다는 점이다. 장기 의존성을 분석하려면 순환 층에서 어텐션 층으로 변경해야 하는 이유를 알아보자.

다음 문장은 대명사 'it'이 무엇을 가리키는지 찾기 위해 긴 경로를 거쳐야 한다.

“Our house was too small to fit a big couch, a large table, and other furniture we would have liked in such a tiny space. We thought about staying for some time, but finally, we decided to sell it.”

문장 마지막 단어 'it'의 의미는 문장 시작에 있는 'Our house' 단어로 거슬러 올라가야만 설명할 수 있다. 기계 입장으로는 꽤 긴 경로이다.

21 역주. GPT-3 175B와 GPT-2 1,542M은 파라미터 개수는 약 113배다. 이를 저자는 비슷하다고 표현한 것은 다음과 같이 비유가 가능하다. 키가 아주 큰 농구 선수가 작은 사람 옆에 있을 때, 바로 옆에서 보면 둘의 차이는 상당히 크게 느껴진다. 하지만 상공에 있는 비행기에서 보았을 때는 둘의 차이는 작게 보인다. 구글은 현재 1조 개 파라미터 모델을 연구하고 있다는 점에서 GPT-3 175B와 GPT-2 1,542M의 파라미터 수 차이는 비슷하다고 볼 수 있다.

최대 경로 길이를 결정짓는 함수의 복잡도와 컨텍스트 개수는 표 7.2에 있다.

층 종류	최대 경로 길이	컨텍스트 크기
셀프 어텐션	$O(1)$	1
순환	$O(n)$	100

표 7.2 최대 경로 길이

2017년 바스와니(Vaswani et al.)는 오리지널 트랜스포머 모델에서 컨텍스트 분석 설계를 최적화했다. 어텐션은 컨텍스트 분석 연산을 일대일 토큰 연산으로 축소시켰다. 모든 층이 동일하기에 트랜스포머 모델 크기를 훨씬 쉽게 확장할 수 있다. GPT-3 모델을 컨텍스트 크기가 10으로 학습할 때와 100으로 학습할 때 최대 경로 길이는 동일하다.

예를 들어, 순환 층은 토큰 별로 컨텍스트를 저장해야 한다. 따라서, 최대 경로 길이가 곧 컨텍스트 크기이다. 최대 경로 길이가 GPT-3 모델일 때 보다  $O(n)$  배 더 길어진다. 멀티-헤드 어텐션(multi-head attention)은 헤드별 GPU를 사용해 병렬 학습이 가능한 반면 RNN은 그렇게 할 수 없다.

유연하고 최적화된 트랜스포머 아키텍처는 다른 요소에도 영향을 미쳤다.

- 바스와니는 36M개 문장으로 최첨단 트랜스포머 모델을 학습시켰다. 브라운은 400B바이트 쌍 인코딩 토큰으로 GPT-3 모델을 학습시켰다.
- 대형 트랜스포머 모델 학습은 전 세계 소수 팀만이 가능할 정도의 학습 리소스가 필요하다. 브라운은 GPT-3 175B를 학습하는 데 총  $2.14 * 10^{23}$  FLOPS를 사용했다.
- 트랜스포머 아키텍처 설계는 세계에서 소수 조직만이 자금을 지원할 수 있는 우수한 자격을 갖춘 팀이 필요하다.

규모와 아키텍처는 계속 발전해 가까운 미래에 조 단위 파라미터 모델까지 발전할 것이다. 슈퍼컴퓨터는 트랜스포머 학습에 필요한 리소스를 계속 제공할 것이다.

이제 제로샷 모델을 어떻게 구현했는지 살펴보자.

### 7.2.3 미세 조정부터 제로샷 모델까지

2018년, 래드포드(Radford et al.)가 이끌던 OpenAI 연구팀은 특정 작업용 모델부터 GPT-3 같은 제로샷 모델까지 트랜스포머를 사용하기를 원했다. 목표는 라벨 없는 데이터로 트랜스포머 학습시키는 것이었다. 어텐션 층이 비지도 학습으로 언어를 학습하는 것은 현명한 조치였다. OpenAI는 트랜스포머가 특정 NLP 작업을 수행하도록 가르치는 대신 언어를 학습하기로 결정했다.

OpenAI는 범용 모델을 만들고자 했다. 그래서 전문가가 라벨을 붙인 데이터에 의존하는 대신 원시 데이터로 트랜스포머 모델을 학습하기 시작했다. 데이터에 라벨을 붙이는 작업은 시간이 많이 걸리고 트랜스포머 학습 과정도 상당히 느리게 만든다. 처음에는 트랜스포머 모델을 비지도 학습한 뒤 지도 학습으로 미세 조정한다.

OpenAI는 디코더 층 쌓기 섹션에서 설명할 디코더 전용 트랜스포머를 선택했다. 이 트랜스포머의 평가 지표는 매우 설득력 있었고 NLP 연구소들 중 최고 NLP 모델 수준에 빠르게 도달했다.

GPT 트랜스포머 모델 첫 번째 버전의 유망한 결과를 보고 2019년에 래드포드는 제로샷 전이 모델(zero-shot transfer model)을 고안했다. 이들의 핵심 철학은 원시 텍스트로 GPT 모델을 계속 학습하는 것이었다. 그런 다음 비지도 분포 예시들을 보고 언어 모델링에 집중하여 연구했다.

$$\text{Examples} = (x_1, x_2, x_3, \dots, x_n)$$

이 예시들은 기호 시퀀스로 구성된다.

$$\text{Sequences} = (s_1, s_2, s_3, \dots, s_n)$$

이를 통해 모든 유형의 입력에 대해 확률 분포로 표현할 수 있는 메타 모델을 만들 수 있었다.

$$p(\text{output}/\text{input})$$

집중적인 학습으로 GPT 모델이 일단 언어를 이해하면, 모든 유형의 다운스트림 작업에 이 개념트를 일반화하는 것이 목표였다.

GPT 모델은 117M 개 파라미터에서 345M 개, 1B 개 이상 파라미터 트랜스포머까지 발전했다. 미세 조정량은 상당히 감소했다. 그 결과 최고 수준 평가 지표에 도달했다.

이를 통해 OpenAI는 훨씬 더 발전할 수 있었다. 브라운은 조건부 확률 트랜스포머 모델이 심층적으로 학습할 수 있다는 가정을 바탕으로 다운스트림 작업용 미세 조정이 거의 또는 전혀 없이도 우수한 결과를 생성할 수 있다는 가정을 세웠다.

$$p(\text{output}/\text{multi- tasks})$$

OpenAI는 모델을 학습한 다음 추가 미세 조정없이 바로 다운스트림 작업을 실행하는 목표에 도달했다. 이런 경이로운 발전은 네 단계로 설명할 수 있다.

- **미세 조정(Fine-tuning, FT)**은 이전 장에서 살펴본 의미대로 수행되어야 한다. 트랜스포머 모델을 학습한 다음 다운스트림 작업에서 미세 조정을 한다는 것이다. 래드포드는 많은 미세 조정 작업을 설계했다. 이후 OpenAI 팀은 이런 미세 조정 작업 수를 0개까지 줄여나갔다.
- **퓨샷(Few-Shot, FS)**은 큰 진전을 의미한다. GPT가 학습 이후 추론이 필요한 경우, 추론 작업 데모 시연이 조건문으로 제공된다. 이런 조건문 제공은 가중치 업데이트를 대체하기에 GPT 팀은 가중치 업데이트 부분을 제거했다. 이 챕터에서 이후 텍스트 완성 예시를 살펴보면 조건문 제공을 모델에 적용해 볼 것이다.
- **원샷(One-Shot, 1S)**은 한 걸음 더 나아간다. 학습한 GPT 모델에게 다운스트림 작업 데모 시연을 한 번만 제공한다. FS과 마찬가지로 가중치 업데이트를 하지 않는다.
- **제로샷(Zero-Shot, ZS)**은 최종 목표다. 학습한 GPT 모델에게 다운스트림 작업 데모 시연을 제공하지 않는다.

이런 접근 방식은 각각 다양한 수준의 효율성을 갖는다. OpenAI GPT 팀은 최첨단 트랜스포머 모델 제작에 많은 노력을 해왔다.

GPT 모델 아키텍처에는 다음 동기가 반영되어 있다.

- 광범위한 학습을 통해 트랜스포머 모델에게 언어를 배우는 방법 가르치기
- 컨텍스트 조건문을 통해 언어 모델링에 집중하기
- 트랜스포머는 컨텍스트를 파악하여 참신한 방식으로 텍스트 완성을 생성한다. 다운스트림 작업 학습에 리소스를 소비하는 대신, 어떤 작업이든 입력을 이해하고 추론하는 데 집중한다.
- 입력 시퀀스 일부를 마스킹하여 모델을 학습시키는 효율적인 방법을 찾아 트랜스포머가 기계 지능으로 사고하게 만든다. 따라서 기계 지능은 인간은 아니지만 효율적이다.

이제 디코더 전용 GPT 모델을 살펴보자.

## 7.2.4 디코더 총 썬기

이제 OpenAI 팀은 언어 모델링에 중점을 두었다는 것을 알았다. 그리고 마스크드 어텐션 서브 층을 사용하는 것이 합리적인 것도 이해가 된다. 따라서 디코더 스택은 유지하고 인코더 스택을 제외하기로 결정했다. 브라운은 디코더 전용 트랜스포머 모델의 크기를 획기적으로 늘려 우수한 결과를 얻었다.

GPT 모델은 바스와니가 설계한 오리지널 트랜스포머의 디코더 스택과 구조가 동일하다. 디코더 스택은 2장, 트랜스포머 모델 아키텍처 살펴보기에서 다뤘다. 필요하다면 2장을 다시 살펴보자.

GPT 모델은 그림 7.1과 같이 디코더 전용 아키텍처를 사용한다.

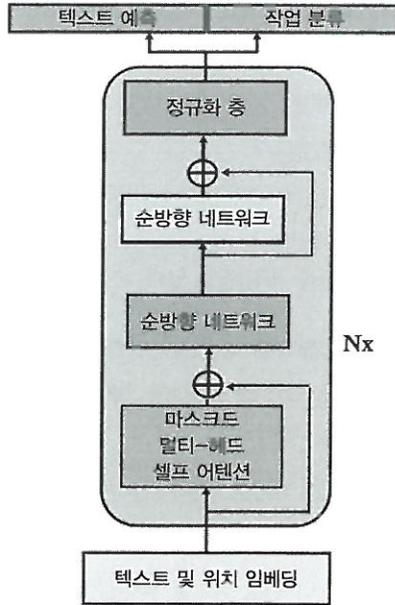


그림 7.1 GPT 디코더 전용 아키텍처

텍스트 및 위치 임베딩 서브 층, 마스크드 멀티-헤드 셀프 어텐션 층, 정규화 서브 층, 순방향 서브 층 및 출력이 있다. 게다가 텍스트 예측과 작업 분류를 갖춘 GPT-2 버전도 존재한다.

OpenAI 팀은 모델별 디코더 모델을 커스터마이징했다.

래드포드는 2019년에 네 개 이상의, 브라운은 2020년에 여덟 개 이상의 GPT 모델을 제안했다.

GPT-3 175B 모델은 전 세계 소수 팀만이 사용할 수 있는 컴퓨터 리소스를 요구할 수준의 크기에 도달했다.

$$n_{\text{params}} = 175.0\text{B}, n_{\text{layers}} = 96, d_{\text{model}} = 12288, n_{\text{heads}} = 96$$

다양한 GPT-3 엔진에 대해 살펴보자.

## 7.2.5 GPT-3 엔진

GPT-3 모델은 다양한 크기의 특정 작업을 수행하도록 학습할 수 있다. 현재 사용 가능한 엔진 목록은 OpenAI에서 확인할 수 있다.(<https://platform.openai.com/docs/models>)

- 기본 엔진들은 다양한 기능을 갖고 있다.
- Davinci 엔진은 복잡한 의도 분석 가능
- Curie 엔진은 빠르고 요약 기능이 뛰어남
- Babbage 엔진은 의미론적 검색이 뛰어남
- Ada 엔진은 텍스트 구문 분석이 뛰어남

OpenAI는 시장에 출시할 더 많은 엔진을 제작하고 있다.

- 지시(instruct) 시리즈는 묘사 기반 설명서를 제공한다. 마이크로소프트 edge 설정에 들어가는 법 알려줘라고 클라이언트가 요청하면 edge 설정에 들어가는 설명서를 알려주는 형태다. 이 챗터의 GPT-3 추가 예제 섹션에서 예제를 확인할 수 있다.
- 코덱스(codex) 시리즈는 자연어를 소스 코드로 변환해 준다. 이 시리즈는 **16장, 트랜스포머 기반 코파일럿의 등장**에서 살펴볼 것이다.
- 콘텐츠 필터 시리즈는 안전하지 않거나 민감 정보를 필터링한다. **16장, 트랜스포머 기반 코파일럿의 등장**에서 살펴볼 것이다.

미세 조정에서 제로 샷 GPT-3 모델에 이르는 과정을 살펴봤다. GPT-3가 다양한 엔진을 만들 수 있다는 것도 살펴봤다.

이제 GPT 모델 소스 코드를 살펴보자. 아직 GPT-3 트랜스포머 모델 소스 코드는 공개되어 있지 않지만, GPT-2 모델 소스 코드로도 GPT 모델들의 내부 동작 방식을 이해하기에 충분하다.

이제 GPT-2 모델과 상호 작용하고 학습할 준비가 됐다.

먼저 텍스트 완성을 위해 학습한 GPT-2 345M 모델을 사용해 보자. 이 모델은 헤드 셀프 어텐션 하위 층 16개와 디코더 층 24개를 가진다.

그런 다음 커스터마이징된 텍스트 완성 GPT-2 117M을 학습해 보자. 이 모델은 헤드 셀프 어텐션 하위 층 12개와 디코더 층 12개를 가진다.

자 이제 사전 학습된 GPT-2 345M 모델과 상호 작용 해보자.

## 7.3

# GPT-2를 사용한 일반 텍스트 완성

GPT-2 일반 모델을 사용한 예제를 자세히 살펴보자. 이 예제의 목표는 GPT 모델이 도달할 수 있는 추상적 추론의 수준을 알아내는 것이다.

이 섹션은 텍스트 완성을 위해 GPT-2 모델과 상호 작용하는 방법을 설명한다. **부록 III, GPT-2를 사용한 일반 텍스트 완성**에서 자세히 설명한 `OpenAI_GPT2.ipynb` 파일의 9단계 부분에 초점을 맞출 것이다.

### NOTE

먼저 이 섹션을 읽고 사전 학습된 일반 GPT-2 모델이 특정 예제에 어떻게 반응하는지 확인할 수 있다. 이후 **부록 III, GPT-2를 사용한 일반 텍스트 완성**을 읽고 일반 GPT-2 모델이 구글 코랩에서 어떻게 구현되는지 살펴보아라. 다음 설명된 9단계 상호 작용이 포함된 **부록 III**을 바로 읽는 것도 괜찮다.

먼저 사전 학습된 GPT-2를 적용한 구체적인 예제를 살펴보자.

### 7.3.1 9단계: GPT-2와 상호 작용하기

이 섹션에서 GPT-2 345M 모델을 사용한다.

모델과 상호 작용하려면 `interact_model` 셀을 실행한다.

```
#@title 9단계: GPT-2와 상호 작용하기
interact_model('345M',None,1,1,300,1,0,'content/gpt-2/models')
```

실행하고 나면 컨텍스트를 입력하라는 메시지가 표시된다.

prompt >>>

그림 7.2 텍스트 완성을 위한 컨텍스트 입력

표준 GPT-2 모델이기 때문에 원하는 모든 유형의 컨텍스트 사용이 가능하다.

임마누엘 칸트 문장을 사용해 보자.

Human reason, in one sphere of its cognition, is called upon to consider questions, which it cannot decline, as they are presented by its own nature, but which it cannot answer, as they transcend every faculty of the mind.

#인간의 이성 은 인식의 한 영역에서 본성상 피할 수 없는 질문을 요청받는 운명을 지니고 있지만, 자신의 모든 능력을 넘어서 있어 스스로 답할 수도 없는 운명을 겪게 된다.

엔터를 눌러 텍스트 생성해 보자. 입력한 임마누엘 칸트 문장이 GPT-2 모델 학습 데이터셋에 없었을 것이다. 그리고 GPT-2는 확률적 모델이기에 출력 결과는 상대적으로 무작위이다.

필자가 실행했을 때 모델이 생성한 첫 몇 줄을 살펴보자.

"We may grant to this conception the peculiarity that it is the only causal logic. In the second law of logic as in the third, experience is measured at its end: apprehension is afterwards closed in consciousness. The solution of scholastic perplexities, whether moral or religious, is not only impossible, but your own existence is blasphemous."

# 우리는 이것이 유일한 인과 관계임을 인정할 수 있다. 논리의 두 번째와 세 번째 법칙에서도 경험은 그 끝에서 측정된다. 두려움은 이후 의식 속에서 사라진다. 도덕적이든 종교적이든 학문적 난해함의 해결은 불가능할 뿐 아니라, 신성 모독적이다.

셀을 중지하려면 셀 실행 버튼을 두 번 클릭하거나 **Ctrl+M**을 눌러 편집 모드에서 나와 커맨드 모드에서 **I**를 두 번 눌러 중지시킬 수 있다.

생성한 결과는 매우 표현이 풍부하다라는 몇 가지 사실을 관찰할 수 있다.

- 입력한 컨텍스트는 조건문으로 지정돼 모델이 결과물을 생성한다.
- 컨텍스트는 모델의 시연 하나로 동작했다. 이 모델은 매개변수를 수정하지 않고 컨텍스트에서 무엇을 말할지 익혔다.
- 텍스트 완성 은 컨텍스트를 조건문으로 사용한다. 이를 통해 미세 조정이 필요 없는 트랜스포머 모델을 가능케 한다.
- 의미론적 관점에서 보면 결과물이 더 흥미로울 수 있다.
- 문법적인 관점에서 볼 때 결과물은 설득력이 있다.

문장 생성을 더 잘할 수는 없을까? 다음 챕터는 커스텀 텍스트 완성의 상호 작용을 다룬다.

## 7.4

# 커스텀 GPT-2 언어 모델 학습

이 섹션에서는 특정 데이터셋에 학습한 커스텀 GPT-2 모델 예시를 살펴볼 것이다. 목표는 GPT 모델이 달성할 수 있는 추상적 추론의 수준을 알아보는 것이다.

이 섹션에서 특정 데이터셋을 학습한 텍스트 완성 GPT-2 모델과의 상호 작용에 대해 설명한다. 부록 IV, GPT-2를 사용해 커스텀 텍스트 완성하기에서 상세히 다룬 `Training_OpenAI_GPT_2.ipynb` 노트북의 12단계를 상세히 볼 것이다.

### [설명 추가]

`Training_OpenAI_GPT_2.ipynb`은 텐서플로우 1.x 버전에 작동한다. 하지만 2023년 8월부터, 구글 코랩은 텐서플로우 1.x 버전을 지원하지 않는다. 두 가지 해결책이 있다.

- (1) `Training_OpenAI_GPT_2.ipynb`을 텐서플로우 1.x가 설치된 로컬 머신에 받아 실행하기
- (2) 구글 코랩에서 깃허브 내에 BONUS 디렉터리에 존재하는 `GPT_2_and_ChatGPT_the_Origins.ipynb`을 실행하기
  - `GPT_2_and_ChatGPT_the_Origins.ipynb`은 다음 내용을 다룬다.
  - GPT-2의 소스 코드 가이드
  - GPT-2를 사용한 텍스트 생성의 허깅페이스 구현
  - OpenAI ChatGPT의 배경 엔진인 OpenAI davinci-instruct 시리즈(GPT-3)를 사용한 텍스트 생성 대화 예제를 실행해 davinci-instruct-beta 엔진의 출력과 ChatGPT의 출력을 비교해 볼 수 있다.
  - 커스텀 챗봇을 미세 조정 방법

### NOTE

이 섹션에서는 커스텀 GPT-2 모델 응답 개선을 다루는 예제를 살펴본다. 부록 IV, GPT-2를 사용해 커스텀 텍스트 완성하기에서는 특정 응답을 얻기 위한 GPT-2 모델 학습 방법을 이해할 수 있다. 부록 IV에서 다음의 12단계를 다루고 있다. 부록을 바로 읽어도 좋다.

먼저 GPT-2를 학습시킴으로써 상호 작용이 어떻게 개선됐는지 알아보자.

### 7.4.1 12단계: 대화형 컨텍스트 및 완성 예제

이제 조건부 샘플을 실행해 보자. 우리가 입력한 컨텍스트는 우리가 원하는 대로 모델이 생각하고 맞춤형 단락을 생성하여 텍스트를 완성토록 한다.

다음 셸을 실행해 보자.

```
#@title 12단계: 대화형 컨텍스트 및 완성 예제
import os # 런타임이 재시작된 이후에 임포트한다.
os.chdir("/content/gpt-2/src")
!python interactive_conditional_samples.py --temperature 0.8 --top_k 40
--model_name '117M'
```

필요한 경우 잠시 **부록 III, GPT-2를 사용한 일반 텍스트 완성**, 9단계로 가서 응답의 차이점을 확인해 보아라. 프로그램은 컨텍스트를 입력하라는 메시지를 표시한다.

prompt >>>

그림 7.3 텍스트 완성을 위한 컨텍스트 입력

이 챕터의 GPT-2를 사용하여 일반 텍스트 완성 섹션의 9단계에서 했던 임마누엘 칸트가 쓴 동일한 단락을 입력해 보자.

```
Human reason, in one sphere of its cognition, is called upon to consider
questions, which it cannot decline, as they are presented by its own nature,
but which it cannot answer, as they transcend every faculty of the mind.
```

```
#인간의 이성(Reason)은 인식의 한 영역에서 본성상 피할 수 없는 질문을 요청받는 운명(Fate)을 지니고 있지만, 자신의
모든 능력을 넘어서 있어 스스로 답할 수도 없는 운명을 겪게 된다.
```

이전에 했던 것처럼 엔터를 눌러 텍스트를 생성해 보자. 출력은 구조적이고 논리적이긴 하지만 실행할 때마다 달라질 수 있어서 트랜스포머의 매력이 돋보인다. 이번 결과는 무작위가 아니며 인상적이다.

GPT-2 모델이 생성한 결과 몇 줄을 살펴보자.

