

10장

BERT 기반 트랜스포머를 사용한 SRL

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

트랜스포머는 지난 몇 년 동안 지난 세대 NLP보다 더 많은 발전을 이루었다. 기존의 NLU 접근 방식은 먼저 문장 구조를 설명하기 위해 구문과 어휘의 특징을 학습하는 것이었다. 이에 따라 SRL(Semantic Role Labeling)을 실행하기 전에 언어의 기본 구문을 이해하도록 NLP 모델을 학습하는 것이 기존 방식이었다.

2019년 “스(Shi)”와 “린(Lin)”의 논문은 예비 구문 및 어휘 학습을 건너뛸 수 있는지에 대한 질문으로 시작한다. BERT 기반 모델이 이러한 고전적인 훈련 단계를 거치지 않고도 SRL을 수행할 수 있을까? 답은 ‘그렇다’이다.

2019년 “스(Shi)”와 “린(Lin)”(2019)은 SRL을 시퀀스 라벨링으로 간주할 수 있고 SRL이 표준화된 입력 형식을 제공할 수 있다고 제안했다. 그들의 BERT 기반 모델은 놀라울 정도로 좋은 결과를 도출했다.

이 장에서는 스와 린의 논문을 기반으로 앨런 인공지능 연구소(Allen Institute for AI)에서 제공한 사전 학습된 BERT 기반 모델을 사용한다. 스와 린은 구문 및 어휘 학습을 생략함으로써 SRL을 한 단계 더 발전시켰다. 어떻게 이것이 가능했는지 살펴볼 것이다.

먼저 SRL과 시퀀스 라벨링 입력 형식을 정의하는 것으로 시작한다. 그런 다음 앨런 인공지능 연구소의 리소스를 사용한다. 이후 구글 코랩(Google Colab) 노트북에서 SRL 작업을 실행하고 결과를 이해해 보자.

마지막으로, SRL 샘플을 실행하여 BERT 기반 모델을 시험해 보자. 첫 번째 샘플로 SRL이 어떻게 작동하는지 확인하고, 좀 더 어려운 샘플을 실행해 볼 것이다. 우리는 점진적으로 BERT 기반 모델을 SRL의 한계까지 밀어붙일 것이다. 트랜스포머 모델의 한계를 측정함으로써, 모델을 현실적이고 실용적으로 유지할 수 있다.

이 장에서는 다음 주제들을 다룬다.

- SRL 정의하기
- SRL 입력 형식 표준화 정의하기
- BERT 기반 모델 아키텍처의 주요 측면
- 인코더 전용 스택이 마스킹된 SRL 입력 형식을 다루는 법
- BERT 기반 모델 SRL 어텐션 프로세스
- 앨런 인공지능 연구소에서 제공하는 리소스 사용하기
- 사전 학습된 BERT 기반 모델을 실행하기 위한 노트북 구축하기
- 기본 예제 문장 라벨링 테스트하기
- 어려운 예제에서 SRL 테스트하기 및 결과 설명하기
- BERT 기반 모델을 SRL 한계까지 사용하고 어떻게 수행됐는지 설명하기

첫 번째 단계는 스와 린이 정의한 SRL 접근 방식을 살펴보는 것이다.

10.1

SRL(Semantic Role Labeling, 의미역 결정)

SRL은 기계만큼이나 인간에게도 어려운 작업이다. 하지만 트랜스포머는 다시 한번 인간의 기준선에 한 걸음 더 다가섰다. 이 챕터에서 먼저 SRL을 정의하고 예제를 시각화한다. 그런 다음 사전 학습된 BERT 기반 모델을 실행한다.

먼저 SRL의 문제 작업을 정의하는 것으로 시작하자.

10.1.1 SRL 정의하기

2019년 스와 린은 어휘나 구문적 특징에 의존하지 않고도 누가, 어디서, 무엇을 했는지 찾을 수 있다는 아이디어를 발전시키고 증명했다. 이 챕터는 캘리포니아 워털루 대학교의 “펑 스(Peng Shi)”와 “지미 린(Jimmy Lin)”의 연구에 기반한다. 이 연구는 트랜스포머가 어떻게 어텐션 층을 활용해 언어 구조를 잘 학습하는지 보여주었다.

SRL은 단어 또는 단어 그룹이 문장에서 수행하는 역할과 서술어와 맺는 관계로 의미역(semantic roles)을 분류하는 작업이다.

의미역이란 명사 또는 명사구가 문장에서 주동사(main verb)와 관련하여 수행하는 역할을 말한다. 예를 들어 *Marvin walked in the park*(마빈이 공원에서 걸었다.)라는 문장에서 *Marvin*은 문장에서 발생하는 사건의 행위자(agent)이다. 행위자는 이벤트를 수행하는 사람을 의미한다. 주동사 또는 지배동사(governing verb)는 *walked*이다.

서술어(predicate)는 주어 또는 행위자에 대한 정보를 설명한다. 서술어는 주어의 특징이나 행동에 대한 정보를 제공하는 모든 것이 될 수 있다. 이 책에서는 주동사(main verb)를 서술어라 할 것이다. 예를 들어, *Marvin walked in the park*에서 서술어는 *walked*이다.