"Now, what is the nature of the reason, except that it is of a kind and power of mind that is so called on to think, and to understand?

The reason is the thing which apprehends, and seeks to understand, the objects of its minds, which apprehend and apprehend the things of itself. No one can speak of this reason without the understanding and the understanding of the human reason. But that knowledge and understanding which is the most natural of all the faculties, which, because it is understood by the human reason, is the most natural of all the faculties, is not without reason, but only without understanding. The human reason, who sees and knows the things which the natural sense, which, though in the natural sense, does not understand, is the source of every other knowledge which we understand. The reason is the thing which does not understand, but understands the things which we comprehend, and therefore, is not to be understood, I mean that the reason is the thing which does not understand, but knows only the things which we do understand..."

#그렇다면, 생각하고 이해하도록 하는 마음의 힘이라는 것 외에는 이성의 본질이란 무엇인가? 이성(이성)은 마음의 대상을 파악하고 이해하는, 자기 자신의 대상을 파악하는 것이다. 인간 이성(이성)에 대한 이해와 그 이해 없이는 아무도 이성(이성)에 대해 말할 수 없다. 그러나 인간 이성(이성)에 의해 이해되기 때문에 모든 능력 중 가장 자연스러운 능력인 지식과 이해는 이성(이성)이 없는 것이 아니라 이해가 없기 때문에 가장 자연스럽다. 자연적 감각으로 이해하지 못할지라도 자연적인 감각으로 보고 인식하는 인간의 이성(이성)은 우리가 이해하는 다른 모든 지식의 원천이다. 이성(이성)은 이해하지 못하지만 우리가 인식하는 것은 이해하기 때문에 이해할 수 있는 것이 아니다. 이성(이성)은 이해하지 않는 것이 아니라 우리가 이해하는 것만 이해하는 것이다.

셀을 중지하려면 셀 실행 버튼을 두 번 클릭하거나 **Ctrl+M**을 눌러 편집 모드에서 나와 커맨드 모드에서 **I**를 두 번 눌러 중지시킬 수 있다.

와우! 학습한 GPT-2 모델이 완성한 텍스트와 사람이 완성한 텍스트의 차이를 알아차릴 수 있는 사람은 아무도 없을 것이다. 또한 각 실행마다 다른 출력을 생성할 수도 있다.

사실, 나는 우리 모델이 철학, 이성, 논리에 대한 추상적인 문제에서 많은 인간을 능가할 수 있다고 생각한다.

실험을 통해 몇 가지 결론을 도출할 수 있다.

- 잘 학습한 트랜스포머 모델은 인간 수준의 텍스트 완성을 생성할 수 있다
- GPT-2 모델의 복잡하고 추상적인 추론 기반 텍스트 생성은 거의 인간 수준에 도달할 수 있다
- 텍스트 컨텍스트는 예상하는 내용을 모델에 조건문으로 주는 효율적인 방법이다
- 텍스트 완성은 컨텍스트 문장이 조건으로 제공된 경우 조건에 따라 텍스트를 생성하는 것이다

조건부 텍스트 컨텍스트를 입력하여 텍스트 완성을 실험해 볼 수 있다. 자체 데이터로 모델을 학습할 수도 있다. dset.txt의 내용을 자체 데이터로 바꾸고 확인해 보라.

학습한 GPT-2 모델은 사람처럼 반응한다는 것을 기억하라. 짧거나 불완전하거나 흥미롭지 않거나 까다로운 컨텍스트를 입력하면 당황하거나 나쁜 결과를 얻을 수 있다. 이는 GPT-2가 실제 생활에서와 마찬가지로 우리에게 최선을 다할 것을 기대하기 때문이다!

GPT-2에서 테스트한 예제에 학습한 GPT-3는 어떻게 응답하는지 확인해 보자.

## 7.5

# OpenAI GPT-3 작업 실행하기

이 챕터에서는 두 가지 방법으로 GPT-3를 실행한다.

- 코드 없이 온라인으로 GPT-3 작업 실행
- 구글 코랩 노트북에서 GPT-3 구현

### NOTE

GPT-3 API에 가입하면 OpenAI에서 무료 예산을 제공한다. 이 무료 예산으로 이 책의 예제를 한두 번 실행하는 데 비용 대부분을 충당할 수 있다.

먼저 온라인에서 NLP 작업을 실행해 보자.

## 7.5.1 온라인에서 NLP 작업 실행하기

API 없이 몇 가지 예시를 GPT-3에 직접 요청해 보자.

먼저 프롬프트(prompt)와 응답에 사용할 변수를 정의해 보자.

- **N**: NLP 작업명(입력값)
- **E**: GPT-3 엔진에 입력할 설명(입력값), E는 T 앞에 온다.
- **T**: GPT-3가 살펴볼 텍스트 혹은 콘텐츠(입력값)
- **S**: GPT-3에 예상하는 것을 보여주기, S는 T 뒤에 오며 필요할 때 추가된다.(입력값)
- **R**: GPT-3의 응답(출력값)

앞에 설명한 프롬프트 변수는 하나의 가이드라인이다. GPT-3는 매우 유연해 다양한 변형이 가능하다.

이제 API 없이 온라인으로 몇 가지 교육용 예제를 실행할 준비가 됐다.

● 기존 지식에 관한 질의 응답(Q&A):

E=Q

T=1965년 미국의 대통령은 누구인가?

S=없음

R=A

**프롬프트와 응답**

Q: 1965년 미국의 대통령은 누구인가?

A: 린든 B 존슨(Lyndon B. Johnson)이 1965년 미국의 대통령이었다.

Q: 달에 도착한 첫 번째 사람은 누구인가?

A: 닐 암스트롱(Neil Armstrong)이 달에 도착한 첫 번째 사람이다.

● 영화를 이모티콘으로 변환(Movie to Emoji)

E=영화 제목

T=없음

S=예시로 암묵적으로 제시

R=이모티콘

**프롬프트와 응답**

백 투 더 퓨처: 🕒🚗👨🏻👉👨🏻👈🕒

배트맨: 🦇🦹🏻

트랜스포머: 🚗👨🏻👉👨🏻👈🚗

**새 프롬프트와 응답**

미션 임파서블: 🚗👨🏻👉👨🏻👈🚗

● 초등학교 2학년생을 위한 요약(Summarize for a 2nd grader)

E=2학년 학생이 다음 지문의 의미를 물었다.

T="""잠정 결론은..."""

S=이 지문을 2학년 학생이 이해할 수 있는 쉬운 표현으로 다시 작성하였다.

""

R=요약 결과

**프롬프트와 응답**

2학년 학생이 다음 지문의 의미를 물었다.

“”잠정 결론은 사실과 허구 두 가지 범주로 나뉘 볼 수 있다. 한 가지 사실은 OpenAI가 세계에서 가장 강력한 NLP 서비스 하나를 보유하고 있다는 것이다. 다른 사실은: OpenAI 엔진은 모든 트랜스포머 모델을 찾을 필요도 없고, 사전 학습과 미세 조정이 필요 없는 강력한 제로샷 엔진이다. 모델 학습에 사용한 슈퍼컴퓨터는 독보적이다. 프롬프트를 잘 설계하면 놀라운 정도로 정확한 응답을 얻을 수 있다. 이 챗봇의 NLP 작업 구현은 초보 개발자도 수행할 수 있는 복사 및 붙여 넣기 작업만 필요했다. 허구는 많은 사람들이 인공지능이 데이터 과학자와 인공지능 전문가를 대체할 것이라는, 반 이상향적이고 과장된 주장에서 시작한다. 과연 그럴까? 이 질문에 답하기 전에 먼저 이 챗봇에서 실행한 예제에 대해 다음과 같은 질문을 스스로에게 해보자. 문장이 틀린 지 어떻게 알 수 있을까? 사람이 직접 읽고 확인하지 않고 어떻게 정답인지 알 수 있을까? 엔진은 이 작업이 문법 교정 작업이라는 것을 어떻게 알았을까? 응답이 잘못된 경우, 잘 설계한 사용자 인터페이스에서 프롬프트를 개선하거나 수동 모드로 되돌리기 위해 어떤 일을 했는가? 이런 질문에 답하려면 규칙 기반, 자동화된 품질 관리 파이프라인 및 기타 여러 도구를 사용하여 사람이 직접 개입해야 한다. 앞의 사실들은 꽤나 설득력이 있다. 트랜스포머로 NLP 작업을 실행하면 대부분 개발이 거의 필요하지 않다. 허구는 설득력이 없다. 인간은 여전히 필요하다. OpenAI 엔진은 인간을 대체하기 위한 것이 아니라 더 높은 수준의 만족스러운 작업을 수행하도록 돕기 위한 것이다. 즉, 제트기를 만들지 않고 비행할 수 있다는 것이다!”” 이 지문을 2학년 학생이 이해할 수 있는 쉬운 표현으로 다시 작성하였다.””

첫 결론은 OpenAI가 세계에서 가장 강력한 NLP 기술을 가졌다는 것이다. 여기서 가장 중요한 사실은 다음과 같다. OpenAI 엔진은 모든 트랜스포머 모델을 찾을 필요도 없고, 사전 학습과 미세 조정이 필요 없는 강력한 제로샷 엔진이다. 모델 학습에 사용한 슈퍼컴퓨터는 독보적이다. 프롬프트를 잘 설계하면 놀라운 정도로 정확한 응답을 얻을 수 있다.

결과가 인상적이고 NLP 역사에 큰 진전으로 보인다.

기대하는 것을 표시할 수 있기 때문에 OpenAI 제로샷 GPT-3 엔진으로 구현 가능한 NLP 작업 수는 무한하다.

AI 전문가로서 바로 사용할 수 있는 API 구현하기 전에 직접 코드를 작성해 볼 필요가 있다. 이제 OpenAI GPT 모델의 아키텍처를 살펴본 다음 GPT-2 모델을 빌드하여 엔진이 어떻게 작동하는지 살펴보자.

GPT 모델에 대해 더 많이 알수록 실제 프로젝트에서 NLP 작업을 더 잘 구현할 수 있다.

위에서 아래로 접근하는 방식(top-to-bottom approach)을 이어가면서 OpenAI GPT 트랜스포머 모델의 아키텍처를 자세히 살펴보자.

## 7.5.2 GPT-3 엔진 시작하기

OpenAI는 세계에서 가장 강력한 트랜스포머 엔진을 보유하고 있다. 하나의 GPT-3 모델은 수백 가지 작업을 수행할 수 있다. GPT-3는 학습하지 않은 많은 작업을 수행할 수 있다.

이 섹션에서는 `Getting_Started_GPT_3.ipynb`의 API를 사용한다.

GPT-3를 사용하기 위해 먼저 OpenAI 웹사이트(<https://openai.com/>)에서 회원가입을 하자.

OpenAI는 구글 번역과 같이 사용자 친화적인 온라인 서비스를 누구나 사용해 볼 수 있는 놀이터를 제공한다. 이제 몇 가지 작업을 시도해 보자.

### 7.5.2.1 GPT-3로 첫 NLP 작업 실행하기

몇 가지 단계를 거쳐 GPT-3 사용을 해보자.

이 책의 깃허브(Github)에 7장 디렉터리에 있는 `Getting_Started_GPT_3.ipynb`을 구글 코랩에서 열어 보자.

노트북의 설정을 변경할 필요는 없다. API를 사용하므로 이 섹션의 작업은 로컬 컴퓨팅 파워가 필요하지 않기 때문이다.

이 섹션에서 진행할 단계는 노트북에 작성된 단계와 동일하다.

NLP 실행은 간단한 세 단계로 이루어진다.

#### 1단계: OpenAI 설치하기

다음 명령으로 `openai`를 설치한다.

```
try:
    import openai
except:
    !pip install openai
    import openai
```

`openai`가 설치되어 있지 않다면, 런타임을 다시 시작해야 한다. 다음 출력과 같이 이 작업을 수행해야 하는 시점을 나타내는 메시지가 표시된다.

```
WARNING: The following packages were previously imported in this runtime:
[pandas]
You must restart the runtime in order to use newly installed versions.

RESTART RUNTIME
```

런타임을 다시 시작한 다음 셀을 다시 실행하여 openai를 불러왔는지 확인한다.

## 2단계: API 키 입력하기

OpenAI 회원가입을 하면 파이썬, C#, Java 및 기타 언어에서 사용할 수 있는 API 키가 제공된다. 이 섹션에서는 파이썬을 사용한다.

```
openai.api_key=[YOUR API KEY]
```

이제 API 키를 다음 셀처럼 업데이트를 할 수 있다.

```
import os
import openai
os.environ['OPENAI_API_KEY']=[YOUR_KEY or KEY variable]
print(os.getenv('OPENAI_API_KEY'))
openai.api_key = os.getenv("OPENAI_API_KEY")
```

이제 NLP 작업을 실행해 보자.

## 3단계: 기본 파라미터로 NLP 작업 실행하기

문법 교정 작업에 대한 OpenAI 예제를 복사하여 붙여 넣자.

```
response = openai.Completion.create(
    engine="davinci",
    prompt="Original: She no went to the market.WnStandard American English:",
    temperature=0,
    max_tokens=60,
    top_p=1.0,
    frequency_penalty=0.0,
```

```
presence_penalty=0.0,
stop=["\n"]
)
```

위 셸은 “She no went to the market.” 문장의 문법 교정 작업이다.

응답을 파싱하여 원하는 대로 처리할 수 있다. OpenAI의 응답은 dictionary 객체이다. 이 객체는 작업에 대한 자세한 정보가 포함되어 있다. 이 객체를 출력해 보자:

```
#응답 객체 출력하기
print(response)
```

객체를 살펴보면,

```
{
  "choices": [
    {
      "finish_reason": "stop",
      "index": 0,
      "logprobs": null,
      "text": " She didn't go to the market."
    }
  ],
  "created": 1639424815,
  "id": "cml-4ElZfXL19jGRNQoojWRRGof8AKr4y",
  "model": "davinci:2020-05-03",
  "object": "text_completion"
}
```

“created”, “id”, “model”의 값은 실행할 때마다 달라질 수 있다.

이 객체의 “text”의 값이 작업 결과를 가리킨다. 출력을 해보자.

```
#응답 객체 출력하기
r = (response["choices"])[0]
print(r["text"])
```

“text”의 값은 문법적으로 올바른 문장임을 확인할 수 있다.

```
She didn't go to the market.
```

### 7.5.2.2 NLP 작업 및 예제

이제 GPT-3 엔진을 산업적 접근 방식으로 다뤄보자. 예를 들어 OpenAI는 API가 필요 없는 대화형 교육 인터페이스를 제공한다. 따라서 교사, 컨설턴트, 언어학자, 철학자 또는 교육 목적으로 GPT-3 엔진 사용을 원하는 누구에게나 AI에 대한 경험이 전혀 없어도 사용할 수 있다.

먼저 노트북에서 API를 사용하는 것부터 시작하자.

#### 문법 교정

이 챕터 GPT-3 엔진 시작하기 섹션에서 살펴본 `Getting_Started_GPT_3.ipynb`으로 돌아가면 다양한 프롬프트로 문법 교정을 실행할 수 있다.

파이썬 노트북을 열고 4단계: 예 1: 문법 교정(Grammar correction)으로 이동한다.

**#6단계: 커스텀 파라미터를 사용해 NLP 테스트 실행하기**

```
response = openai.Completion.create(
    #기본 엔진: davinci
    engine="davinci",
    #테스크:"Original"에 사용할 기본 프롬프트
    prompt="Original: She no went to the market.\n Standard American English.",
    temperature=0,
    max_tokens=60,
    top_p=1.0,
    frequency_penalty=0.0,
    presence_penalty=0.0,
    stop=["\n"]
)
```

요청 본문(body)는 프롬프트 외 다양한 파라미터가 있다. 본문은 몇 가지 주요 파라미터를 포함한다.

- `engine="davinci"`: 사용할 OpenAI GPT-3 엔진을 선택한다. 향후 다른 모델을 선택할 수 있다.
- `temperature=0`: 0.9와 같이 값이 높을수록 모델이 더 많은 위험을 감수해야 한다. `temperature`와 `top_p`를 동시에 수정하지 말자.
- `max_tokens=60`: 응답의 최대 토큰 수다.
- `top_p=1.0`: `temperature`와 같이 샘플링을 제어할 때 사용한다. 만약 `top_p`에 0.2 값을 할당하면, 시스템이 확률 질량 함수 값이 높은 상위 20% 토큰만 가져가도록 설정한다.
- `frequency_penalty=0.0`: 0과 1 사이 값을 설정할 수 있다. 응답에서 토큰의 빈도를 제한한다.
- `presence_penalty=0.0`: 0과 1사이 값을 설정할 수 있다. 새로운 토큰과 아이디어 사용을 강제한다.
- `stop=["\n"]`: 모델에 새 토큰 생성을 중지하는 신호를 설정한다.

파라미터 중 일부는 **부록 III, GPT-2를 사용한 일반 텍스트 완성의 7b-8단계**: 모델 불러오고 정의하기 섹션에 소스 코드 수준 설명이 있다.

접근 권한이 있는 경우 GPT-3 모델에서 혹은 **부록 III, GPT-2를 사용한 일반 텍스트 완성의 GPT-2** 모델에서 이러한 파라미터를 가지고 놀 수 있다. 파라미터 개념은 두 경우 모두 같다.

이 섹션에서는 프롬프트에 대해 중점적으로 설명한다.

```
prompt="Original: She no went to the market,\n\nStandard American English."
```

이 프롬프트는 세 가지 부분으로 나눌 수 있다.

- **Original**: 모델이 작업할 원본 텍스트가 뒤에 온다는 신호를 모델에 알린다.
- **She no went to the market.\n\n**: 모델에 원본 텍스트임을 보여준다.
- **Standard American English**: 모델에 어떤 작업을 예상하는지 보여준다.

작업을 변경하며 모델이 어떤 기능까지 도달할 수 있는지 확인하자.

#### ● Standard American English 경우

```
prompt="Original: She no went to the market.\n\nStandard American English:"
```

응답의 text는:

```
"text": " She didn't go to the market."
```

괜찮은 결과다. 결과에서 축약(didn't)을 원하지 않는다면 어떻게 해야할까?

#### ● 축약을 원하지 않는 경우

```
prompt="Original: She no went to the market.\n\nEnglish with no contractions:"
```

응답의 text는:

“text”: “ She **did not** go to the market.”

인상적이다. 다른 언어를 사용해보자.

- 축약없는 프랑스어 경우 응답 text

“text”: “ Elle n'est pas all\u00e9e au march\u00e9.”

인상적이다. \u00e9는 후처리가 필요하며 é가 된다.

더 많은 옵션이 가능하다. 다양한 분야의 무한한 상상력으로 사용해보자.

## GPT-3 추가 예제

OpenAI에는 많은 예제가 있다. 작업을 탐색할 수 있는 온라인 놀이터를 제공한다. 각 예제 소스 코드로 제공한다.(<https://platform.openai.com/examples>)

문법 교정 섹션에서 살펴본 것과 같은 예시를 클릭해 보자.

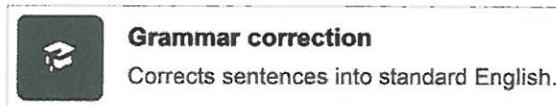


그림 7.4 OpenAI 문법 교정 섹션

각 작업의 프롬프트와 응답 설명을 제공한다.

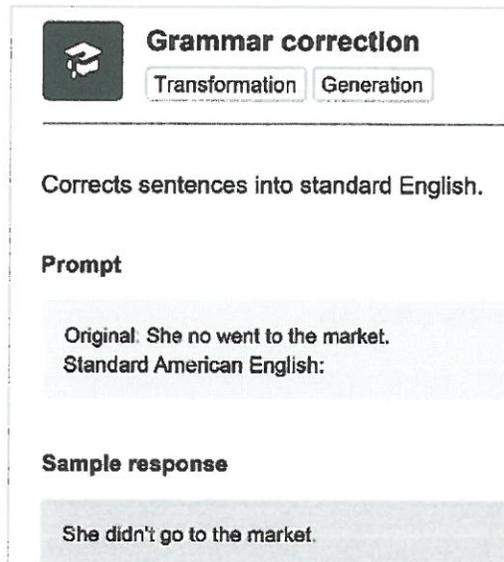


그림 7.5 프롬프트 교정 샘플 응답

이 챕터의 “온라인에서 NLP 작업 실행하기” 섹션에서 했던 작업을 플레이그라운드로 이동해서 실행할 수 있다. Open in Playground 버튼을 눌러보자.



그림 7.6 플레이그라운드 열기 버튼

이 챕터의 구글 코랩 노트북에 있는 코드를 복사하여 붙여 넣기 해 API를 실행할 수 있다.

```
API request
davinci python Copy
1 import os
2 import openai
3
4 openai.api_key = os.getenv('OPENAI_API_KEY')
5
6 response = openai.Completion.create(
7     engine='davinci',
8     prompt='Original: She no went to the market.\nStandard
9     temperature=0.7,
10    max_tokens=50,
11    top_p=1.0,
12    frequency_penalty=0.0,
13    presence_penalty=0.0,
14    stop=['\n']
15 )
```

그림 7.7 Davinci 엔진을 사용하는 코드 실행하기

Getting\_Started\_GPT3.ipynb에 OpenAI GPT-3 구현 연습용 예제 열 개가 포함되어 있다.

각 예제는

- 먼저 OpenAI에서 제공하는 설명 링크를 제공한다. 링크는 셀 상단에 제공된다.
- 셀을 실행해 GPT-3 동작을 확인할 수 있다.

노트북에서 다음 열 가지 예제를 실행해 보자.

- 예제 1: 문법 교정
- 예제 2: 영어를 프랑스어로 번역
- 예제 3: 설명서를 제공하는 Instruct 시리즈
- 예제 4: 영화를 이모티콘으로 변환
- 예제 5: 프로그래밍 언어를 다른 언어로 변환. 예를 들어 파이썬을 자바스크립트로 변환하는 것을 가리킨다.

주의할 점: 이 예제를 실행하는 데에 OpenAI에서 특별한 권한을 얻어야 할 수 있다. 코드 생성기인 Davinci 코덱스 엔진을 사용하기 때문이다. 이 예제가 노트북에서 실행되지 않는다면 OpenAI에 연락해 코덱스 접근 권한을 요청하자.

- 예제 6: 고급 트윗 분류기
- 예제 7: 질의응답(Q&A)
- 예제 8: 텍스트 요약하기
- 예제 9: 비정형 데이터 파싱하기
- 예제 10: 시간 복잡도 계산하기

예제(<https://platform.openai.com/examples>) 페이지에서 다른 많은 작업을 실행할 수 있다.

이제 GPT-2와 GPT-3의 출력을 비교해 보자.

사전 학습 GPT-2 모델과 커스터마이징 학습한 GPT-2 모델에 입력한 예제를 GPT-3에 입력하면 어떤 결과를 도출할까? 이런 궁금증을 해소해 보자.

GPT-2 모델에 사용한 예제: 인간의 이성인 인식의 한 영역에서 자신의 본성에 의해 제시되는 질문을 거부할 수 없지만 마음의 모든 기능을 초월하기 때문에 대답할 수 없는 질문을 고려하도록 요청받습니다.

GPT-3 응답: 이것이 의미하는 바는 우리의 경험에는 이성을 초월하는 것들이 있다는 것입니다. 이성이 나쁜 지침이거나 불충분하다는 것이 아니라 이성의 손이 닿지 않는 것들이 있다는 것입니다. 그 예로 신의 존재를 들 수 있습니다.

GPT-3의 응답은 놀랍다. 응답은 깊은 철학적 성찰을 포함하고 있는 텍스트의 의미를 설명하고 있다.

이 장에서 트랜스포머는 추상적 추론까지 가능하다는 점을 확인할 수 있다. 이러한 역량은 빠르게 변화하는 세상에서 미세한 의사 결정을 내리는 데 도움이 될 수 있다.

OpenAI GPT-3는 완전히 학습한 모델이다. 하지만 GPT-3는 미세 조정이 가능하다. 어떻게 하는지 살펴보자.

이 챕터에서 로직을 학습하기 위해 GPT-3를 미세 조정하는 방법을 설명한다. 트랜스포머는 인간 수준의 언어를 이해하기 위해 논리, 추론 및 함의(entailment)를 학습해야 한다.

#### NOTE

미세 조정은 GPT-3를 나만의 애플리케이션으로 만들고, 요구 사항에 맞게 커스터마이징하는 데 있어 핵심이다. 애플리케이션의 편견(bias)을 없애고 원하는 것을 가르치고 AI에 발자국을 남길 수 있다. 자유로운 AI 사용을 누리기 위한 티켓과도 같다.

이 섹션에서는 `kantgpt.csv`를 사용해 임마누엘칸트 작품을 GPT-3가 학습하게 한다. **4장, RoBERTa 모델 처음부터 사전 학습하기**에서 BERT 타입 모델 학습에서 사용한 파일과 비슷한 파일을 사용한다.

GPT-3의 미세 조정을 익히면 다른 유형의 데이터를 사용해 특정 도메인, 지식 그래프 및 텍스트를 학습시킬 수 있다.

OpenAI는 GPT-3 엔진을 미세 조정할 수 있는 효율적이고 문서화가 잘된 서비스를 제공한다. 이 장의 7.2.1 10억 개 파라미터를 가진 트랜스포머 모델 부상 섹션에서 볼 수 있듯이, 이 서비스는 GPT-3 모델을 다양한 유형의 엔진으로 학습시켰다.

Davinci 엔진은 강력하지만 비용이 비쌀 수 있다. Ada 엔진은 GPT-3를 탐색하기에 더 저렴하면서도 충분한 결과를 제공한다.

GPT-3 미세 조정은 두 단계로 이루어진다.

- 데이터 전처리하기
- GPT-3 모델 미세 조정하기

## 7.7.1 데이터 전처리하기

깃허브 7장 디렉터리에 존재하는 Fine\_Tuning\_GPT\_3.ipynb 파일을 열어보자.

OpenAI는 데이터 전처리 과정을 자세히 문서화해 두었다.

(<https://platform.openai.com/docs/guides/fine-tuning/prepare-training-data>)

### 7.7.1.1 1단계: OpenAI 설치하기

1단계는 openai를 설치하고 임포트한다.

```
try:
    import openai
except:
    !pip install openai
    import openai
```

설치가 완료되면 런타임을 다시 시작하고 셀을 다시 실행하여 openai를 임포트(import) 했는지 확인한다.

```
import openai
```

로그를 시각화할 수 있는 wandb를 설치할 수 있다.

```
try:
    import wandb
except:
    !pip install wandb
    import wandb
```

이제 API 키를 입력해 보자.

### 7.7.1.2 2단계: API 키 입력하기

2단계는 API 키를 입력하는 것이다.

```
openai.api_key="[YOUR_KEY]"
```

### 7.7.1.3 3단계: OpenAI의 데이터 전처리 모듈 활성화하기

먼저 파일을 불러온다. 이 섹션에서는 kantgpt.csv를 불러온다. kantgpt.csv는 원시 비정형 파일이다. OpenAI는 데이터 클리너를 내장하고 있다. 데이터 클리너는 전처리 단계별로 사용자에게 적용 여부를 질문한다.

OpenAI는 파일이 CSV 파일임을 감지하고 이를 JSONL 파일로 변환한다. JSONL은 각 줄마다 구조화된 텍스트를 포함한다.

OpenAI는 단계별로 적용하기로한 전처리 단계를 추적한다.

```
Based on the analysis we will perform the following actions:
- [Necessary] Your format 'CSV' will be converted to 'JSONL'
- [Necessary] Remove 27750 rows with empty completions
- [Recommended] Remove 903 duplicate rows [Y/n]: y
- [Recommended] Add a suffix separator ' ->' to all prompts [Y/n]: y
- [Recommended] Remove prefix 'completion:' from all completions [Y/n]: y
- [Recommended] Add a suffix ending 'Wn' to all completions [Y/n]: y
- [Recommended] Add a whitespace character to the beginning of the
completion [Y/n]: y
```

OpenAI는 변환 결과를 kantgpt\_prepared.jsonl에 저장한다.

이제 GPT-3 미세 조정할 준비가 됐다.

## 7.7.2 GPT-3 미세 조정하기

노트북을 데이터 전처리용과 미세 조정용으로 분리할 수 있다.

### 7.7.2.1 4단계: OS 환경 만들기

4단계는 API 키를 입력한 os 환경을 만든다.

```
import openai
import os
os.environ['OPENAI_API_KEY']=[YOUR_KEY]
print(os.getenv('OPENAI_API_KEY'))
```

### 7.7.2.2 5단계: OpenAI의 Ada 엔진 미세 조정하기

5단계는 데이터 전처리 후 JSONL 파일을 사용하여 OpenAI Ada 엔진을 미세 조정한다.

```
!openai api fine_tunes.create -t "kantgpt_prepared.jsonl" -m "ada"
```

openai 모듈은 많은 API를 갖고 있다.

미세 조정을 도중에 중단한다면, 아래와 같은 명령으로 미세 조정을 이어갈 수 있다.<sup>22</sup>

```
!openai api fine_tunes.follow -i [YOUR_FINE_TUNE]
```

<sup>22</sup> 역주. 미세 조정 작업명은 fine\_tunes.create를 실행 시 나타난 출력 "Created fine-tune: ft-....."에서 확인이 가능하다.

### 7.7.2.3 6단계: 미세 조정된 모델과 상호 작용하기

6단계는 미세 조정된 모델과 상호 작용하는 단계이다. 임마누엘 칸트가 작성한 법한 시퀀스를 프롬프트에 입력해 보자:

```
!openai api completions,create -m ada:[YOUR_MODEL INFO] "Several concepts  
are a priori such as"
```

미세 조정작업 생성에서 출력 뒷부분에 [YOUR\_MODEL INFO]가 출력된다. 이것을 입력하여 텍스트 완성 명령을 실행시킨다.

텍스트 완성 결과는 꽤 설득력이 있다.

```
Several concepts are a priori such as the term freedom and the concept of  
_free will_substance
```

GPT-3를 미세 조정해보았다. API로 트랜스포머를 이해하고 AI 파이프라인을 설계하는 것이 얼마나 중요한지 볼 수 있었다. 이를 통해 AI 전문가 역할이 어떻게 변화하는지 살펴보자.

간단히 말해, 4차 산업혁명 시대 개발자 역할은 여러 분야를 넘나드는 AI 전문가가 되는 것이다. 개발자, 데이터 과학자, AI 전문가는 언어학, 비즈니스 목표, 주제별 전문 지식 등을 점진적으로 많이 배우게 될 것이다. 4차 산업혁명 AI 전문가는 여러 분야를 넘나드는 실질적인 지식과 경험을 바탕으로 팀을 이끌게 될 것이다.

트랜스포머를 구현할 때 세 가지 영역에서 인적 전문가가 반드시 필요하다.

- 도덕 및 윤리

4차 산업혁명 AI 전문가는 인간과 유사한 트랜스포머를 구현할 때 도덕적, 윤리적 관행을 준수하도록 보장해야 한다. 예를 들어, 유럽에서는 필요한 경우 자동화된 결정을 사용자에게 설명하도록 요구하는 엄격한 규정이 존재한다. 미국은 시민을 자동화된 편견으로부터 보호하기 위한 차별금지 법(anti-discrimination laws)을 시행하고 있다.

- 프롬프트 및 응답

사용자와 UI 개발자는 NLP 작업에 적합한 프롬프트를 생성하고, 트랜스포머 모델에 작업 수행 방법을 보여주고, 응답을 확인하는 방법을 설명해 줄 4차 산업혁명 AI 전문가를 필요로 한다.

- 품질 관리 및 모델 이해

하이퍼파라미터를 조정한 후에도 모델이 예상과 다르게 동작하면 어떻게 해야 할까? 14장, 블랙박스 트랜스포머 모델 해석하기에서 이러한 문제를 다뤄본다.

## 7.8.1 잠정 결론

잠정 결론을 두 가지 범주로 나누어 보자.(사실과 허구)

한 가지 사실은 OpenAI가 세계에서 가장 강력한 NLP 서비스 하나를 보유하고 있다는 것이다. 다른 사실은

- OpenAI 엔진은 모든 트랜스포머 모델을 찾을 필요도 없고, 사전 학습과 미세 조정이 필요 없는 강력한 제로샷 엔진이다
- 모델 학습에 사용한 슈퍼컴퓨터는 독보적이다
- 프롬프트를 잘 설계하면 놀라운 정도로 정확한 응답을 얻을 수 있다
- 이 장의 NLP 작업 구현은 초보 개발자도 수행할 수 있는 복사 및 붙여 넣기 작업만 필요했다.

많은 사람들이 AI가 데이터 과학자와 AI 전문가를 대체할 것이라고 믿는다. 과연 그럴까?

이 질문에 답하기 전에 먼저 이 장에서 실행한 예제에 대해 다음과 같은 질문을 스스로에게 해보자.

- 문장이 틀린지 어떻게 알 수 있나요?
- 사람이 직접 읽고 확인하지 않고 어떻게 정답인지 알 수 있을까요?
- 엔진은 이 작업이 문법 교정 작업이라는 것을 어떻게 알았나요?
- 응답이 잘못된 경우, 잘 설계한 사용자 인터페이스에서 프롬프트를 개선하거나 수동 모드로 되돌리기 위해 어떤 일을 했었나요?

이런 질문에 답하려면 규칙 기반, 자동화된 품질 관리 파이프라인 및 기타 여러 도구를 사용하여 사람이 직접 개입해야 한다.

앞의 사실들은 꽤나 설득력이 있다. 트랜스포머로 NLP 작업을 실행하면 대부분 개발이 거의 필요하지 않다. 하지만 인간은 여전히 필요하다. OpenAI 엔진은 인간을 대체하기 위한 것이 아니라 더 높은 수준의 만족스러운 작업을 수행하도록 돕기 위한 것이다. 즉, 제트기를 만들지 않고 비행할 수 있다는 것이다.

AI의 미래로 향하는 멋진 여정에서 독자 여러분만의 새롭고 매력적인 AI 전문가 역할을 탐험해서 이 챕터에 나온 질문에 답해보자.

이번 장을 한번 정리하고 다음 장으로 넘어가자.

슈퍼컴퓨터에서 파라미터 수십억 개를 학습하는 트랜스포머 모델의 새로운 시대를 살펴봤다. OpenAI의 GPT 모델은 대부분 NLP 개발팀이 도달할 수 있는 범위를 뛰어넘어 NLU를 발전시키고 있다.

API를 사용하거나 혹은 API 없이 온라인으로 GPT-3 제로샷 모델이 NLP 작업을 수행하는 방법을 살펴봤다. 구글 번역(Google Translate)의 온라인 버전은 이미 온라인에서 AI를 주류로 사용할 수 있는 길을 열었다.

GPT 모델의 설계를 살펴봤다. 모델들은 모두 오리지널 트랜스포머의 디코더 스택을 기반으로 구축됐다. 마스크드 어텐션 서브 층은 왼쪽에서 오른쪽으로 학습하는 철학을 계속한다. 그러나 강력한 계산 성능과 어텐션 서브 층 덕분에 매우 효율적으로 학습할 수 있다는 것을 살펴보았다.

그런 다음 345M 파라미터 GPT-2 모델을 텐서플로우(TensorFlow)로 구현했다. 목표는 학습한 모델과 상호 작용하여 어떤 기능까지 도달할 수 있는지 확인하는 것이었다. 컨텍스트가 출력을 조절하는 것을 확인했다. 하지만 칸트 데이터셋의 특정 입력의 경우 예상한 결과에 도달하지 못했다.

커스텀 데이터셋으로 117M 파라미터 GPT-2 모델을 학습시켰다. 비교적 적게 학습한 모델과 상호 작용은 흥미로운 결과를 보였다.

OpenAI의 API로 온라인 NLP 작업을 실행하고 GPT-3 모델을 미세 조정했다. 이 장에서 완전히 사전 학습한 트랜스포머와 그 엔진이 엔지니어의 도움을 거의 받지 않고도 많은 작업을 자동으로 수행할 수 있음을 보여주었다.

이는 앞으로 사용자가 더 이상 AI NLP 개발자, 데이터 과학자, AI 전문가를 필요로 하지 않는다는 것을 의미할까? 사용자가 작업 정의와 입력 텍스트를 클라우드 트랜스포머 모델에 업로드하고 결과를 다운로드하기만 하면 될까?

전혀 그렇지 않다. 4차 산업혁명 시대의 데이터 과학자와 AI 전문가는 강력한 AI 시스템의 파일럿으로 진화할 것이다. 이들은 입력한 데이터가 윤리적이고 안전한지 확인하는 역할이 더 중요해질 것이다. 또한 트랜스포머가 어떻게 구축됐는지 이해하고 AI 생태계의 하이퍼파라미터를 조정하게 될 것이다.

**8장 법률 및 금융 문서에 트랜스포머를 적용하여 요약하기**에서 멀티태스크 모델로서의 트랜스포머 모델을 한계까지 끌어올리고 새로운 영역을 개척해보자.

## 7.10 문제

01. 제로샷 방식은 파라미터를 한 번만 학습한다. 참 / 거짓
02. 제로샷 모델을 실행할 때 기울기(gradient) 업데이트가 수행된다. 참 / 거짓
03. GPT 모델은 디코더 스택만 갖는다. 참 / 거짓
04. 117M GPT 모델을 로컬 머신에서 학습하는 것은 불가능하다. 참 / 거짓
05. GPT-2 모델을 특정 데이터셋에 학습하는 것은 불가능하다. 참 / 거짓
06. GPT-2 모델은 텍스트 생성 시 조건문 사용이 불가능하다. 참 / 거짓
07. GPT-2 모델은 입력 컨텍스트를 분석해 콘텐츠 완성 결과를 반환할 수 있다. 참 / 거짓
08. GPU 8개 미만인 머신에서 345M 파라미터 GPT 모델과 상호 작용할 수 없다. 참 / 거짓
09. 285,000 CPU를 가진 슈퍼컴퓨터는 존재하지 않는다. 참 / 거짓
10. 수천 개 GPU를 가진 슈퍼컴퓨터는 AI의 판도를 바꾸고 있다. 참 / 거짓

## 7.11 참고 문헌

- OpenAI의 GPT-3 엔진: <https://beta.openai.com/docs/engines/engines>
- 제시 빅(Jesse Vig)의 BertViz 깃허브 저장소: <https://github.com/jessevig/bertviz>
- OpenAI의 슈퍼컴퓨터: <https://blogs.microsoft.com/ai/openai-azuresupercomputer/>
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin, 2017, Attention is All You Need: <https://arxiv.org/abs/1706.03762>
- Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, 2018, Improving Language Understanding by Generative Pre-Training: [https://cdn.openai.com/research-covers/language-unsupervised/language\\_understanding\\_paper.pdf](https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf)
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, 2019, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding: <https://arxiv.org/abs/1810.04805>
- Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, 2019, Language Models are Unsupervised Multitask Learners: [https://cdn.openai.com/betterlanguage-models/language\\_models\\_are\\_unsupervised\\_multitask\\_learners.pdf](https://cdn.openai.com/betterlanguage-models/language_models_are_unsupervised_multitask_learners.pdf)
- Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, Dario Amodei, 2020, Language Models are Few-Shot Learners: <https://arxiv.org/abs/2005.14165>
- Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, Samuel R. Bowman, 2019, SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems: <https://w4ngatang.github.io/static/papers/superglue.pdf>
- Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, Samuel R. Bowman, 2019, GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding: <https://arxiv.org/pdf/1804.07461.pdf>
- OpenAI의 GPT-2 깃허브 저장소: <https://github.com/openai/gpt-2>
- N. Shepperd의 깃허브 저장소: <https://github.com/nshepperd/gpt-2>
- 커먼 크롤(Common Crawl) 데이터셋: <https://commoncrawl.org/big-picture/>

# 08장

---

## 법률 및 금융 문서에 트랜스포머를 적용하여 요약하기

---

- 8.1 범용 텍스트 투 텍스트 모델 디자인하기
- 8.2 T5를 사용해 요약하기
- 8.3 GPT-3로 요약하기
- 8.4 정리하기
- 8.5 문제
- 8.6 참고 문헌

앞선 장들에서 여러 트랜스포머 생태계의 아키텍처 학습, 미세 조정 및 사용에 대해 탐색했다. 7장, GPT-3 엔진을 사용한 초인간 트랜스포머 등장에서 OpenAI는 미세 조정이 필요 없고 개발이 필요하지 않으며 단 몇 줄의 코드로 구현 가능한 제로샷(zero-shot) 모델을 실험하기 시작했다는 사실을 확인했다.

이러한 혁신의 근간 개념은 트랜스포머가 기계에게 언어를 인간과 같은 방식으로 이해하고 표현하도록 가르쳐왔기 때문이다.

2019년 라펠(Raffel et al.)은 트랜스포머 메타 모델을 디자인했다. 모든 자연어 처리 문제는 텍스트-투-텍스트(text-to-text) 함수로 나타낼 수 있다. 즉, 모든 종류의 자연어 처리 작업은 일종의 텍스트 맥락이 필요하며 이를 이용하여, 어떤 형태의 텍스트 응답을 생성한다는 것이다.

자연어 처리 작업을 텍스트-투-텍스트 표현으로 나타내면 트랜스포머를 이해할 수 있는 고유한 시각을 얻을 수 있다. 텍스트-투-텍스트 접근법으로 학습 및 미세 조정 단계에서 전이 학습을 통해 언어를 학습할 수 있다.

라펠(Raffel et al.)은 이 접근 방식을 사용해 텍스트-투-텍스트 전이 트랜스포머(Text-To-Text Transfer Transformer)를 개발했으며, 앞 글자를 따서 T5 모델이라고 부른다.

이번 장에서는 T5 트랜스포머 모델의 개념과 아키텍처를 살펴보고, 그다음에는 허깅페이스(Hugging Face)를 사용하여 문서 요약에 T5를 적용해 보겠다.

마지막으로, 우리는 텍스트-투-텍스트 접근 방식을 사용해 GPT-3 엔진에 텍스트 맥락을 제공하고 이를 통해 추론하도록 할 것이다. 제로샷 응답이 완벽하지는 않지만 충격적일 정도로 높은 성능을 보여주는 것을 확인할 수 있다.

이 장에서는 다음 주제들을 다룬다.

- 텍스트-투-텍스트 트랜스포머 모델
- T5 모델의 아키텍처
- T5 사용 방법
- 트랜스포머 모델의 훈련(training)에서 학습(learning)으로의 진화<sup>23</sup>

23 역주. training은 책 전반에서 학습으로 번역하지만, 이 장에서 모델이 수동적으로 언어를 배우는 것을 넘어 능동적으로 추론하는 제로샷 학습까지 할 수 있다는 것을 의미한다

- 허깅페이스 트랜스포머 모델
- T5 모델 구현
- 법률 텍스트 요약
- 금융 텍스트 요약
- 트랜스포머 모델의 한계
- GPT-3 사용

첫 번째 단계로 라펠이 정의한 텍스트-투-텍스트 전이 방법을 탐색해 보자.

구글의 NLP 기술 혁명은 2017년 바스와니가 발표한 트랜스포머 모델에서 시작됐다. 「Attention is All You Need」는 RNN, CNN을 비롯해서 NLP 분야에 대해 30년 이상 축적된 인공 지능에 관한 상식을 뒤엎어버렸다. 이는 우리가 NLP/NLU의 석기 시대에서 오랜 시간이 지난 진화를 통해 21세기로 나아갈 수 있게 해주었다.

7장, GPT-3 엔진을 사용한 초인간 트랜스포머 등장은 구글의 바스와니가 발표한 오리지널 트랜스포머와 OpenAI의 브라운이 발표한 GPT-3 트랜스포머 사이에 있었던 혁신적인 기술 발전과정을 요약하고 있다. 오리지널 트랜스포머는 어텐션만으로도 NLP/NLU 작업에서 높은 성능을 낼 수 있다는 것을 보여줬다.

OpenAI의 두 번째 혁명은 GPT-3를 통해, 미세 조정된 사전 학습 모델에서 더 나아가 미세 조정없이 몇 가지 학습 데이터만으로(few-shot) 모델 학습이 가능하다는 것을 보여줬다. 즉, 기계가 우리 인간이 하는 것처럼 언어를 학습하고 하위 작업에 적용할 수 있다.

T5 모델의 의미를 이해하기 위해서는 이 두 가지 혁신적 기술을 이해하는 것이 필수적이다. 첫 번째 혁신적 기술은 어텐션 기법이었다. 두 번째 혁신적 기술은 기계에게 언어를 이해(NLU)하게 가르침을 주고, 그런 다음 우리가 하는 것처럼 NLP 문제를 해결하게끔 하는 기법이다.

2019년 구글은 OpenAI와 마찬가지로 트랜스포머를 추상적인 자연어 이해의 수준으로 끌어올릴 수 있는 방법에 대해 고민하고 있었다.

이러한 기술적 혁명들은 너무나 혁신적이었다. 이제는 소스 코드를 벗어나, 트랜스포머를 보다 높은 수준에서 분석할 때다.

라펠은 텍스트-투-텍스트 모델의 콘셉트를 제안하고 구현했다.

트랜스포머 발전에 기여한 두 번째 혁신적 기술을 이해하기 위해 추상 모델(Abstract models)에 대해 알아보자.

## 8.1.1 텍스트-투-텍스트 트랜스포머 모델의 등장

라펠은 하나의 목표를 가지고 선구자로서 여정을 시작했다. 통합된 텍스트 투 텍스트 트랜스포머 전이 학습의 한계 탐구하기, 이 접근 방식을 사용하는 구글 팀은 처음부터 오리지널 트랜스포머의 기본 아키텍처를 수정하지 않을 것이라고 강조했다.

그 당시 라펠은 기술이 아닌 개념에 초점을 맞추고 싶어 했다. 따라서, 파라미터 혹은 층  $n$ 개를 가진 최신 은 총알(silver bullet) 트랜스포머 모델을 만드는 것에 관심이 없었다. T5 팀은 트랜스포머가 언어 이해에 얼마나 높은 성능을 보일 수 있을지 알아내고 싶었다.

인간은 언어를 배우고 그 지식을 전이 학습을 통해 다양한 NLP 작업에 적용할 수 있다. T5 모델의 핵심 개념은 우리 인간처럼 일을 할 수 있는 추상적인 모델을 찾는 것이었다.

의사소통을 할 때, 우리는 항상 시퀀스(A)를 시작으로 시작해 또 다른 시퀀스(B)로 의사소통을 이어 나가게 된다. B는 또 다른 시작이 되어, 그림 8.1에 나와 있는 것처럼 다른 시퀀스를 이끌게 된다.

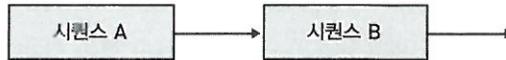


그림 8.1 시퀀스-투-시퀀스로 표기한 의사소통 방식

우리는 체계화된 소리로 음악을 통해 의사소통을 하기도 한다. 체계화된 몸의 움직임으로 춤을 추며 의사소통하기도 한다. 그리고 조화롭게 배열된 형태와 색상으로 그림을 그리며 스스로를 표현하기도 한다.

우리는 텍스트라고 부르는 여러 단어로 의사소통한다. 텍스트를 이해하려 할 때, 우리는 문장 안에 있는 모든 단어들에게 모든 방향에서 주의를 기울인다. 각 단어의 중요성을 이해하려고 노력한다. 문장을 이해하기 어려울 때는, 한 단어를 문장 안의 나머지 키워드(keywords)와 비교(query)해 가며 각 단어의 중요성(values and attention)을 결정한다. 이와 같은 방식이 트랜스포머의 어텐션 층<sup>24</sup>이다.

잠시 생각해 보자. 이 메커니즘은 속임수같이 간단해 보인다. 그럼에도 불구하고, RNNs, CNNs 등과 관련된 사고과정을 뒤엎는데 35년 이상이 걸렸다!

T5가 배우고 진전하며 때로는 우리가 더 잘 생각할 수 있도록 도와주는 것을 지켜보는 것은 매우 매력적이다!

24 Key, query, value로 구성되어있다.

시퀀스 내 모든 토큰에 동시에 주의를 기울이는 어텐션 층의 기술적 혁명은 T5의 개념적 혁명으로 이어졌다.

T5 모델은 텍스트-투-텍스트 전이 트랜스포머(Text-To-Text Transfer Transformer)로 간주할 수 있다. 따라서 모든 NLP 작업은 텍스트-텍스트 문제로 생각할 수 있다.

## 8.1.2 작업에 특화된 입력 형식 대신 접두사 사용하기

라펠은 여전히 하나의 문제를 해결해야 했다. 작업에 특화된 입력 형식 통합하기. 이를 위해 트랜스포머에 제출된 모든 작업에 대해 하나의 입력 형식을 갖는 방법을 찾는 아이디어를 제시했다. 이렇게 하면 하나의 텍스트-투-텍스트 형식으로 모든 종류의 작업에 대해 모델 파라미터를 훈련할 수 있다.

구글 T5 팀은 간단한 해결책을 제안했다. 입력 시퀀스에 접두사를 추가하는 것이다. 만약, 과거 몇몇 천재들에 의해 접두사가 발명되지 않았다면 언어마다 수천 개의 단어가 추가로 필요했을 것이다. 예를 들어, 'Pre' 접두사가 없었다면 선사시대가(Precambrian), 선사시대(Prehistoric), 선불(Prepayment) 등 수천 개의 다른 단어를 설명하기 위한 별도의 단어를 찾아야 했을 것이다.

라펠은 입력 시퀀스에 접두사를 추가하는 것을 제안했다. T5의 접두사는 [CLS]와 같은 몇몇 트랜스포머 모델에서 분류를 위해 사용하는 태그나 표시가 아니다. 대신, T5의 접두사는 트랜스포머가 해결해야 할 작업의 본질을 포함하고 있다. 아래 예와 같이 의미를 전달하기 위해 사용될 수 있다.

- **6장, 트랜스포머를 사용한 기계 번역**에서 했던 것처럼 번역 작업 수행을 위해 다음과 같이 명령할 수 있다.  
translate English to German: + [sequence]
- **3장, BERT 모델 미세 조정하기**에서 BERT 모델을 미세 조정할 때 했던 것처럼 The Corpus of Linguistic Acceptability (CoLA, 언어 적합성 말뭉치) 전달을 위해 다음과 같이 명령할 수 있다.  
cola sentence + [sequence]
- **5장, 트랜스포머를 사용한 다운스트림 NLP 작업**에서 설명한 것처럼 자연어 추론(inference)과 함의(entailment)는 비슷한 문제다. 의미론적 텍스트 유사성 벤치마크(Semantic Textual Similarity Benchmarks)를 위해 다음과 같이 명령할 수 있다.  
stsb sentence 1:+[sequence]
- 이번 장의 T5를 사용해 요약하기에서 풀게 될 요약 문제들에 대해 다음과 같이 명령할 수 있다.  
summarize + [sequence]

그림 8.2처럼 NLP의 다양한 작업을 하나의 형식으로 나타낼 수 있다.



그림 8.2 트랜스포머 모델의 입력 형식 통합하기

그림 8.3에서 보이듯이, 통합 입력 형식을 사용해 T5에서 해결해야 하는 어떤 문제에 대해서도 트랜스포머 모델이 결과 시퀀스를 생성하도록 할 수 있다. 그 결과, 많은 NLP 작업의 입력과 출력이 통합될 수 있었다.

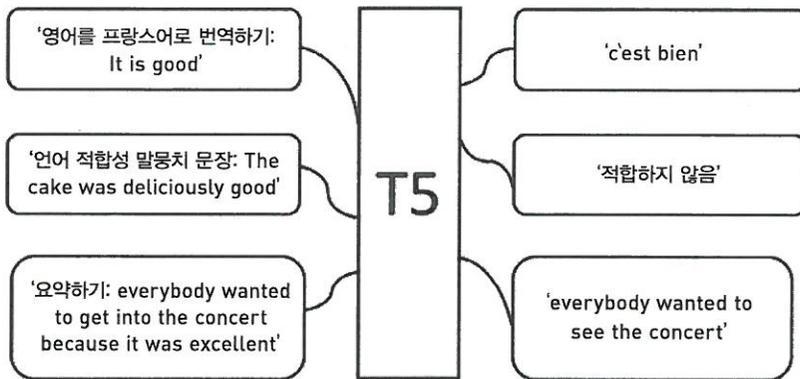


그림 8.3 T5의 텍스트-텍스트 프레임워크

이와 같은 통합 과정을 거침으로써 다양한 작업에 대해 동일한 모델, 하이퍼파라미터 및 옵티마이저를 사용할 수 있게 되었다.

지금까지는 표준 텍스트-텍스트 입력-출력 형식에 대해 알아보았다. 이제 T5 트랜스포머 모델의 구조를 살펴보자.

### 8.1.3 T5 모델

라펠은 텍스트 출력을 얻기 위한 표준 입력 형식을 설계하는 데 초점을 맞췄다. 구글 T5 팀은 인코더만 사용한 BERT나 디코더만 사용한 GPT와 같은 오리지널 트랜스포머에서 파생된 새로운 아키텍처를 시도하고 싶지 않았다. 대신, NLP 작업을 표준 형식으로 정의하는 데 중점을 두었다.

그들은 그림 8.4에서 볼 수 있듯이 2장, 트랜스포머 모델 아키텍처 살펴보기에서 정의한 오리지널 트랜스포머 모델을 사용하기로 결정했다.

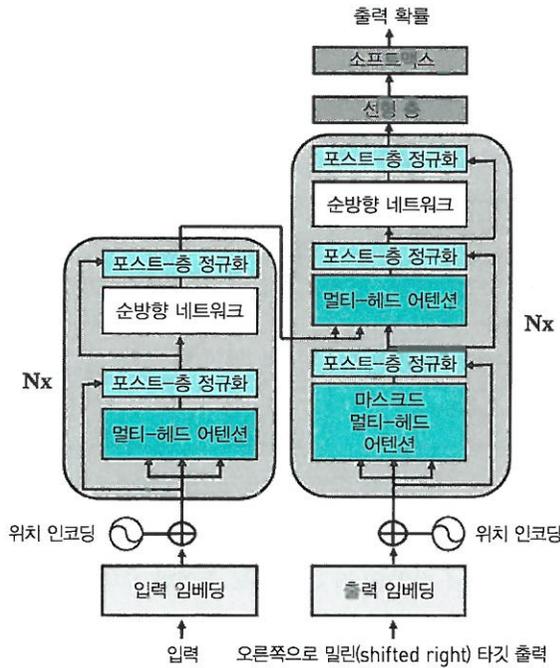


그림 8.4 T5 모델에서 사용한 오리지널 트랜스포머의 구조

라펠은 오리지널 트랜스포머의 아키텍처와 용어 대부분을 유지했다. 그러나 몇 가지 주요 측면에 초점을 맞추어, 약간의 어휘 및 기능적인 변경을 가했다. 다음은 T5 모델의 주요 특징 중 일부다.

- 모델에서 인코더와 디코더는 그대로 유지된다. 그러나 인코더와 디코더 층은 “블록”이 되고, 서브 층(sublayer)은 “서브컴포넌트(subcomponents)”가 되어 셀프-어텐션(self-attention) 층과 순방향 네트워크(feedforward network)를 포함하게 된다. “블록”과 “서브 컴포넌트”라는 용어를 레고의 구성 요소처럼 간주한다면, 모델을 구축하기 위한 작업은 “블록”, 조각 및 구성 요소를 조립하는 것으로 생각할 수 있다. 트랜스포머의 구성 요소는 다양한 조립에 사용할 수 있는 표준 빌딩 블록이다. 트랜스포머 모델의 기본적인 빌딩 블록을 이해하면 어떤 트랜스포머 모델도 이해할 수 있다. 이 내용은 **2장, 트랜스포머 모델 아키텍처 살펴보기**에서 다른 내용이다.
- 셀프-어텐션은 2장에서 본 것처럼 단어 집합에 대해 작업을 수행하므로 “순서에 독립적”이다. 셀프-어텐션은 재귀가 아닌 행렬의 내적(dot product)을 사용한다. 이는 시퀀스 내 각 단어와 다른 단어 간의 관계를 탐색한다. 또한 내적을 수행하기 전에 단어의 임베딩에 위치 인코딩이 추가된다.
- 오리지널 트랜스포머는 입력 데이터에 사인 및 코사인 신호를 적용했거나 학습된 위치 임베딩을 사용했다. T5는 입력에 임의의 위치 정보를 추가하는 대신 상대 위치 임베딩을 사용한다. T5에서 위치 인코딩은 셀프 어텐션을 사용해 데이터의 관계를 비교한다. 자세한 내용은 이 장의 참고문헌 섹션에 있는 2018년 쇼(Shaw et al.)의 논문을 참조하기 바란다.
- 위치 임베딩은 모델의 모든 층에서 공유되며 다시 계산된다.

우리는 텍스트-투-텍스트 접근법을 통해 T5 트랜스포머 모델의 입력을 표준화했다. 이제 T5를 사용하여 문서를 요약해 보자.

## 8.2

# T5를 사용해 요약하기

NLP 요약 작업은 텍스트의 핵심 부분을 추출하는 작업이다. 이 섹션에서는 이 챕터에서 사용할 허깅페이스를 소개하고 T5-large 트랜스포머 모델을 초기화해 볼 것이다. 마지막으로, 법률 및 기업 문서를 포함한 모든 문서를 요약하는 방법을 살펴보겠다.

우선 허깅페이스의 프레임워크를 소개하겠다.

### 8.2.1 허깅페이스

허깅페이스는 트랜스포머를 더 높은 수준에서 구현하기 위한 프레임워크를 디자인했다. 우리는 **3장, BERT 모델 미세 조정하기**에서 BERT 모델을 미세 조정하고 **4장, RoBERTa 모델 처음부터 사전 학습하기**에서 RoBERTa 모델을 처음부터 사전 학습하기 위해 허깅페이스를 사용했다.

지식을 확장하기 위해, 우리는 **6장, 트랜스포머를 사용한 기계 번역에서 트랙스(Trax)**와 같은 다른 접근법을 탐구하고, **7장, GPT-3 엔진을 사용한 초인간 트랜스포머 등장**에서는 OpenAI의 모델을 알아보았다. 이 장에서는 허깅페이스 프레임워크를 다시 사용하고 온라인 리소스에 대해 더 자세히 설명한다. 마지막으로, GPT-3 엔진의 독특한 잠재력에 대해 설명할 것이다.

허깅페이스는 프레임워크 내에서 모델, 데이터셋, 지표라는 세 가지 주요 리소스를 제공한다.

#### 8.2.1.1 허깅페이스 트랜스포머 리소스

이 하위 섹션에서는 이 챕터에서 구현할 T5 모델을 선택할 것이다. 그림 8.5에서 볼 수 있듯이 다양한 모델을 허깅페이스 모델 페이지에서 찾을 수 있다.

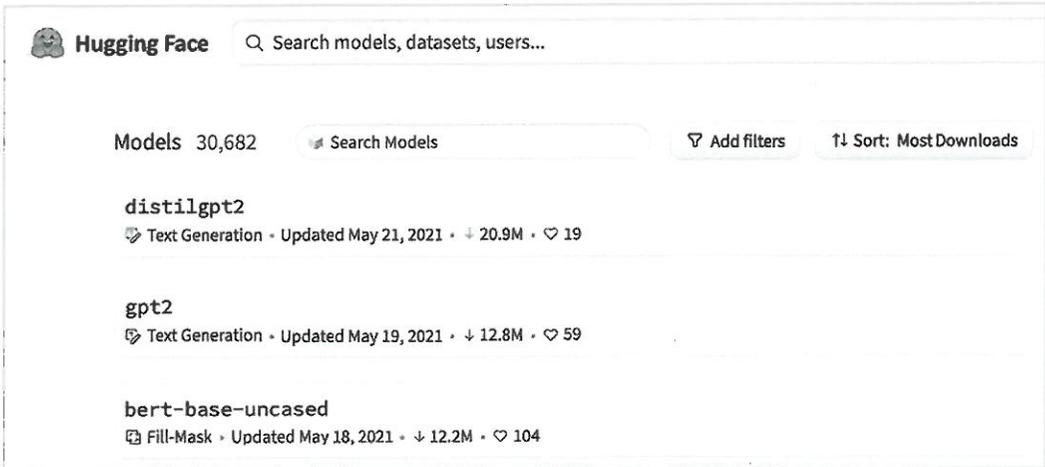


그림 8.5 허깅페이스 모델들

<https://huggingface.co/models>에서 모델을 검색할 수 있다. 우리의 경우, 구글 코랩에서 원활하게 실행할 수 있는 t5-large 모델을 찾고 있다.

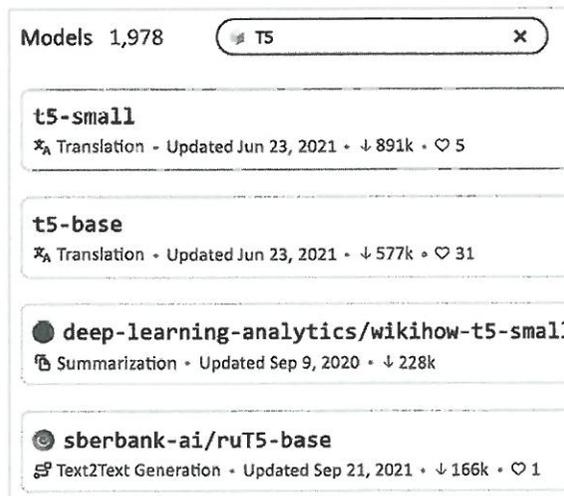


그림 8.6 T5 모델 검색하기

사용가능한 T5 트랜스포머는 다음과 같다.

- 기본(base)은 기준 모델로, 계층 12개와 파라미터 약 2억 2천만 개를 가진 BERT<sub>base</sub>와 유사하게 설계되었다.
- 작은(small) 모델은 계층 6개와 파라미터 약 6천만 개를 가지고 있다.
- 큰(large) 모델은 계층 12개의 파라미터 약 7억 7천만 개를 가진 BERT<sub>large</sub>와 유사하게 설계되었다.
- 3B와 11B는 각각 약 28억 개와 110억 개의 파라미터를 가진 24개 계층의 인코더와 디코더를 사용한다.

BERT<sub>BASE</sub>와 BERT<sub>LARGE</sub>에 대한 더 자세한 설명은 3장, BERT 모델 미세 조정하기에서 살펴볼 수 있다.

여기서는, t5-large 를 사용해 볼 것이다.

```
How to use from the  /transformers library

from transformers import AutoTokenizer, AutoModelWithLMHead

tokenizer = AutoTokenizer.from_pretrained("t5-large")

model = AutoModelWithLMHead.from_pretrained("t5-large")
```

그림 8.7 허깅페이스 모델을 사용하는 방법

그림 8.7은 우리가 작성할 코드에서 모델을 어떻게 사용하는지 보여준다. 또한 모델의 파일 목록과 기본 설정 파일을 살펴볼 수 있다. 이 장의 T5-large 트랜스포머 모델 초기화 섹션에서 모델을 초기화할 때 설정 파일을 살펴볼 것이다. 허깅페이스는 또한 데이터셋과 지표를 제공한다.

- 데이터셋은 모델을 학습하고 테스트하는 데 사용할 수 있다. <https://huggingface.co/datasets>
- 지표 리소스는 모델의 성능을 측정하는 데 사용할 수 있다. <https://huggingface.co/metrics>

데이터셋과 지표는 NLP에서 일반적으로 쓰이는 것을 사용할 것이다. 이 장에서는 이러한 데이터셋이나 지표를 구현하지 않을 것이다. 대신 텍스트를 요약하는 방법에 초점을 맞출 것이다.

T5 트랜스포머 모델을 초기화하는 것으로 시작하자.

## 8.2.2 T5-large 트랜스포머 모델 초기화

이 하위 섹션에서는 T5-large 모델을 초기화할 것이다. 깃허브의 이 장의 디렉터리에 있는 Summarizing\_Text\_with\_T5.ipynb 노트북을 열어보자.

그럼, T5로 시작해 보자.

### 8.2.2.1 T5로 시작해 보기

이 섹션에서는 허깅페이스 프레임워크를 설치한 다음 T5 모델을 초기화할 것이다.

우선, 허깅페이스 트랜스포머 모듈을 설치해보자.

```
!pip install transformers
```

#### NOTE

참고: 허깅페이스는 변화하는 시장 상황에 맞춰 지속적으로 라이브러리와 모듈들을 업데이트한다. 만약 디폴트 버전이 동작하지 않는다면 다음과 같은 명령어로 특정 버전을 설치해야 할 수 있다. `!pip install transformers==[노트북에서 실행할 특정 버전]`

허깅페이스를 최대한 안정적으로 사용하기 위해 `sentencepiece`를 0.1.94 버전으로 고정했다.

```
!pip install sentencepiece==0.1.94
```

허깅페이스는 복제(clone)할 수 있는 깃허브 저장소를 가지고 있지만, 허깅페이스의 프레임워크를 통해 다양한 고수준의 트랜스포머 함수들을 사용할 수 있다.

모델을 초기화할 때 모델의 아키텍처를 표시할지 선택할 수 있다.

```
display_architecture=False
```

`display_architecture`를 `True`로 설정하면, 인코더 층, 디코더 층 및 순방향 서브 층의 구조가 표시된다. 프로그램은 이제 `torch`와 `json`을 가져온다.

```
import torch
import json
```

트랜스포머 작업을 한다는 것은 연구소에서 공유하는 다양한 트랜스포머 아키텍처와 프레임워크에 열려 있어야 한다는 것을 의미한다. 가능하다면, 파이토치와 텐서플로우를 사용하는 것을 추천한다. 중요한 것은 트랜스포머 모델의 추상화 수준(구체적인 작업 모델 또는 제로샷 모델)과 전반적인 성능이다.

토큰라이저, 생성 및 설정 클래스를 가져오도록 하자.

```
from transformers import T5Tokenizer, T5ForConditionalGeneration, T5Config
```

여기서는 T5-large 모델을 사용할 것이지만, 이번 장의 허깅페이스 섹션에서 살펴본 허깅페이스 목록에서 다른 T5 모델을 선택할 수 있다.

이제 T5-large 조건부 생성 모델과 T5-large 토큰라이저를 가져오도록 하겠다.

```
model = T5ForConditionalGeneration.from_pretrained('t5-large')
tokenizer = T5Tokenizer.from_pretrained('t5-large')
```

사전 학습된 토큰라이저를 초기화하는 것은 한 줄로 끝난다. 그러나 토큰화된 사전이 우리가 필요한 모든 어휘를 포함하고 있는지 증명할 수 있는 것은 아니다. 9장, 데이터셋에 적합한 토큰라이저에서 토큰라이저와 데이터셋의 관계를 조사할 것이다.

이제 'cpu'로 torch.device를 초기화한다. 이 예제는 CPU만으로도 충분하다. torch.device 객체는 torch 텐서가 할당될 장치이다.

```
device = torch.device('cpu')
```

T5 모델의 구조를 탐색할 준비가 되었다.

### 8.2.2.2 T5 모델 구조 탐색하기

이 하위 섹션에서는 T5-large 모델의 아키텍처와 설정을 탐색할 것이다.

display\_architecture==true이면, 모델의 설정을 볼 수 있다.

```
if display_architecture==True:
    print(model.config)
```

예를 들어, 모델의 기본 파라미터를 볼 수 있다.

```
.../...  
"num_heads": 16,  
"num_layers": 24,  
.../...
```

이 모델은 헤드 16개와 층 24개를 가진 T5 트랜스포머 모델이다.

또한, T5의 텍스트-투-텍스트 구현을 볼 수 있는데, 입력 문장에 접두사를 추가하여 수행할 작업을 트리거한다. 이 접두사는 모델의 파라미터를 수정하지 않고도 다양한 작업을 텍스트-투-텍스트 형식으로 표현할 수 있도록 한다. 우리의 경우, 접두사는 “summarization” 이다.

```
"task_specific_params": {  
  "summarization": {  
    "early_stopping": true,  
    "length_penalty": 2.0,  
    "max_length": 200,  
    "min_length": 30,  
    "no_repeat_ngram_size": 3,  
    "num_beams": 4,  
    "prefix": "summarize: "  
  }  
}
```

T5는 다음과 같은 기능을 수행한다.

- 가장 중요한 네 개의 텍스트 완성 예측을 확장하는 빔 서치(beam search) 알고리즘을 도입한다.
- 배치 당 num\_beam 문장이 완성되면 일찍 종료(early stopping)한다.
- no\_repeat\_ngram\_size 길이만큼 n-gram을 반복하지 않도록 한다.
- min\_length와 max\_length를 사용하여 샘플의 길이를 제어한다.
- 길이 페널티를 적용한다.

눈여겨볼 파라미터는 어휘 크기이다.

```
"vocab_size": 32128
```

어휘 크기는 중요한 주제이다. 어휘가 너무 많다면 모델이 적합하게 수렴하기 힘들다. 반면에 너무 적은 어휘는 NLP 작업을 왜곡할 수 있다. 9장, 데이터셋에 적합한 토큰라이저에서 이에 대해 더 탐구할 것이다.

또한, 모델을 단순히 출력하여 트랜스포머 스택의 세부 정보를 볼 수 있다.

```
if(display_architecture==True):  
    print(model)
```

예를 들어, 우리는 인코더 스택(0에서 23까지 번호가 매겨진)의 블록(층) 내부를 엿볼 수 있다.

```
(12): T5Block(  
  (layer): ModuleList(  
    (0): T5LayerSelfAttention(  
      (SelfAttention): T5Attention(  
        (q): Linear(in_features=1024, out_features=1024, bias=False)  
        (k): Linear(in_features=1024, out_features=1024, bias=False)  
        (v): Linear(in_features=1024, out_features=1024, bias=False)  
        (o): Linear(in_features=1024, out_features=1024, bias=False)  
      )  
      (layer_norm): T5LayerNorm()  
      (dropout): Dropout(p=0.1, inplace=False)  
    )  
    (1): T5LayerFF(  
      (DenseReluDense): T5DenseReluDense(  
        (wi): Linear(in_features=1024, out_features=4096, bias=False)  
        (wo): Linear(in_features=4096, out_features=1024, bias=False)  
        (dropout): Dropout(p=0.1, inplace=False)  
      )  
      (layer_norm): T5LayerNorm()  
      (dropout): Dropout(p=0.1, inplace=False)  
    )  
  )  
)
```

모델이 어텐션 서브 층에서 1,024개의 피처에 대한 작업을 수행하고, 순방향 서브 층 내부에서 4,096개의 피처로 계산하여 1,024개의 피처를 출력하는 것을 볼 수 있다. 트랜스포머의 대칭 구조는 모든 층에서 유지된다.

인코더 층, 디코더 층, 어텐션 서브 층, 순방향 서브 층 중 특정 부분을 선택하여 모델의 특정 측면을 살펴볼 수도 있다.

원하는 셀만 실행하여 모델의 특정 부분을 선택할 수도 있다.

```
if display_architecture==True:
    print(model.encoder)
if display_architecture==True:
    print(model.decoder)
if display_architecture==True:
    print(model.forward)
```

지금까지 T5 트랜스포머를 초기화해봤다. 이제 문서를 요약해 보자.

## 8.2.3 T5-large를 사용해 문서 요약하기

이 섹션에서는 어떤 텍스트든 요약할 수 있는 요약 함수를 만들 것이다. 법적 및 금융 관련 문서를 요약할 것이다. 마지막으로, 이 방식의 한계를 설명할 것이다.

우선 요약 함수를 만드는 것으로 시작해 보자.

### 8.2.3.1 요약 함수 만들기

먼저, `summarize`라는 요약 함수를 만들어보자. 이렇게 하면 요약하려는 텍스트를 함수에 보내기만 하면 된다. 이 함수는 두 개의 파라미터를 사용한다. 첫 번째 파라미터는 요약할 텍스트인 `preprocess_text`이다. 두 번째 파라미터는 요약된 텍스트의 최대 길이인 `ml`이다. 이 두 파라미터는 함수를 호출할 때마다 보내는 변수이다.

```
def summarize(text,ml):
```

허깅페이스는 준비된 요약 함수들을 제공한다. 하지만 필요할 때마다 이 작업을 커스터마이징하는 방법을 배우는 것을 추천한다.

문맥 텍스트에서 `\n` 문자를 제거한다.

```
preprocess_text = text.strip().replace("\n","")
```

그런 다음 T5 작업 접두사 `summarize`를 입력 텍스트에 적용한다.

```
t5_prepared_Text = "summarize: "+preprocess_text
```

T5 모델은 어떤 작업이든 접두사 + 입력 시퀀스의 통일된 구조를 사용한다.

이 구조는 간단해 보이지만, NLP 트랜스포머 모델을 범용 학습과 제로샷 다운스트림 작업에 더 용이하게 만든다.

준비된 텍스트를 출력해 보자.

```
print ("Preprocessed and prepared text: \n", t5_prepared_Text)
```

간단하다고 생각할 수 있겠지만, RNN과 CNN에서 트랜스포머로 가는 데에 35년 이상이 걸렸다. 그리고 특정 작업용으로 설계된 트랜스포머에서 미세 조정이 거의 필요하지 않은 멀티태스크 모델로 가기 위해서는 세계에서 가장 뛰어난 연구팀이 필요했다. 마침내 구글 연구팀은 해결하려는 NLP 문제를 접두사로 표현하는 표준 입력 형식을 만들었다. 이것은 상당한 업적이다!

출력에는 전처리된 텍스트가 포함되어 있다.

```
Preprocessed and prepared text:  
summarize: The United States Declaration of Independence
```

해결할 작업을 나타내는 접두사 `summarize`를 확인할 수 있다.

이제 텍스트는 토큰 ID로 인코딩되어 토치 텐서로 반환된다.

```
tokenized_text = tokenizer.encode(t5_prepared_Text, return_tensors="pt").  
to(device)
```

인코딩된 텍스트를 T5로 시작해 보기 섹션에서 설명한 T5 모델에 보내 요약 생성할 준비가 되었다.

```
# 요약하기  
summary_ids = model.generate(tokenized_text,  
    num_beams=4,  
    no_repeat_ngram_size=2,  
    min_length=30,  
    max_length=ml,  
    early_stopping=True)
```

빔의 수는 가져온 모델과 동일하지만, no\_repeat\_ngram\_size는 3 대신 2로 줄여졌다.

이제 생성된 출력은 토큰라이저를 사용하여 디코딩된다.

```
output = tokenizer.decode(summary_ids[0], skip_special_tokens=True)  
return output
```

지금까지 요약 함수를 가져오고, 초기화하고, 정의해 봤다. 이제 일반적인 주제로 T5 모델을 실험해 보자.

### 8.2.3.2 일반적인 주제 샘플

이 하위 섹션에서는 프로젝트 구텐베르크에서 가져온 텍스트를 T5 모델로 실행할 것이다. 샘플을 사용하여 요약 함수에 대한 테스트를 실행할 것이다. 다른 텍스트를 복사하여 붙여 넣거나 코드를 추가하여 텍스트를 로드할 수 있다. 또한 원하는 데이터셋을 불러오고 순회하며 요약을 호출할 수도 있다.

이 장의 목표는 T5가 어떻게 작동하는지를 보기 위해 몇 가지 샘플을 실행하는 것이다. 입력 텍스트는 미국의 독립선언서를 포함하는 프로젝트 구텐베르크 전자책의 시작 부분이다.

```
text = """
```

미국의 독립선언서는 1971년 초 구텐베르크 프로젝트가 처음으로 공개한 전자 텍스트이다. 열람을 위해서는 이메일로 전송된 지침서에 따라 테이프나 디스크팩을 직접 장착해야 했다. 디스크팩은 케이크 캐리어에 담긴 큰 케이크 크기였고 가격은 1500 달러였으며 용량은 5 메가 바이트였고, 이 파일은 1-2%의 크기를 차지했다. 두 개의 백업 테이프와 하나의 종이테이프에 보관되었다. 2001년 말까지 온라인에 10,000개의 파일을 올린다면, 2001년의 비슷한 가격대의 드라이브의 1-2% 정도면 충분하다.

```
"""
```

이제 `summarize` 함수를 호출하고 우리가 요약하고자 하는 텍스트와 요약의 최대 길이를 전달해 보자.

```
print("Number of characters:",len(text))
summary=summarize(text,50)
print("\n\nSummarized text: \n\n",summary)
```

출력은 입력된 텍스트 길이가 534자라는 것을 보여주며, 전처리된 원본 텍스트(ground truth)와 요약된 결과를 보여준다.

문자 수: 534

전처리된 텍스트:

요약: 미국의 독립선언서는...

요약된 내용:

미국의 독립선언서는 1971년 초 구텐베르크 프로젝트가 처음으로 출판한 전자 텍스트이다. 2001년 말까지 온라인에 10,000개의 파일을 올린다면, 2001년의 비슷한 가격대의 드라이브의 1-2% 정도면 충분하다. 미국의 독립선언서는 1971년 초 구텐베르크 프로젝트가 처음으로 공개한 전자 텍스트이다.

이제 T5를 사용하여 더 어려운 요약을 해보자.

### 8.2.3.3 권리장전 샘플

권리장전에서 가져온 샘플은 개인의 특별한 권리를 표현하기 때문에 더 어렵다.

```
#권리장전, v
```

```
text = ""
```

```
누구라도 대배심의 고발이나 공소 제기에 의하지 아니하고는 사형에 해당하는 죄나 중죄에 대하여 심문당해서는 아니 된다. 다만 전쟁 시나 공공의 위험이 발생했을 때 육해군이나 민병대에 복무 중인 경우는 예외로 한다. 또한 어느 누구도 동일한 범죄에 대하여 생명이나 신체의 위험에 두 번 처해져서는 아니 되고, 어느 형사 사건에서도 자신의 증인이 될 것을 강요받아서는 아니 되며, 적법절차에 의하지 아니하고 생명이나 자유 또는 재산이 박탈당해서는 아니 된다. 또 사유재산권은 정당한 보상 없이는 공익 목적을 위하여 수용되어서는 아니 된다.
```

```
"""
```

```
print("Number of characters: ", len(text))
```

```
summary=summarize(text,50)
```

```
print ("WnWnSummarized text: Wn",summary)
```

트랜스포머는 확률론적 알고리즘이므로, 실행할 때마다 출력이 달라질 수 있다는 것을 기억하자.

그렇다고 해도, T5는 입력 텍스트를 실제로 요약하지 않고 단순히 줄였다는 것을 볼 수 있다.

문자 수: 591

전처리된 텍스트:

요약: 누구라도 대배심의 고발이나.

요약된 텍스트:

누구라도 사형에 해당하는 죄나 중죄에 대하여 심문당해서는 아니 된다. 다만 전쟁 시나 공공의 위험이 발생했을 때 육해군이나 민병대에 복무 중인 경우는 예외로 한다.

이 예시는 트랜스포머 모델이나 다른 NLP 모델이 이런 텍스트를 만났을 때 마주하는 한계를 보여주는 중요한 샘플이다. 항상 잘 동작하는 샘플만 보여주고 사용자들이 트랜스포머가 우리가 직면하는 모든 NLP 문제를 해결했다고 믿게 해선 안 된다.

아마 더 긴 텍스트를 요약해야 하는 상황이 있을 수 있다. 다른 파라미터를 사용하는 것이 더 적합했을 수도 있다. 더 큰 모델을 사용했어야 했을지도 모른다. 아니면 T5 모델의 구조를 변경했어야 했을지도 모른다. 하지만, 얼마나 열심히 복잡한 텍스트를 NLP 모델로 요약하려고 하든 간에 항상 모델이 요약하지 못하는 문서가 있을 것이다.

모델이 작업에 실패할 때, 우리는 겸손해져야 하고 그것을 인정해야 한다. SuperGLUE 인간 수준 성능은 이기기 어렵다. 우리는 인내심을 가지고, 더 열심히 일하고, 트랜스포머 모델을 개선해야 한다. 아직 개선할 여지가 있다.

라펠은 그들의 T5에 대한 접근법을 설명하기 위해 통합 텍스트-투-텍스트 트랜스포머와 전이 학습의 한계 탐구(T5: Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer)라는 적절한 제목을 선택했다.

충분한 시간을 갖고 자신만의 예제로 실험해보자. 현대 자연어 처리의 선구자로서 전이 학습의 한계를 탐구하자! 때로는 흥미로운 결과를 발견할 수 있고, 때로는 개선이 필요한 부분을 찾을 수도 있다.

이제 기업법 예시를 한번 시도해 보자.

### 8.2.3.4 기업법 샘플

기업 법은 많은 법적 세부 사항을 포함하고 있어서 요약 작업이 상당히 까다로울 수 있다. 이 예시는 미국 몬태나 주의 기업 법 일부이다.

```
#몬태나 주 기업 법
```

```
#https://corporations.uslegal.com/state-corporation-law/montana-corporation-law/#:~:text=Montana%20Corporation%20Law,carrying%20out%20its%20business%20activities.
```

```
text = """몬태나 주의 법인 관련 법률에 따르면 법인은 모든 합법적인 목적을 위해 몬태나 주에 설립될 수 있다. 몬태나 주에서 법인은 사업 활동을 수행하는 데 자연인과 동일한 모든 권한을 갖는다. 법인은 법인 명의로 소송을 제기하거나 소송을 받을 수 있다. 이는 영속 계승권을 갖는다. 법인은 부동산 또는 동산에 대한 지분을 사고팔거나 취득할 수 있다. 또한 미국 내의 주, 영토 또는 지구, 또는 외국 국가에서 사업을 수행하고 운영하며 사무실을 가지고 권한을 행사할 수 있다. 법인은 다양한 직무에 임원 및 대리인을 임명하고 보수를 정할 수 있다.
```

```
법인의 이름은 "corporation" 또는 줄임말인 "corp."이라는 단어를 포함해야 한다. 기업의 이름은 같은 주에서 설립된 다른 법인의 이름과 혼동되지 않아야 하며, 해당 주에서 사업 거래를 하는 외국 법인이 채택한 임의의 이름과 혼동돼서도 안된다.
```

```
법인은 한 명 이상의 자연인이 정관을 작성하여 주무부처 장관에게 제출함으로써 설립된다. 이사의 자격은 정관이나 내규에 따라 정한다. 초기 이사들의 이름과 주소, 그리고 설립 목적은 정관에 명시되어야 한다. 정관에는 법인명, 발행 허가 주식 수, 법인이 수행하는 사업 성격에 대한 간단한 설명, 후임자가 선출될 때까지 이사들의 이름과 주소, 설립자의 이름과 주소가 있어야 한다. 주주들은 이사회를 변경할 권한이 있다.
```

```
"""
```

```
print("Number of characters:",len(text))
summary=summarize(text,50)
print ("WnWnSummarized text: Wn",summary)
```

결과는 만족스럽다.

문자 수: 1816

전처리된 텍스트:

요약: 몬태나 주의 법인 관련 법률에 따르면...

요약된 텍스트:

법인은 모든 합법적인 목적을 위해 몬태나 주에 설립될 수 있다. 법인은 법인 명의로 소송을 제기하거나 소송을 받을 수 있으며 이는 영구적으로 승계된다. 사업을 수행하고 지속적으로 운영하고 사무실을 가질 수 있다.

이번에는 T5가 텍스트를 요약하기 위한 몇 가지 필수적인 측면을 찾아내었다. 자신의 샘플로 결과를 확인해 보는 시간을 가져보자. 결과에 영향을 미치는 파라미터를 조정해 보는 것도 좋은 방법이다.

T5를 사용해 텍스트를 요약하는 방법을 구현했으니, 이번에는 OpenAI GPT-3 엔진을 사용해 요약 작업을 해볼 시간이다.

## 8.3

# GPT-3로 요약하기

T5 트랜스포머의 아키텍처를 살펴보았다. 이번에는 GPT-3 엔진이 하나의 텍스트에서 어떻게 동작하는지 살펴볼 것이다. 기업과 모델을 벤치마킹하는 것이 목표는 아니다. 4차 산업시대의 AI 전문가가 자연어 처리에 대한 폭넓은 지식을 갖도록 하는 것이 목표이다.

먼저, <https://openai.com/>에 가서 회원가입과 로그인을 한다.

그런 다음 예제 페이지(<https://platform.openai.com/examples>)로 이동하여 2학년용 요약(Summarize for a 2nd grader)을 선택한다.

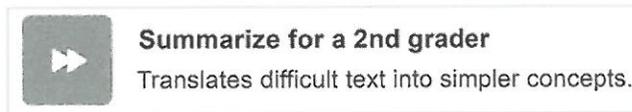


그림 8.8 GPT-3 예제 페이지

창이 열리면 프롬프트를 입력할 수 있다.

바로 앞의 섹션에서 사용한 예제 텍스트를 GPT-3에 넣어보자.

프롬프트는 P = E + T + S 형태다.

- E는 모델이 설명을 간단하게 만들도록 지시한다.  
*2학년 아이가 이 글의 의미를 물어보았다.*
- 텍스트 T는 이전 섹션과 동일하며 따옴표 안에 있다.  
*"" 기업에 대한 법률은 몬태나 주에서 어떤 합법적인 목적을 위해서도 기업을 설립할 수 있다고 규정한다. 몬태나 주에서는 기업이 사업 활동을 수행하는 데 자연인과 동일한 모든 권한을 갖는다. .../... 주주들은 이사회의 규모를 변경할 수 있는 권한이 있다. ""*
- S: 기대되는 작업 유형을 나타낸다.  
*그가 이해할 수 있는 2학년용 간단한 용어로 다시 말해 줘다: ""*

GPT-3 다빈치 엔진에 의해 생성된 결과는 상당히 설득력 있다.

*기업은 비즈니스다.*

*기업은 사람이다.*

*기업은 사람이 할 수 있는 모든 일을 할 수 있다.*

*기업은 소송을 당할 수 있다.*

*기업은 물건을 사고팔고 교환할 수 있다.*

*기업은 소유주의 이름과 다른 고유한 이름이 있다.*

*기업은 사람들이 비즈니스를 시작하고, 정부에 서류를 제출하여 공식적으로 만들어진다.*

T는 유지하고 E와 S를 수정할 수 있다.

- E: 다음과 같이 변경할 수 있다.

*내 MBA 대학생이 이 글의 의미를 물어보았다.*

- S: 다음과 같이 변경할 수 있다.

*MBA 대학에 재학 중인 그를 위해 바꾸어 말했다.*

GPT-3 엔진은 이전 요청의 응답을 생성했지만 대학생들에게 유용한 정보를 추가했다.

*기업은 비즈니스 결정을 책임지는 이사들을 둘 수 있다. 이사들은 주주들에 의해 선출된다.*

*주주들은 이사회 규모의 규모를 변경할 수 있다.*

GPT-3 모델은 매우 설득력 있으며 클라우드 인공지능의 부상을 대표한다. 16장, 트랜스포머 기반 코파일럿의 등장에서 프롬프트 요약에 대해 더 깊이 들어갈 것이다. 하지만, 그전에 더욱 많은 것들을 탐구할 필요가 있다.

이번 장에서는 T5 트랜스포머 모델이 오리지널 트랜스포머의 인코더와 디코더 스택의 입력을 표준화하는 방법을 살펴보았다. 오리지널 트랜스포머 아키텍처는 인코더와 디코더 스택의 각 블록(또는 층)이 동일한 구조를 가지고 있었다. 그러나 NLP 작업에 대한 표준화된 입력 형식을 가지고 있지는 않았다.

2018년 라펠은 텍스트-투-텍스트 모델을 정의하여 다양한 NLP 작업을 위한 표준 입력을 설계했다. 그들은 입력 시퀀스에 NLP 문제 유형을 해결하기 위한 접두사를 추가했다. 이로써 표준 텍스트-투-텍스트 형식이 만들어졌고, **Text-To-Text Transfer Transformer(T5)**가 탄생했다. 이는 거짓말처럼 간단하지만, 다양한 NLP 작업에 대해 동일한 모델과 하이퍼파라미터를 사용할 수 있게 해 주었다. T5의 발명은 트랜스포머 모델의 표준화 프로세스를 한 단계 더 나아가게 만들었다.

그다음, 우리는 모든 텍스트를 요약할 수 있는 T5 모델을 구현해 보았다. 학습용 데이터셋에 포함되지 않은 텍스트로 모델을 테스트해 보았다. 또한 헌법 및 기업 샘플로 모델을 테스트해 보았다. 그 결과는 흥미로웠지만, 라펠이 예상한 것과 같이 트랜스포머 모델의 한계도 발견할 수 있었다.

마지막으로, 우리는 GPT-3 엔진의 방법론과 계산 효율성의 엄청난 파워를 탐구해 보았다. 트랜스포머를 사용하는 것은 훌륭한 시도이다. 세계에서 가장 강력한 트랜스포머 엔진 중 하나를 가지고 있다면, 완벽하진 않지만 효과적으로 결과를 얻을 수 있다.

우리의 목표는 기업과 모델을 벤치마킹하는 것이 아니다. 4차 산업시대 AI 전문가가 트랜스포머에 대한 깊은 이해를 가지도록 하는 것이 목표이다.

다음 장인 **9장, 데이터셋에 적합한 토큰라이저**에서는 토큰라이저의 한계를 탐구하고 NLP 작업을 개선할 가능성이 있는 방법을 정의할 것이다.

## 8.5

## 문제

01. T5 모델은 BERT 모델과 같이 인코더 스택만 가지고 있다. 참 / 거짓
02. T5 모델은 인코더 스택과 디코더 스택을 모두 가지고 있다. 참 / 거짓
03. T5 모델은 절대적인 위치 인코딩이 아닌 상대적인 위치 인코딩을 사용한다. 참 / 거짓
04. 텍스트-투-텍스트 모델은 요약을 위해서만 디자인되었다. 참 / 거짓
05. 텍스트-투-텍스트 모델은 입력 시퀀스에 NLP 작업을 결정하는 접두사를 적용한다. 참 / 거짓
06. T5 모델은 각 작업마다 특정 하이퍼파라미터를 필요로 한다. 참 / 거짓
07. 텍스트-투-텍스트 모델의 장점 중 하나는 모든 NLP 작업에 대해 동일한 하이퍼파라미터를 사용한다는 것이다. 참 / 거짓
08. T5 트랜스포머는 순방향 네트워크를 가지고 있지 않다. 참 / 거짓
09. 허깅페이스는 트랜스포머를 보다 쉽게 구현할 수 있도록 하는 프레임워크이다. 참 / 거짓
10. OpenAI의 트랜스포머 엔진은 게임 체인저이다. 참 / 거짓

## 8.6

## 참고 문헌

- Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J. Liu, 2019, Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer: <https://arxiv.org/pdf/1910.10683.pdf>
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin, 2017, Attention is All You Need: <https://arxiv.org/abs/1706.03762>
- Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani, 2018, Self-Attention with Relative Position Representations: <https://arxiv.org/abs/1803.021>
- Hugging Face Framework and Resources: <https://huggingface.co/>
- 미국 법률, 몬태나 주 기업법: <https://corporations.uslegal.com/state-corporation-law/montana-corporation-law/#:-:text=Montana%20Corporation%20Law,carrying%20out%20its%20business%20activities>
- 토머스 제퍼슨(Thomas Jefferson)의 미국 독립선언: <https://www.gutenberg.org/ebooks/1>
- 미국 권리장전: <https://www.gutenberg.org/ebooks/2>

# 09장

---

## 데이터셋에 적합한 토큰나이저

---

- 9.1 데이터셋에 적합한 토큰나이저
- 9.2 특정 어휘가 포함된 표준 NLP 작업
- 9.3 GPT-3의 범위 살펴보기
- 9.4 정리하기
- 9.5 문제
- 9.6 참고 문헌

트랜스포머를 공부하다 보면 모델의 아키텍처와 학습 데이터셋에만 집중하기 쉽다. 우리는 지금까지 오리지널 트랜스포머를 살펴보고, BERT를 미세 조정하고, RoBERTa를 학습하고, GPT-3를 탐구하고, GPT-2를 학습하고, T5 모델을 구현했으며, 주요 벤치마크 작업과 데이터셋 또한 확인했다.

하지만, RoBERTa 토큰라이저를 학습하고 데이터를 인코딩하면서 토큰라이저의 한계를 확인하거나 모델에 적합한지 평가하지 않았다. 이는 데이터에 기반한다. 2019년의 라펠을 포함해서, 이 책에서 인용한 모든 저자들은 트랜스포머 모델을 위한 데이터셋 준비에 시간을 투자했다.

이번 장에서는, 트랜스포머 다운스트림 작업의 성능을 저해하는 토큰라이저의 한계점을 살펴본다. 학습된 토큰라이저를 그대로 신뢰하면 안된다. 상황에 따라, 일반적인 토큰라이저에서 처리되지 않는 특별한(예를 들어, 고급 의학 용어 같은) 단어 사전이 필요할 수 있다.

먼저 토큰라이저의 종류와 무관하게 성능을 측정할 수 있는 몇 가지 모범 사례(best practice)를 소개할 것이다. 토큰화 관점에서, 데이터셋과 토큰라이저에 대한 기본적인 가이드를 주고자 한다.

다음으로, Word2Vec 토큰라이저로 토큰라이저의 한계를 알아보고, 모든 토큰화 방법에서 나타나는 문제점을 파이썬으로 확인하고 설명할 것이다.

또한, 특정 어휘를 포함하는 데이터셋으로 GPT-2 모델을 실행하여 조건 여부에 따라 샘플을 생성하고 탐구할 것이다.

더 나아가서, 바이트 단위 BPE(byte-level BPE)의 한계를 살펴볼 것이다. GPT-2 토큰라이저의 결과를 보여주는 파이썬 프로그램을 작성하고, 데이터 인코딩 과정에서 발생하는 문제점들을 검토할 것이다. 이를 통해 일반적인 NLP 분석에 GPT-3의 우월함이 꼭 필요하진 않음을 확인할 것이다.

하지만, 이 장의 끝에선, **품사(Part-of-Speech, POS)** 작업으로 GPT-3 엔진을 분석하며, 모델이 얼마나 이해하는지 그리고 준비된 토큰화 사전이 요구사항에 적합한지 확인할 것이다.

이번 장에서는 다음 주제를 다룬다.

- 토큰라이저의 출력을 확인하기 위한 기본지침
- 원시데이터 전략 및 데이터 전처리 전략
- Word2Vec 토큰화 문제와 한계
- Word2Vec 토큰라이저를 평가하기 위한 파이썬 프로그램 작성
- 바이트 단위 BPE 알고리즘의 출력을 평가하기 위한 파이썬 프로그램 작성
- 특정 어휘를 사용하여 NLP 작업 커스터마이징
- GPT-2로 조건 어부에 따라 샘플 실행하기
- GPT-2 토큰라이저 평가하기

먼저 2019년에 라펠이 정의한 텍스트-투-텍스트 방법론을 살펴보자.

트랜스포머 학습을 위해 벤치마크 데이터셋을 다운로드하면 많은 장점이 있다. 데이터가 준비되어 있으며, 모든 연구실이 동일한 자료를 참조한다. 또한, 동일한 데이터로 트랜스포머 모델의 성능을 다른 모델과 비교할 수 있다.

하지만, 트랜스포머 모델의 성능을 향상시키기 위해서는 더 많은 작업이 필요하다. 만약 프로덕션 환경을 위한 트랜스포머 모델을 만든다면 더욱 신중한 계획과 명확한 모범 사례가 필요하다.

이번 섹션에서는 치명적인 문제점을 피하기 위한 모범 사례들을 정의할 것이다.

그리고 나서, 코사인 유사도를 사용하여 토큰화와 데이터 인코딩의 한계를 확인하는 파이썬 예제를 살펴보겠다.

먼저 모범 사례를 확인하자.

### 9.1.1 모범 사례

라펠은 텍스트-투-텍스트 T5 트랜스포머 모델을 정의했다. 그리고 더 나아가서, 원시 데이터를 전처리하지 않고 사용하는 통념을 깨기 시작했다.

데이터 전처리는 학습시간을 줄여준다. 예를 들어, 커먼 크롤(Common Crawl)은 웹에서 추출한 라벨이 없는 텍스트로 이루어져 있으며, 마크업과 텍스트가 아닌 데이터는 삭제했다.

그런데, 구글 T5 팀은 커먼 크롤의 텍스트 중 상당수가 자연어 또는 영어 수준에 미치지 못한다는 사실을 발견했고, 데이터셋을 사용하기 전에 정리가 필요하다고 결정했다.

라펠의 의견대로, 전처리와 품질 관리 단계에 기업 수준의 모범사례를 적용할 것이다. 앞으로 소개할 예시는, 실제 프로젝트에 적용 가능한 수준의 데이터셋을 얻기 위해 필요한 방대한 작업을 보여준다.

그림 9.1은 데이터셋에 적용할 주요 품질 관리 목록이다.

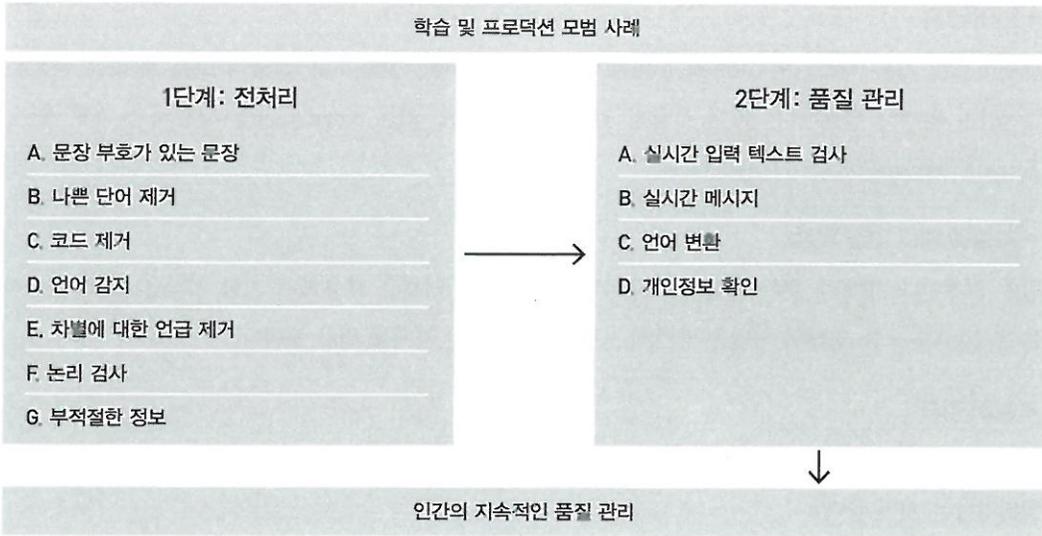


그림 9.1 트랜스포머 데이터셋 모범 사례

그림 9.1에서 볼 수 있듯, 품질 관리 목록은 트랜스포머를 학습할 때의 전처리 단계(1단계)와 프로덕션 환경에서의 품질 관리 단계(2단계)로 나눌 수 있다.

먼저 전처리 단계의 주요 측면을 살펴보겠다.

### 9.1.1.1 1단계: 전처리

라펠은 모델 학습 전 전처리를 권장했다. 여기에 필자는 몇 가지 아이디어를 덧붙였다.

트랜스포머는 언어를 배우는 입장이며, 우리는 트랜스포머의 선생님이다. 하지만 기계 학생에게 언어를 가르치기 위해서는, 적절한 영어가 무엇인지 설명할 수 있어야 한다.

따라서, 데이터를 사용하기 전에 표준 휴리스틱을 일부 적용해야 한다.

- 문장 부호가 있는 문장

마침표나 물음표 같은 문장 부호로 끝나는 문장을 사용하는 것이 좋다.

- 나쁜 단어 제거

나쁜 단어는 제거해야 한다. 다음 사이트에서 예시 목록을 확인할 수 있다.(<https://github.com/LDN/OOBW/List-of-Dirty-Naughty-Obscene-and-Otherwise-Bad-Words>)

- 코드 제거

가끔은 코드가 필요한 경우도 있어서 까다롭다. 하지만 NLP 작업에선 일반적으로 코드를 제거하는 것이 좋다.

- 언어 감지

웹사이트엔 기본 텍스트인 “lorem ipsum”이 종종 포함되어 있다. 데이터셋의 모든 내용이 원하는 언어인지 확인이 필요하다. 50개 이상의 언어를 감지할 수 있는 langdetect를 사용하는 것도 좋다. (<https://pypi.org/project/langdetect/>)

- 차별에 대한 언급 제거

필수 과정이다. 웹에서 모든 것을 스크랩하거나 특정 데이터셋을 사용해서 기반 지식을 구축하는 것을 추천한다. 모든 형태의 차별을 막아라. 누구나 윤리적인 기계를 원할 것이다!

- 논리 검사

**자연어 추론(Natural Language Inference, NLI)** 데이터셋을 학습한 모델로 말이 되지 않는 문장을 필터링하는 것이 좋다.

- 부적절한 정보

동작하지 않는 링크, 비윤리적 웹사이트 또는 개인을 나타내는 텍스트를 제거해야 한다. 어려운 작업이지만, 확실한 가치가 있다.

### 9.1.1.2 2단계: 품질 관리

학습된 모델은 언어를 배운 사람처럼 행동한다. 입력 데이터로부터 무엇을 이해하고 학습할 수 있는지 파악한다. 1단계: 전처리 과정을 거친 입력 데이터는 학습 데이터에 정보를 더할 수 있어야 하고, 학습 데이터셋은 기업 프로젝트의 기반 지식이 된다. 사용자는 데이터셋에서 NLP 작업을 수행할 수 있고, 질문에 대한 신뢰할 수 있는 답변 또는 특정 문서에 대한 유용한 요약 을 얻을 수 있다.

1단계: 전처리에서 설명한 모범 사례를 실시간 데이터에도 적용할 수 있어야 한다. 트랜스포머에 문서 요약 같은 NLP 작업이나 사용자의 요청이 들어온다면 바로 수행할 수 있어야 하기 때문이다.

트랜스포머는 역사상 가장 강력한 NLP 모델이며, 우리는 높은 윤리적 책임감을 가져야 한다.

다음은 몇 가지 모범 사례이다.

- 실시간 입력 텍스트 검사

나쁜 정보를 수용하면 안 된다. 입력을 실시간으로 파싱하며 받아들일 수 없는 데이터를 필터링해야 한다. (1단계 참고)

- 실시간 메시지

필터링된 데이터와 그 이유를 사용자가 로그로 확인할 수 있도록 해야 한다. 트랜스포머에 부적절한 요청이 오면 실시간으로 메시지를 표시한다.

- 언어 변환

가능하다면 희귀한 어휘를 일반적인 어휘로 변경한다. 이 장의 Word2Vec 토큰화 챕터의 사례 4를 참고하라. 항상 가능한 것은 아니지만, 가능한 경우, 한 단계 앞서 나갈 수 있다.

- 개인정보 확인

트랜스포머 모델에 데이터를 스트리밍하거나 사용자 입력을 분석할 때, 사용자나 국가의 승인이 없는 한, 개인정보를 데이터셋과 작업에서 제외해야 한다. 어려운 주제이고, 필요하다면 법률 자문을 받아야 한다.

여러 모범 사례를 확인했다. 이제 인간의 품질 관리가 왜 필수인지 알아보겠다.

### 9.1.1.3 인간의 지속적인 품질 관리

트랜스포머는 복잡한 NLP 작업 대부분을 점진적으로 대체할 것이다. 하지만 인간의 개입은 여전히 필수적이다. 우리는 소셜 미디어 대기업이 모든 것을 자동화했다고 생각하지만, 플랫폼엔 무엇이 좋고 나쁜지 결정하는 콘텐츠 관리자가 있다.

트랜스포머를 학습시키고, 구현하고, 출력을 확인하고, 유의미한 결과를 다시 학습 데이터에 포함시키면, 데이터는 지속적으로 개선되고 트랜스포머를 계속 발전시킬 수 있다.

그림 9.2는 지속적인 품질 관리가 어떻게 트랜스포머의 학습 데이터를 증대시키고 프로덕션 성능을 향상시키는지 보여준다.

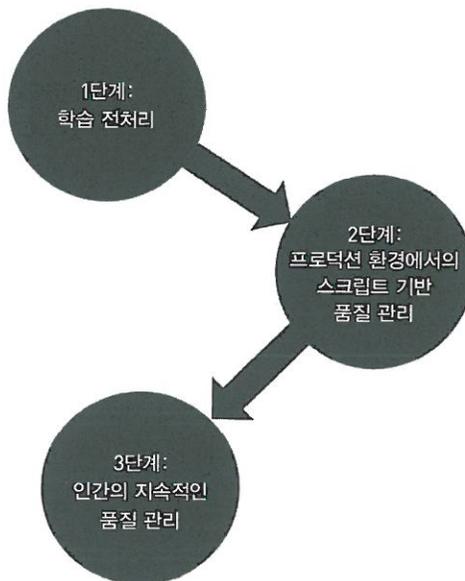


그림 9.2 인간의 지속적인 품질 관리

라펠이 설명한 몇 가지 모범사례와, 기업의 AI 프로젝트 매니저 경험으로 얻은 필자의 아이디어를 덧붙인 가이드라인을 살펴봤다.

이제 파이썬 프로그램을 통해 토큰라이저의 한계를 보여주는 예제를 살펴보겠다.

### 9.1.2 Word2Vec 토큰화

모든 것이 순조로우면, 아무도 학습된 토큰라이저를 의심하지 않는다. 실제 삶에서도 마찬가지다. 엔진에 대해 생각하지 않고도 몇 년간 차를 운전할 수 있다. 그러다 어느 날 차가 갑자기 고장났을 때 왜 그랬는지 그제야 이유를 찾으려 한다.

학습된 토큰라이저에서도 마찬가지이다. 때로는 예상하지 못한 결과가 나타난다. 예를 들어, 그림 9.3 처럼 일부 단어 쌍이 잘못 계산될 수 있다.

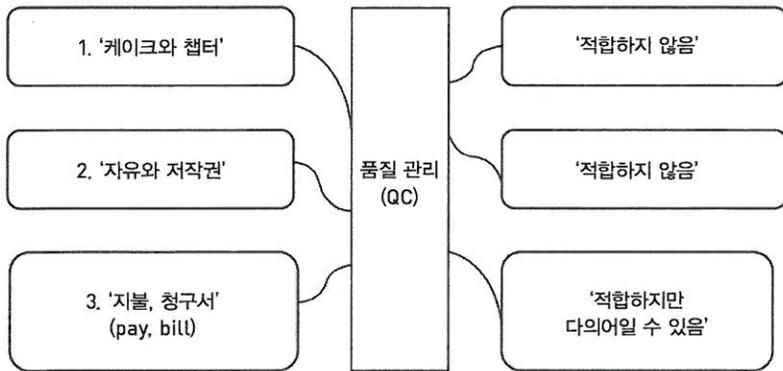


그림 9.3 토큰라이저가 잘못 계산한 단어 쌍

그림 9.3은 미국 독립선언(American Declaration of Independence)과 권리장전(Bill of Rights), 마그나 카르타(English Magna Carta)에서 발췌한 예시이다.

- 케이크와 철퍼는 어울리지 않지만, 토큰라이저에 의해 두 단어의 코사인 유사도가 높다고 계산됐다.
- freedom은 언론의 자유를 의미한다. "저작권"은 무료(free) 전자책 편집자가 기입한 문구다.
- 지불(pay)과 청구서(bill)는 일상 영어에서 함께 사용된다. 다의성(polysemy)은 한 단어가 여러 의미를 가지는 경우를 의미하는데, 예를 들어, Bill은 지불할 금액을 뜻하기도 하지만 권리장전(Bill of Rights)을 가리키기도 한다. 수용할 만한 결과이지만, 그저 운이었을 수 있다.

계속하기 전에, 몇 가지를 짚고 넘어가겠다. QC는 품질 관리(quality control)를 의미한다. 어떤 전략적 기업 프로젝트에도 QC는 필수이다. 출력 결과의 품질이 프로젝트의 생존을 결정한다. 전략적으로 중요하지 않은 프로젝트라면, 일부 오류를 허용할 수 있다. 하지만 전략적 프로젝트라면, 몇 가지 오류에도 위험성을 평가하고 프로젝트를 계속할지 또는 중단할지 판단할 것이다.

품질 관리와 리스크 관리의 관점에서 볼 때, 관련성이 없는 데이터셋(쓸모없는 단어가 너무 많거나 중요한 단어가 누락된 데이터셋)를 토큰화하면 임베딩 알고리즘에 혼란을 주고 “ 좋지 않은 결과”를 초래할 수 있다. 토큰화가 임베딩에 영향을 주므로, 이 챕터에서는 “토큰화”라는 단어를 임베딩을 포함하는 넓은 의미로 사용한다.

전략적으로 중요한 AI 프로젝트라면(특히 의료, 비행기 또는 로켓 조립 등 중요 분야), 단 하나의 오류도 극적인 “나쁜 결과”를 초래한다.

Open Tokenizer.ipynb를 열어보자. 2장, 트랜스포머 모델 아키텍처 살펴보기에서 사용한 positional\_encoding.ipynb의 내용과 이어진다.

Word2Vec 알고리즘의 확률적 특성으로 인해 결과는 실행할 때마다 다를 수 있다.

먼저 필수 구성 요소를 설치하고 임포트한다.

```
#@title 사전 요구 사항
!pip install gensim==3.8.3
import nltk
nltk.download('punkt')
import math
import numpy as np
from nltk.tokenize import sent_tokenize, word_tokenize
import gensim
from gensim.models import Word2Vec
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings(action = 'ignore')
```

text.txt 데이터셋에는, 미국 독립선언, 권리장전, 마그나 카르타, 임마누엘 칸트(Immanuel Kant)의 작품 등을 비롯한 텍스트가 들어있다.

이제 text.txt를 토큰화하고 word2vec 모델을 학습한다.

```
#@title Word2Vec 토큰화
#'text.txt' file
sample = open("text.txt", "r") s = sample.read()
# 이스케이프 문자 처리하기
f = s.replace("\n", " ") data = []
# 문장 파싱하기
for i in sent_tokenize(f):
    temp = []
    # 문장을 단어로 토큰화하기
    for j in word_tokenize(i):
        temp.append(j.lower())
    data.append(temp)
# 스킵 그램 모델을 생성하기
model2 = gensim.models.Word2Vec(data, min_count = 1, size = 512, window = 5, sg = 1)
print(model2)
```

window=5는 흥미로운 파라미터로, 입력 문장에서 현재 단어와 예측 단어 사이의 거리를 제한하는 역할이다. sg=1로 설정하면 학습 시 스킵 그램(skip-gram) 알고리즘을 사용한다.

출력 결과는 모델 어휘의 크기가 10,816 임베딩 차원이 512이고, 학습률(learning rate)은 alpha=0.025였음을 보여준다.

```
Word2Vec(vocab=10816, size=512, alpha=0.025)
```

단어를 임베딩으로 표현하는 모델을 확보했고, 이제 코사인 유사도를 계산하는 함수 similarity(word1, word2)를 정의할 수 있다. 함수에 word1과 word2를 전달하면, 두 단어 사이의 코사인 유사도를 계산할 것이다. 값이 클수록, 유사도가 높음을 의미한다.

함수는 먼저 단어가 모델에 있는지([unk] 여부)를 감지하고 메시지를 표시한다.

```

#@title 코사인 유사도
def similarity(word1,word2):
    cosine=False #초기값
    try:
        a=model2[word1]
        cosine=True
    except KeyError: # KeyError 예외 발생
        print(word1, "[unk] key not found in dictionary")#False 암시
    try:
        b=model2[word2]#a=True 암시
    except KeyError: #KeyError 예외 발생
        cosine=False # a, b 모두 참이어야 함
        print(word2, "[unk] key not found in dictionary")

```

word1과 word2가 모두 모델의 단어사전에 존재하는, cosine==True인 경우에만 코사인 유사도를 계산한다.

```

if(cosine==True):
    b=model2[word2]
    # 코사인 유사도 계산하기
    dot = np.dot(a, b)
    norma = np.linalg.norm(a)
    normb = np.linalg.norm(b)
    cos = dot / (norma * normb)
    aa = a.reshape(1,512)
    ba = b.reshape(1,512)
    #print("Word1",aa)
    #print("Word2",ba)
    cos_lib = cosine_similarity(aa, ba)
    #print(cos_lib, "word similarity")

if(cosine==False):cos_lib=0:
return cos_lib

```

함수는 계산한 코사인 유사도, cos\_lib를 반환한다.

이제 여섯 가지 사례를 살펴보겠다. text.txt를 데이터셋이라고 부를 것이다.

먼저 사례 0을 살펴보자.

### 9.1.2.1 사례 0: 단어가 데이터셋과 사전에 있는 경우

freedom과 liberty는 데이터셋에 있는 단어이며, 코사인 유사도는 다음과 같이 구할 수 있다.

```
#@title 사례 0: 단어가 데이터셋과 사전에 있는 경우
word1="freedom";word2="liberty"
print("Similarity",similarity(word1,word2),word1,word2)
```

함수의 한계를 알아보기 위해 다양한 텍스트를 집어넣었기 때문에, 유사도가 0.79로 제한되었다.

```
Similarity [[0.79085565]] freedom liberty
```

유사도 알고리즘의 결과가 모든 상황에서 동일하지는 않다. 이 섹션의 결과는 데이터셋의 내용, 크기, 모듈의 버전 등에 따라 다를 수 있다. 다음 예시에서처럼 셀을 10번 돌려보면, 같은 결과가 나올 수도 있고 아닐 수도 있다.

다음 경우에는, 구글 코랩(Google Colab) VM과 CPU에서 같은 결과를 10번 얻었다.

```
Run 1: Similarity [[0.62018466]] freedom liberty
Run 2: Similarity [[0.62018466]] freedom liberty
...
Run 10: Similarity [[0.62018466]] freedom liberty
```

하지만, 코랩 메뉴에서 “런타임 연결 해제 및 삭제(Disconnect and delete runtime)”를 선택하고, 새로운 VM과 CPU에서는 다음 결과를 얻었다.

```
Run 1: Similarity [[0.51549244]] freedom liberty
Run 2: Similarity [[0.51549244]] freedom liberty
...
Run 10: Similarity [[0.51549244]] freedom liberty
```

“런타임 연결 해제 및 삭제(Disconnect and delete runtime)”를 한번 더 선택하고, 새로운 VM과 CPU와 함께 GPU도 활성화 하자, 다음 결과를 얻었다.

```
Run 1: Similarity [[0.58365834]] freedom liberty
Run 2: Similarity [[0.58365834]] freedom liberty
...
Run 10: Similarity [[0.58365834]] freedom liberty
```

확률론적 알고리즘은 확률에 기반한 결과를 보인다. 필요하다면 n 번씩 예측을 수행해보는 것이 좋다. 이제 단어가 없을 때 일어나는 일을 살펴보자.

### 9.1.2.2 사례 1: 단어가 데이터셋이나 사전에 없는 경우

단어가 없으면 다양한 문제가 발생한다. 이번에는, 유사도 함수에 corporations와 rights를 전달해 보겠다.

```
#@title 사례 1: 단어가 데이터셋이나 사전에 없는 경우
word1="corporations";word2="rights"
print("Similarity",similarity(word1,word2),word1,word2)
```

단어 사전에는 corporations가 없다.

```
corporations [unk] key not found in dictionary
Similarity 0 corporations rights
```

더 이상 진행할 수 없다! 단어는 알 수 없는(unknown) 토큰이다.

빠진 단어가 중요한 단어라면, 사건과 문제를 연쇄해서 유발하고 트랜스포머 모델의 출력을 왜곡할 것이다. 이러한 단어를 unk라고 지칭하겠다.

다음 몇 가지 가능성을 확인하고, 적절히 답해야 한다.

- unk가 데이터셋에 있었지만 토큰화된 사전에 사용되지 않았다.
- unk가 데이터셋에 없었다(corporation은 여기에 해당한다). 이 경우에는, 사전에 없는 이유를 설명할 수 있다.

- 사용자가 트랜스포머에 토큰화되지 않는 토큰을 입력할 때마다, 운영 환경에서 unk가 나타날 것이다.
- unk가 데이터셋에서는 중요하지 않았지만, 트랜스포머 사용 시에는 중요하다.

몇몇 경우에 트랜스포머가 끔찍한 결과를 생성한다면, 이 문제점들은 점점 커질 것이다. 학습 단계에서, 특정 다운스트림 작업에 대한 트랜스포머의 성능이 0.8이라면 훌륭하다고 생각할 수 있다. 하지만 현실에서, 20% 확률로 틀리는 시스템은 아무도 사용하고 싶지 않다.

대신, 소셜미디어처럼 메시지에 언어 구조가 없고 모호한 환경이라면 0.8이라는 성능이 만족스러울 수는 있다.

이제 최악의 시나리오가 만들어진다. 우리가 앞에서 살펴본 문제를 NLP 팀이 발견하고 바이트 단위 BPE를 도입한다고 가정하자. 필요하다면 **4장, RoBERTa 모델 처음부터 사전 학습하기**의 3단계: 토큰나이저 학습하기 부분을 몇 분간 복습하는 것도 좋다.

팀이 문제해결을 위해 오직 바이트 단위 BPE만 사용하면 악몽이 시작된다.

- unk는 단어 조각으로 나누어진다. 예를 들어, corporation은 corp + o + ra + tion + s가 된다. 이러한 토큰들은 데이터셋에서 발견될 확률이 높다.
- unk는 데이터셋에 있는 단어들로 표현되지만, 원래 토큰의 뜻을 표현하지 못한다.
- 트랜스포머는 잘 학습되고, unk가 의미없이 학습된 것을 아무도 알지 못한다.
- 트랜스포머는 0.8에서 0.9 수준의 뛰어난 결과를 보일 것이다.
- 전문 분야의 사용자가 중요한 순간에 잘못된 결과를 적용하기 전까진 모두가 박수를 보낼 것이다.<sup>25</sup> 예를 들어, 영어에서 corp은 기업 또는 상병을 의미하는데, 이로 인해 corp과 다른 단어들 사이에 잘못된 연관관계가 생길 수 있다.

일반적인 소셜 미디어라면 사소한 주제에 트랜스포머를 사용하기 충분할 수 있다. 하지만 실제 기업 프로젝트에서 데이터셋에 맞는 토큰나이저를 생성하려면 많은 노력이 필요하다. 실생활에서 데이터셋은 사용자의 입력에 따라 매일 증가한다. 사용자 입력은 정기적으로 학습하고 업데이트해야 하는 데이터셋의 일부인 것이다.

다음은 품질 관리를 보장하는 한 과정의 예시이다.

- 바이트 단위 BPE 알고리즘으로 토큰나이저를 학습한다.
- 이번 장의 토큰화된 데이터 확인하기 섹션에서 살펴볼 프로그램으로 결과를 확인한다.

25 역주. 오류가 발생하고 일람이 발생하는 대신, 동적에는 이상이 없게 되어, 문제를 발견하기 힘들어졌다.

- Word2Vec 알고리즘으로 학습한 품질 관리 전용 토큰나이저 또한 준비한다. 이 토큰나이저를 사용해 데이터셋을 파싱하고, unk 토큰을 찾아 데이터베이스에 넣는다. 쿼리를 실행하여 중요한 단어가 누락되었는지 확인한다.

이렇게 자세하게 검토하는 프로세스가 불필요해 보일 수 있으며, 누락된 단어로도 추론하는 트랜스포머의 능력에 의존하고 싶을 수도 있다.

하지만 중요한 의사 결정이 필요한 전략적 프로젝트의 경우, 여러 가지 품질 관리 방법을 실행하는 것이 좋다. 만약 법률 요약 프로젝트라면 단어 하나로 법정에서 승패가 갈릴 수 있다. 항공(비행기, 로켓) 프로젝트라면 단 하나의 오류라도 허용되지 않는다.

더 많은 품질 관리 프로세스를 수행할수록, 트랜스포머 솔루션을 더욱 신뢰할 수 있다.

신뢰할 수 있는 데이터셋을 얻기 위해서는 많은 노력이 필요하다! 모든 트랜스포머 논문은 수용할 만한 데이터셋을 얻기 위한 과정을 다루고 있다.

노이즈가 많은 관계도 문제를 일으킨다.

### 9.1.2.3 사례 2: 노이즈가 많은 관계

이번 사례에서는 데이터셋에 있는 두 단어, etext와 declaration을 살펴보겠다.

```
#@title 사례 2: 노이즈가 많은 관계
word1="etext";word2="declaration"
print("Similarity",similarity(word1,word2),word1,word2)
```

두 단어 모두 토큰나이저 사전에도 포함되어 있다.

```
Similarity [[0.880751]] etext declaration
```

0.5를 초과하는 것을 보면, 코사인 유사도는 연관관계에 확신이 있다. 알고리즘의 확률적 특성 때문에 실행마다 결과가 다를 수 있다.

사소한 수준이나 소셜 미디어 수준에서는 모든 것이 좋아 보인다.

하지만 전문적인 수준에서, 재앙적인 결과다!

etext는 프로젝트 구텐베르크(Project Gutenberg) 사이트에 있는 각 전자책 서문에 포함된 단어이다. 트랜스포머의 목표를 생각해 보자.

- 편집자의 서문을 파악하는 것인가?
- 아니면 책의 내용을 파악하는 것인가?

트랜스포머의 용도에 따라 다르며, 결정에 며칠이 걸릴지도 모른다. 예를 들어, 편집자는 서문을 이해하는 트랜스포머를 사용해서 서문을 생성하고 싶을 수 있다. 그렇다면 콘텐츠를 제거해야 할까?

declaration은 권리장전의 내용과 관련된 의미있는 단어이다.

etext는 프로젝트 구텐베르크가 모든 전자책에 넣은 서문의 일부이다.

트랜스포머가 텍스트를 생성할 때 etext 대신 declaration을 사용하는 등 잘못된 자연어 추론을 할 수 있다.

이제 누락된 단어 문제를 살펴보겠다.

### 9.1.2.4 사례 3: 텍스트의 단어가 사전에 없는 경우

경우에 따라 텍스트에 있는 단어가 사전에 없을 수 있다. 이 경우, 결과가 왜곡된다.

두 단어, pie와 logic을 살펴보자.

```
#@title 사례 3: 텍스트의 단어가 사전에 없는 경우
word1="pie";word2="logic"
print("Similarity",similarity(word1,word2),word1,word2)
```

pie는 사전에 없다.

```
pie [unk] key not found in dictionary
Similarity 0 pie logic
```

pie가 토큰화 사전에 있을 것이라고 가정했지만, 그렇지 않은 경우도 대비해야 한다. pie는 데이터셋에 없는 단어이다.

따라서, 파이프라인에는 사전에 없는 단어를 감지하고 수정하거나 대체하는 기능이 필요하다. 또한, 파이프라인에는 데이터셋의 중요 단어를 감지하는 기능도 포함하는 것이 좋다.

이제 희귀한 단어로 인해 발생하는 문제를 살펴보자.

#### 9.1.2.5 사례 4: 희귀한 단어

단순한 어플리케이션 이상의 특정 트랜스포머 작업이라면 희귀한 단어가 치명적인 영향을 끼칠 수 있다.

희귀한 단어 관리는 자연어 처리의 다양한 분야와 관련이 있다.

- 희귀한 단어는 데이터셋에 존재해도 영향이 없거나 모델 학습을 방해한다.
- 희귀한 단어는 의학, 법률, 공학 용어 또는 기타 전문 용어일 수 있다.
- 희귀한 단어는 속어일 수 있다.
- 영어에는 수백 가지 변형이 있다. 예를 들어 미국, 영국, 싱가포르, 인도, 호주 등 여러 국가에서 서로 다른 영어 단어를 사용한다.
- 희귀한 단어는 수세기 전에 쓰이고 잊힌 단어나 전문가만 사용하는 용어일 수 있다.

예를 들어, `justiciar`라는 단어를 확인해 보자.

##### #@title 사례 4: 희귀한 단어

```
word1="justiciar";word2="judgement"  
print("Similarity",similarity(word1,word2),word1,word2)
```

`judgement`와의 유사도는 그럴듯하지만 더 높아야 한다.

```
Similarity [[0.6606605]] justiciar judgement
```

`justiciar`라는 단어가 억지스럽다고 생각할 수 있다. 토큰나이저는 이 단어를 13세기 초에 쓰여진 마그나 카르타에서 가져왔다. 안타깝게도, 프로그램은 혼란스러워지고 실행할 때마다 예상치 못한 결과가 나올 것이다.

#### NOTE

실행마다 예측 결과가 다를 수 있다. 하지만 이 결과로 트랜스포머 모델을 사용하는 프로젝트에서 토큰화와 임베딩을 주의해야 한다는 것을 알 수 있다.

그러나, 마그나 카르타의 여러 조항은 21세기 영국에서도 여전히 유효하다! 일례로 조항 1, 13, 39, 40은 여전히 유효하다!

다음은 데이터셋에도 있는 내용으로, 마그나 카르타의 가장 유명한 부분이다.

(39) 모든 자유인은 공정한 법의 판결에 의해서 또는 땅의 법에 의해서를 제외하고 압류나 투옥, 또는 권리나 소유의 박탈, 또는 추방이나 망명, 또는 어떤 다른 방법으로 그의 지위의 박탈을 받지 않을 것이며 우리가 그들에게 강압적으로 하거나, 또는 다른 사람들에게 그렇게 하도록 보내는 것을 할 수 없을 것이다.

(40) 권리나 정의는 팔 수도 없고 취소할 수도 없고 지연될 수도 없다.

법률회사에서 문서 요약 등을 위한 트랜스포머 모델을 구현한다면, 주의해야 한다.

이제 희귀한 단어 문제를 해결하는 방법을 알아보겠다.

#### 9.1.2.6 사례 5: 희귀한 단어 대체하기

희귀한 단어를 대체하는 것 자체가 하나의 프로젝트다. 이러한 작업은 특정 과제나 프로젝트에 적합하다. 어떤 기업이 예산을 투자해 항공 분야의 지식 베이스를 구축한다고 가정하겠다. 이 경우에는 토큰라이저의 사전에 빠진 단어를 찾기 위해 시간을 할애할 가치가 있다.

문제점을 주제별로 그룹화하여 해결하고 지식 베이스를 정기적으로 업데이트 할 것이다.

사례 4는 우연히 *justiciar*라는 단어를 발견했다. 단어의 기원을 찾아보면, 프랑스 노르만어에서 유래했으며 프랑스 라틴어와 유사한 *judicaire*의 어원이다.

다음과 같이 *justiciar*를 동일한 개념을 전달하는 *judge*로 대체 할 수 있다.

```
#@title 사례 5: 희귀한 단어 대체하기
word1="judge"word2="judgement"
print("Similarity",similarity(word1,word2),word1,word2)
```

흥미로운 결과를 얻을 수 있지만 알고리즘의 결정론적이지 않은 측면 때문에 여전히 주의해야 한다.

```
Similarity [[0.7962761]] judge judgement
```

*justiciar*이라는 단어를 그대로 사용하되, 이 단어의 현대적 의미인 *judge*와 비교해 볼 수도 있다. 희귀한 단어 대체하기를 다음과 같이 구현해 볼 수 있다.

```
word1="justiciar";word2="judge"
print("Similarity",similarity(word1,word2),word1,word2)
```

이렇게 몇몇 희귀 단어는 더 자주 사용되는 단어로 대체해야 한다.

결과는 다음과 같이 만족스럽다.

```
Similarity [[0.9659128]] judge judgement
```

상관관계가 0.9 이상인 대체단어를 찾을 때까지 쿼리를 실행할 수도 있다. 만약 중요한 법률 프로젝트라면, 희귀한 단어가 포함된 문서를 표준 영어로 번역하는 것도 좋다. 이렇게 하면 트랜스포머의 NLP 작업 성능이 올라가고, 회사의 지식 베이스가 점진적으로 발전할 것이다.

다음으로, 코사인 유사도로 함의(entailment)를 검증하는 방법을 살펴보겠다.

#### 9.1.2.7 사례 6: 함의

이번에는, 사전에 있는 단어를 정해진 순서대로 테스트한다. 유사도 함수로, “pay”와 “debt”가 관련 있는지 살펴보자.

```
#@title 사례 6: 함의
word1="pay";word2="debt"
print("Similarity",similarity(word1,word2),word1,word2)
```

결과는 다음과 같다.

```
Similarity [[0.89891946]] pay debt
```

이렇게 데이터셋의 여러 단어 쌍을 가져와서 관련이 있는지 확인할 수 있다. 예를 들어, pay와 debt는 법률 부서의 이메일에서 추출할 수 있는 단어 쌍이다. 코사인 유사도가 0.9 이상이면, 이메일에서 쓸모 없는 부분을 제거하고, 본문만 회사의 데이터셋에 추가할 수 있다.

이제 잘 학습된 토큰라이저가 NLP 작업에 얼마나 적합한지 살펴보자.

## 9.2

## 특정 어휘가 포함된 표준 NLP 작업

이번 챕터는, Word2Vec 토큰화 섹션의 사례 4: 희귀한 단어와 사례 5: 희귀한 단어 대체하기에 초점을 맞춘다.

Training\_OpenAI\_GPT\_2\_CH09.ipynb를 사용할 것이다. 이 노트북 파일은 7장, GPT-3 엔진을 사용한 초인간 트랜스포머 등장에서 사용한 파일에서 다음 내역을 변경한 버전이다.

- dset 대신, 의학 내용이 포함된 mdset을 데이터셋으로 사용한다.
- 바이트 단위 BPE로 토큰화된 텍스트를 다루기 위해 파이썬 함수를 추가했다.

Training\_OpenAI\_GPT\_2\_CH09.ipynb의 내용 중에 7장, GPT-3 엔진을 사용한 초인간 트랜스포머 등장과 부록 III, 부록 IV에서 다룬 내용은 설명하지 않겠다. 7장처럼 필요한 파일을 미리 업로드해야 한다.

모델을 학습하는 시간에는 제한이 없다. 모델을 저장하려면 중단해야 한다.

깃허브의 Chapte09/gpt-2-train\_files 경로에 필요한 파일이 있다. 7장과 노트북 내용이 같지만, 데이터셋은 다르다. 해당 경로의 mdset을 사용한다.

먼저, 의학 콘텐츠를 학습한 GPT-2로 조건 없이 샘플을 생성해 보겠다.

### [설명 추가]

Training\_OpenAI\_GPT\_2\_CH09.ipynb은 텐서플로우 1.x 버전에 작동한다. 하지만 2024년 현재, 구글 코랩은 텐서플로우 1.x 버전을 지원하지 않는다. 두 가지 해결책이 있다.

- (1) Training\_OpenAI\_GPT\_2\_CH09.ipynb을 텐서플로우 1.x가 설치된 로컬 머신에 받아 실행하기
- (2) 구글 코랩에서 깃허브 내에 Chapter09 디렉터리에 존재하는 Summarizing\_with\_ChatGPT.ipynb을 실행하기

Summarizing\_with\_ChatGPT.ipynb은 OpenAI API에서 제공하는 GPT-3.5(ChatGPT)를 사용해 요약 작업하는 방법을 소개한다. 이 때, 희귀 단어를 포함하는 경우를 실험해 트랜스포머의 한계를 확인한다. 이 노트북 파일은 다음 단계를 따른다.

1. openai를 설치하고 API Key 설정하기
2. 요약하기
3. 토큰화하기
4. 한계 파악하기
5. 결론

7장의 이론을 이해한 다음, 이 노트북의 4. 한계 파악하기에서 더 많은 한계를 실험하고 어떻게 해결할 수 있을지 생각해보자.

## 9.2.1 GPT-2로 조건 없이 샘플 생성하기

트랜스포머 내부의 동작을 이해하기 위해 직접 실행해 보자. 물론, 모든 과정을 뛰어넘고 OpenAI API를 사용할 수도 있다. 하지만, 4차 산업시대의 AI 전문가가 트랜스포머가 파이프라인에서 어떻게 동작하는지 모호하지 않고 명확하게 말할 수 있어야 한다.

사례 4: 희귀한 단어와 사례 5: 희귀한 단어 대체하기에서 전문용어, 오래된 영어, 전 세계의 다양한 영어, 속어 등에서 희귀한 단어를 볼 수 있었다.

2020년의 뉴스는 코로나19와 관련된 의학 용어로 가득 차있다. 이번 섹션에서는 GPT-2 트랜스포머가 의학 용어에 어떻게 대응하나 살펴보려 한다.

마르티나 콘테(Martina Conte)와 나디아 로이(Nadia Loy)가 2020년에 발표한 논문, 「Multi-cue kinetic model with non-local sensing for cell migration on a fibers network with chemotaxis」를 데이터셋으로 사용해 인코딩하고 학습할 것이다.

논문 제목에는 이해하기 어렵고 희귀한 단어가 사용되었다.

gpt-2-train\_files 경로의 mdset.txt 파일을 불러온 후 7장, GPT-3 엔진을 사용한 초인간 트랜스포머 등장을 참고하여 실행한다. 7장의 가이드에 따라 셀 단위로 실행할 수 있다. 이번 섹션은 오리지널 GPT-2를 API가 아니라 로우-레벨(low-level)로 직접 실행한다.

의학 데이터셋으로 모델을 학습하고 조건 없이 샘플 생성하기(Step 11: Generating Unconditional Samples) 셀을 실행한다.

```
#@title 단계 11: 조건 없이 샘플 생성하기
import os # 런타임이 재시작된 이후에 임포트한다.
os.chdir("/content/gpt-2/src")
!python generate_unconditional_samples.py --model_name '117M'
```

#### NOTE

이번 노트북에 있는 명령어와 코드의 실행시간은 장비 성능에 따라 달라진다. 이 책의 GPT-2 코드는 교육을 목적으로 하고 있다. 실제 운영에선 OpenAI의 GPT-3 API를 사용할 것을 권장한다. 해당 방법의 응답이 더 빠르기 때문이다.

셀을 실행하고 원하는 시점에 중단하면 다음과 같이 랜덤 출력이 생성된다.

커뮤니티 기반 기계는 생물막 성장을 촉진한다. 커뮤니티 구성원들은 생화학의 세포가 환경 및 자신과 상호 작용하는 방식을 발견하는 주요 도구로 삼고, 효과적인 모방을 위해 모든 구성 요소를 식별하고 이해한다.

#### 2. OL 인식

상변화(IP) 폴리머라제("tcrecs"라고도 함)의 세포 이중 절단은 진동 연산자로 plainNA를 절단하는 동안 프로세스의 개시 및 유지를 허용하는 기본 구성을 변경하는 이중교차 효소의 특징적인 패턴을 나타낸다. translational parasubstitution(TMT) 중에 발생한 급진적인 수정 직후에 SYX는 어느 정도 통제되지 않은 활성화를 달성했다. TRSI 돌연변이는 모든 세포에 걸쳐 세포 이중 트라이어드(DTT) 신호 전달에 통합된 가장 중요한 TCbase sps의 자가인산화를 도입했으며, 이를 통해 우리는 R h 및 물론 IC 2A를 허용한다.->

.../...

출력 내용을 자세히 보면, 다음 요인들을 확인할 수 있다.

- 생성된 문장구조가 비교적 그럴듯하다.
- 출력 내용의 문법도 나쁘지 않다.
- 전문가가 아닌 사람에게서는 인간의 결과물처럼 보일 수 있다.

그런데, 콘텐츠의 내용은 말이 안된다. 트랜스포머는 우리가 학습시킨 의학 논문과 관련된 실제 내용을 만들 수 없었다. 더 나은 결과를 얻으려면 많은 노력이 필요하다. 물론 데이터셋을 키울수도 있다. 하지만 그렇게해도 우리가 원하는 정보가 담긴다고 확신할 수 없다. 예를 들어, 다음 문장을 포함한 데이터셋으로, 코로나19와 관련된 의학 프로젝트를 한다고 생각해보자.

- 코로나19는 위험한 바이러스가 아니라, 일반 감기와 같다.
- 코로나19는 매우 위험한 바이러스다.
- 코로나19는 바이러스가 아니라 연구실에서 만든 것이다.
- 코로나19는 연구실에서 만들었으리가 없다!
- 백신은 위험하다!
- 백신은 생명을 구한다.
- 정부는 팬데믹을 제대로 관리하지 못했다.
- 정부는 필요한 조치를 했다.

여러 상반되는 문장이 있을 수 있다. 이런 상반된 내용들 때문에 전문적인 의료 프로젝트, 항공, 운송 등 주요 분야에서 데이터셋과 토큰라이저를 도메인에 맞게 최적화해야 한다.

수십억 단어로 구성된 데이터셋이 있다고 가정해도, 모순되고 노이즈가 많아 신뢰할 수 있는 결과를 얻을 수 없다. 과학 논문과 관련된 소량의 데이터로 제한 한다고 해도, 과학자들도 모든 내용에 서로 동의하지는 않는다.

결국 신뢰할 수 있는 결과를 도출하려면 많은 노력과 탄탄한 팀이 필요하다.

이제 GPT-2 모델에 조건을 넣어보자.

## 9.2.2 조건이 있는 샘플 생성하기

이번 섹션에선, 대화형 컨텍스트 완성 예제(Step 12: Interactive Context and Completion Examples) 셸을 실행한다.

```
#@title 단계 12: 대화형 컨텍스트 완성 예제
import os # 런타임이 재시작된 이후에 임포트한다.
os.chdir("/content/gpt-2/src")
!python interactive_conditional_samples.py --temperature 0.8 --top_k 40
--model_name '117M' --length 50
```

4차 산업시대 AI 전문가는 코드에 덜 집중하는 대신 트랜스포머 모델에게 무엇을 해야 하는지 보여주 기 위해 노력해야 한다. 모든 모델엔 무엇을 해야 하는지 보여주는 단계가 필요하고, 조건없이 애매하 게 데이터를 생성하는 것은 충분하지 않다.

다음과 같이, GPT-2 모델에게 의학 논문 일부를 조건으로 줄 수 있다.

이러한 과정에서 세포는 환경을 감지하고 특정 표적(택시, taxis)을 향해 특정 방향으로 이동하도록 유도 하는 외부 요인에 반응하여 특정 선호 방향으로 지속적으로 이동한다. 이동을 유도하는 안내 신호는 생화학 적 또는 생물물리학적일 수 있다. 예를 들어 생화학적 단서는 단방향 자극을 수반하는 화학적 이동을 유발 하는 용해성 인자나 성장 인자 등이 될 수 있다. 단방향 자극. 단방향 자극을 생성하는 다른 단서에는 예를 들어, 햅토택시를 유도하는 기질에 결합된 리간드, ECM의 강성이 증가하는 영역으로의 이동을 포함하는 듀 로택시, 전기장 또는 전류에 의해 유도되는 방향성 운동을 규정하는 갈바노택시라고도 하는 전기택시, 빛의 자극에 의해 방향이 지정된 운동을 의미하는 광택시 등이 있다 [34]. 중요한 생물물리학적 단서는 세포외기 질(ECM)의 특성 중 일부이며, 그중에서도 콜라겐 섬유의 정렬과 강성이 가장 먼저 고려된다. 특히, 섬유 정렬 은 접촉 유도를 자극하는 것으로 나타났다 [22, 21]. TL;DR:

입력 텍스트 마지막에 “TL;DR:”을 추가해서, 조건으로 준 텍스트를 GPT-2가 요약하도록 했다. 출력 결과는 문법적으로, 의미적으로 모두 합리적이다.

단일 조직의 ECM이 가장 효과적이다.  
이 문제를 해결하기 위해 우리는 활성화 시 단백질이 외인성 표적으로 전환되도록 유도하는 새로운 이미징 및 면역 염색 방식을 개발했다.

출력은 결정론적이지 않으므로, 다음과 같은 결과도 나올 수 있다.

레이저(즉, 한 번에 한 번만 펄스를 쏘는 레이저)가 아니라 표적을 겨냥하여 주어진 방향으로 레이저를 쏘는 방식이다. 간단히 말해, 조심하도록 하자.

결과는 나아졌지만 더 많은 연구가 필요하다. 이 장과 앞선 예시로 다음 결과를 도출할 수 있다.

방대한 양의 무작위 웹 크롤링 데이터로 트랜스포머를 사전 학습하면 트랜스포머에게 한글을 가르칠 수 있다. 하지만, 마치 우리처럼, 특정 분야의 전문가가 되려면 트랜스포머도 해당 분야를 학습해야 한다.

이제 더 나아가서 토큰화된 데이터를 다뤄보겠다.

### 9.2.3 토큰화된 데이터 확인하기

이 섹션에서는 사전 학습된 토큰라이저로 인코딩한 첫 단어를 읽어볼 것이다.

다음 셀을 실행하기 전에, 기존에 실행 중인 셀을 중지해야 한다.

이번 챗터의 노트북 파일인 Training\_OpenAI\_GPT2\_CH09.ipynb에서 “토큰화된 데이터 확인하기 (Additional Tools: Controlling Tokenized Data)” 셀로 이동하자. 이 셀은 이번 챗터를 위해 추가했다.

먼저, mdset의 의료 논문을 인코딩한 out.npz의 압축을 푼다.

```
#@title 추가 도구: 토큰화된 데이터 확인하기
#out.npz 압축 풀기
import zipfile
with zipfile.ZipFile('/content/gpt-2/src/out.npz', 'r') as zip_ref:
    zip_ref.extractall('/content/gpt-2/src/')
```

압축을 풀면 arr\_0.npy를 읽어올 수 있다. arr\_0.npy는 우리가 사용하려는 인코딩된 데이터셋이 있는 넘파이 배열이다.

```
#인코딩된 데이터셋에 있는 arr_0.npy 로드하기
import numpy as np
f=np.load('/content/gpt-2/src/arr_0.npy')
print(f)
print(f.shape)
for i in range(0,10):
    print(f[i])
```

배열의 앞쪽 요소들이 몇 개 출력된다.

```
[1212 5644 326 ... 13 198 2682]
```

이제, encoder.json을 열고 파이썬 딕셔너리(dictionary)로 변환한다.

```

#encoder.json 로드하기
import json
i=0
with open("/content/gpt-2/models/117M/encoder.json", "r") as read_file:
    print("Converting the JSON encoded data into a Python dictionary")
    developer = json.load(read_file) #인코딩된 데이터를 파이썬 딕셔너리(dictionary)로 변환하기
    for key, value in developer.items(): #변환한 데이터 확인하기
        i+=1
        if(i>10):
            break;
        print(key, ":", value)

```

마지막으로, 인코딩된 데이터셋에서 앞쪽 500개 토큰의 키(key)와 값(value)을 출력한다.

```

#인코딩된 토큰의 키(key)와 값(value) 검색하기
for i in range(0,500):
    for key, value in developer.items():
        if f[i]==value:
            print(key, ":", value)

```

mdset.txt의 앞 단어는 다음과 같다.

This suggests that

GPT-2의 사전 학습된 토큰라이저가 쉽게 식별하도록 하기 위해 이 단어들을 추가로 넣어두었다.

This : 1212  
 Ġsuggests : 5644  
 Ġthat : 326

(Ġ)는 공백 문자로, 공백 다음에 오는 토큰을 쉽게 구별할 수 있다. 이제 의학 논문의 다음 단어도 확인해 보자.

amoeboid

amoeboid는 희귀한 단어다. GPT-2 토큰라이저는 amoeboid를 하위 단어로 분리했다.

```
Ġam : 716
o : 78
eb : 1765
oid : 1868
```

공백 문자를 제외하고 확인해 보자. amoeboid를 am+o+eb+oid로 분리했다. 바이트 단위 BPE 전략을 사용한 덕분에, 언논(unknown) 토큰은 나오지 않았다.

하지만, 트랜스포머의 어텐션 층은 다음과 같은 시퀀스에 영향을 받았을 것이다.

- I am처럼 am을 포함한 시퀀스
- o를 포함하고 o가 분리된 시퀀스
- tabloid처럼 oid를 포함한 단어가 있는 시퀀스

전혀 좋은 소식이 아니다. 다음 단어를 더 살펴보자.

```
amoeboid and mesenchymal
```

출력은 잘 표시되지만, 토큰 결과가 혼란스럽다.

```
Ġam : 716
o : 78
eb : 1765
oid : 1868
Ġand : 290
Ġmes : 18842
ench : 24421
ym : 4948
al : 282
```

왜 문제가 되는지 와닿지 않을 수 있다. 이유를 요약하자면, 바로 다의성(polysemy) 때문이다. word2vec 토큰라이저를 사용하면 amoeboid 같은 희귀한 단어는 제외되고 언논 토큰으로 처리된다.

바이트 단위 BPE를 사용하면, go와 go+ing처럼 단어의 변형을 더 잘 표현하며 전반적으로 좋은 성능을 보일 수 있다.

하지만, amoeboid의 am 토큰 같은 경우는 낮은 수준에서 다의성 문제를 유발한다. am은 일종의 접두사로 am+bush의 하위 단어가 되기도 하지만, I+am과 같이 사용된다. 어텐션 층은 두 종류의 am에 대해서 존재하지 않는 연관성을 찾으려 할 것이다. 이러한 다의성은 NLU에서 중요한 문제다.

진전이 있지만, NLP를 개선하기 위해 더 열심히 노력해야 한다.

몇 가지 예제로 실제 프로젝트에서 직면하는 일상적인 문제를 확인했다. 유용하다고 생각하는 예제를 시간을 내서 실행해 보자.

마지막으로, 트랜스포머 모델의 NLU 수준을 확인하는 작업을 진행해 보겠다.

## 9.3

# GPT-3의 범위 살펴보기

OpenAI GPT-3처럼 매우 강력한 트랜스포머도 한계가 존재한다. GPT-3가 amoeboid라는 희귀한 의학 용어에 어떻게 대응하는지 확인해 보자. 많은 프로젝트에서 기술적인 전문용어를 만나게 된다. 데이터셋에 최적화하려면 트랜스포머 단어 사전과 임베딩의 품질을 관리해 주어야 한다.

인간은 문제를 감지하고 바로잡을 수 있다. 예를 들어, 우리는 이번 챕터의 토큰화된 데이터 확인하기 섹션에서 amoeboid의 문제를 감지했다.

먼저 GPT-3에게 amoeboid의 의미를 물어보자.

## Playground

**Q: What is the definition of amoeboid?**

그림 9.4 GPT-3에게 "amoeboid"의 의미를 물어보기

amoeboid는 "아메바를 닮은"이라는 형용사이지만, GPT-3는 다음과 같이 명사라고 답했다.

A: 아메보이드는 "아메바를 닮은"이라는 뜻의 명사이다.

GPT-3에게 더 자세히 질문해도 여전히 틀린 답변을 한다.

Q: 아메보이드는 명사인가, 형용사인가?

A: 아메보이드는 명사이다.

마지막으로, 명확한 정의를 요구하자 정확한 답을 얻었다.

Q: 아메보이드는 의학 용어로 무엇을 의미하는가?

A: 아메보이드는 "아메바와 닮았다"는 뜻이다.

정의는 잘 알고 있지만, 문법 분석엔 약한 모습이다.

실제 프로젝트라면, 단어의 정의를 아는 것과 형용사나 명사 같은 문장 속 역할을 아는 것 중 어느 쪽이 더 중요할까?

#### NOTE

의학 프로젝트라면 단어의 정의만 알아도 충분할 수 있다. 이 경우라면, GPT-3도 충분하다. 정의로도 충분하다면, 문장을 이해하기 위해 SRL(Semantic Role Labeling, 의미역 결정)이 필수적인 것은 아니다.

문법 교육을 위한 학교 프로젝트라면 문법적인 측면이 더 중요할 것이다. 하지만 공급망 관리나 금융, 전자상거래 프로그램이라면 그렇지 않다.

**7장, GPT-3 엔진을 사용한 초인간 트랜스포머 등장**에서 어느 경우에도 OpenAI GPT-3를 미세 조정할 수 있음을 확인했다.

이번 섹션의 결론은 학습된 트랜스포머 모델에 필요한 데이터가 모두 있는지 확인해야 한다는 것이다. 그렇지 않다면, 토큰화 작업이 불완전할 것이다. 의학 사전을 가져오고 거대한 의학 문서 코퍼스를 만들어서 특정 어휘를 포함시켜야 할지도 모른다. 그 후에도 모델이 충분히 정확하지 않다면 데이터셋을 토큰화하고 모델을 처음부터 학습시켜야 한다.

2024년의 개발자는 개발 업무가 줄어든 대신 많은 것을 생각하고 설계해야 한다.

이번 장을 마무리하고 다른 NLU 작업으로 넘어가겠다.

이번 장은 토큰화와 이어지는 데이터 인코딩 작업이 트랜스포머 모델에 어떤 영향을 주는지 확인했다. 트랜스포머는 스택 내의 임베딩 서브 층과 위치 인코딩 서브 층을 통과한 토큰만 사용한다. 모델이 인코더-디코더 구조인지, 인코더 또는 디코더만 사용했는지는 중요하지 않다. 데이터셋이 학습하기 좋아 보이는지는 중요하지 않다.

토큰화 과정이 일부라도 실패하면, 트랜스포머 모델은 중요한 토큰을 놓칠 것이다.

일반적인 언어 작업에서, 원시 데이터로도 트랜스포머 모델을 학습하기 충분함을 확인했었다.

하지만, 10억 개의 단어로 토큰라이저를 학습해도, 극히 일부 어휘로 사전을 생성하는 것 또한 확인했다. 우리처럼, 토큰라이저는 학습 중인 언어의 본질을 파악한다. 토큰라이저는 가장 중요하고 자주 사용된 단어만 기억하는 것이다. 이러한 접근 방식은 일반적인 작업에서 잘 동작하지만, 특정 작업이나 어휘에서는 문제가 발생한다.

토큰라이저의 한계를 극복하기 위한 다양한 방법 중 몇 가지를 살펴보았다. 토큰라이저가 데이터를 어떻게 생각하고 인코딩하는지 고려하며, 처리하고자 하는 텍스트에 맞게 언어를 검사하는 방법을 적용했다.

GPT-2에 조건 여부에 따른 작업을 요청하고 결과를 확인했다.

마지막으로, 토큰화 및 데이터셋 최적화와 관련하여 GPT-3의 한계를 확인했다. 이번 장은 AI 전문가가 앞으로 꽤 오랫동안 존재할 것임을 암시한다!

다음 10장, BERT 기반 트랜스포머를 사용한 SRL에서는 NLU를 깊게 살펴보고 BERT 모델에게 문장의 의미를 물어볼 것이다.

## 9.5 문제

01. 토큰화 사전은 언어에 존재하는 모든 단어가 들어있다. 참 / 거짓
02. 학습된 토큰라이저는 어떤 데이터셋도 인코딩할 수 있다. 참 / 거짓
03. 데이터셋을 사용하기 전에 미리 검토하는 것이 좋다. 참 / 거짓
04. 데이터셋에서 외설적인 데이터를 제거하는 것이 좋다. 참 / 거짓
05. 차별적인 주장을 하는 데이터를 제거하는 것이 좋다. 참 / 거짓
06. 원시 데이터셋을 그대로 사용하면 노이즈 데이터와 유용한 데이터 사이에 연관성이 생길 수 있다. 참 / 거짓
07. 일반적인 사전 학습된 토큰라이저에는 700년 전 어휘가 들어있다. 참 / 거짓
08. 현대 영어를 학습한 토큰라이저로 오래된 영어를 인코딩하면 문제가 발생할 수 있다. 참 / 거짓
09. 현대 영어를 학습한 토큰라이저로 의학 등 전문 분야 데이터를 인코딩하면 문제가 발생할 수 있다. 참 / 거짓
10. 학습된 토큰라이저의 인코딩 결과를 직접 확인하는 것이 좋다. 참 / 거짓

## 9.6

## 참고 문헌

- Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu, 2019, Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer: <https://arxiv.org/pdf/1910.10683.pdf>
- OpenAI의 GPT-2 깃허브 저장소: <https://github.com/openai/gpt-2>
- N. Shepperd의 깃허브 저장소: <https://github.com/nshepperd/gpt-2>
- 허깅페이스 프레임워크 및 리소스: <https://huggingface.co/>
- 미국 법률, 몬태나 주 기업법: <https://corporations.uslegal.com/statecorporation-law/montana-corporation-law/#:~:text=Montana%20Corporation%20Law,carrying%20out%20its%20business%20activities>
- Martina Conte, Nadia Loy, 2020, Multi-cue kinetic model with non-local sensing for cell migration on a fibers network with chemotaxis: <https://arxiv.org/abs/2006.09707>
- 토머스 제퍼슨(Thomas Jefferson)의 미국 독립선언: <https://www.gutenberg.org/ebooks/1>
- 미국 권리장전과 관련 텍스트: <https://www.gutenberg.org/ebooks/2>
- 마그나 카르타: <https://www.gutenberg.org/ebooks/10000>
- 순수이성비판, 실천이성비판, 윤리형이상학 정초: <https://www.gutenberg.org>

the 1990s, the number of people in the world who are under 15 years of age is expected to increase from 1.1 billion to 1.5 billion.

There are a number of reasons why the world's population is growing so rapidly. One of the main reasons is that the number of children born to each woman has increased. This is due to a number of factors, including:

- The fact that women are having children at a younger age.
- The fact that women are having more children.

Another reason why the world's population is growing so rapidly is that the number of people who are surviving to old age has increased. This is due to a number of factors, including:

- The fact that people are living longer.
- The fact that people are having fewer children.

The world's population is growing so rapidly that it is expected to reach 7 billion by the year 2011. This is a significant increase from the 5 billion people who lived in the world in 1987.

The world's population is growing so rapidly that it is expected to reach 9 billion by the year 2050. This is a significant increase from the 5 billion people who lived in the world in 1987.

The world's population is growing so rapidly that it is expected to reach 10 billion by the year 2050. This is a significant increase from the 5 billion people who lived in the world in 1987.

The world's population is growing so rapidly that it is expected to reach 11 billion by the year 2050. This is a significant increase from the 5 billion people who lived in the world in 1987.

The world's population is growing so rapidly that it is expected to reach 12 billion by the year 2050. This is a significant increase from the 5 billion people who lived in the world in 1987.

The world's population is growing so rapidly that it is expected to reach 13 billion by the year 2050. This is a significant increase from the 5 billion people who lived in the world in 1987.

The world's population is growing so rapidly that it is expected to reach 14 billion by the year 2050. This is a significant increase from the 5 billion people who lived in the world in 1987.

The world's population is growing so rapidly that it is expected to reach 15 billion by the year 2050. This is a significant increase from the 5 billion people who lived in the world in 1987.

The world's population is growing so rapidly that it is expected to reach 16 billion by the year 2050. This is a significant increase from the 5 billion people who lived in the world in 1987.

The world's population is growing so rapidly that it is expected to reach 17 billion by the year 2050. This is a significant increase from the 5 billion people who lived in the world in 1987.

The world's population is growing so rapidly that it is expected to reach 18 billion by the year 2050. This is a significant increase from the 5 billion people who lived in the world in 1987.

The world's population is growing so rapidly that it is expected to reach 19 billion by the year 2050. This is a significant increase from the 5 billion people who lived in the world in 1987.

The world's population is growing so rapidly that it is expected to reach 20 billion by the year 2050. This is a significant increase from the 5 billion people who lived in the world in 1987.

The world's population is growing so rapidly that it is expected to reach 21 billion by the year 2050. This is a significant increase from the 5 billion people who lived in the world in 1987.

The world's population is growing so rapidly that it is expected to reach 22 billion by the year 2050. This is a significant increase from the 5 billion people who lived in the world in 1987.

The world's population is growing so rapidly that it is expected to reach 23 billion by the year 2050. This is a significant increase from the 5 billion people who lived in the world in 1987.

The world's population is growing so rapidly that it is expected to reach 24 billion by the year 2050. This is a significant increase from the 5 billion people who lived in the world in 1987.

The world's population is growing so rapidly that it is expected to reach 25 billion by the year 2050. This is a significant increase from the 5 billion people who lived in the world in 1987.