*in the park*에 있는 단어들은 *walked*의 의미를 수식한다. 이런 단어들을 수식어라 한다.

서술어에 대한 명사 또는 명사구는 논항(arguments) 또는 논항 용어(argument terms)라 한다. 예를 들어 *Marvin*은 서술어 *walked*의 논항이다.

SRL에는 구문 트리(syntax tree)나 어휘 분석이 필요하지 않다는 것을 알 수 있다. SRL을 시각화해 보자.

10.1.1.1 SRL 시각화하기

이 챗터에서는 앨런 인공지능 연구소의 시각화 및 코드 리소스를 사용한다(자세한 내용은 참고 문헌 참조). 앨런 인공지능 연구소에는 이 챗터에서 SRL을 시각화하는 데 사용한 것과 같은 훌륭한 상호 작용 온라인 도구가 있다. <https://demo.allennlp.org/>에서 이러한 도구를 사용할 수 있다.

앨런 인공지능 연구소는 공익을 위한 인공지능을 추구한다. 우리는 이 가치를 잘 활용할 것이다. 이 챗터의 모든 그림은 AllenNLP 도구를 사용하여 생성됐다.

앨런 인공지능 연구소는 지속적으로 발전하는 트랜스포머 모델을 제공한다. 따라서 이 챗터의 예제를 실행하면 책과는 다른 결과가 나올 수 있다. 이 챗터를 활용하는 가장 좋은 방법은 다음과 같다.

- 프로그램을 단순히 실행하는 것 이상으로 개념을 읽고 이해한다.
- 예제를 이해하는 데 시간을 할애한다.

그런 다음 이 챗터에서 사용한 도구를 사용하여 원하는 문장으로 직접 실험을 해 보자.(<https://demo.allennlp.org/semantic-role-labeling>)²⁶

이제 SRL 예제를 시각화하자. 그림 10.1은 Marvin walked in the park를 시각화한 것이다.



그림 10.1 SRL의 시각화

그림 10.1에서 다음 라벨을 볼 수 있다.

- Verb(V): 문장의 서술어
- Argument: 논항0(ARG0)으로 표현되고, 문장의 논항
- Modifier(ARGM): 문장의 수식어. 이 경우에는 위치이다. 부사, 형용사 또는 서술어의 의미를 수정하는 모든 것이 가능하다.

²⁶ 역주. allennlp 데모의 서비스가 종료되었다. 실험을 원한다면 <https://hanlp.hankcs.com/en/demos/sr.html>에서 실행해보자.

시각화의 짧은 텍스트 버전도 제공한다.

```
walked: [ARG0: Marvin] [V: walked] [ARGM-LOC: in the park]
```

SRL을 정의하고 예제를 살펴봤다. 이제 BERT 기반 모델을 살펴보자.

10.1.2 사전 학습된 BERT 기반 모델 실행

이 섹션에서는 이 장에서 사용하는 BERT 기반 모델의 아키텍처를 설명하는 것으로 시작한다. 이후 BERT 모델을 사용해 SRL 샘플을 실험하는 방법을 정의한다.

먼저 BERT 기반 모델의 아키텍처를 살펴보자.

10.1.2.1 BERT 기반 모델의 아키텍처

AllenNLP의 BERT 기반 모델은 12개 층을 가진 인코더 전용 BERT 모델이다. 스와 린이 설명한 대로 AllenNLP 팀은 BERT 모델에 선형 분류 층을 추가했다.

필요한 경우, BERT 모델에 대한 자세한 설명은 3장, BERT 모델 미세 조정하기를 참고하자.

BERT 기반 모델은 간단한 접근 방식과 아키텍처로 양방향 어텐션(bidirectional attention)을 최대한 활용한다. 트랜스포머의 핵심 잠재력은 어텐션 층에 있다. 우리는 인코더와 디코더 스택이 모두 있는 트랜스포머 모델과 인코더 전용 혹은 디코더 전용 트랜스포머 모델 모두 살펴봤다. 트랜스포머의 가장 큰 장점은 인간에 가까운 어텐션 층의 접근 방식에 있다.

스와 린이 서술어 식별 작업의 입력 포맷을 정의했다. 이것을 보면 트랜스포머의 언어 이해 학습이 얼마나 표준화됐는지를 알 수 있다.

```
[CLS] Marvin walked in the park.[SEP] walked [SEP]
```

표준화된 학습 과정은 다음과 같다.

- [CLS]는 분류 작업임을 나타낸다.
- [SEP]는 문장의 끝을 나타내는 첫 번째 구분 기호이다.
- [SEP] 뒤에는 저자가 설계한 서술어 식별이 이어진다.
- [SEP]는 두 번째 구분 기호로, 서술어 식별자의 끝을 나타낸다.

이 형식만으로도 문장의 의미적 역할을 식별하고 라벨을 지정하도록 BERT 모델을 학습하기에 충분하다.

SRL 샘플을 실험하기 위한 환경을 설정해 보자.

10.1.2.2 BERT SRL 환경 설정

구글 코랩 노트북을 사용해 <https://demo.allennlp.org/>의 Semantic Role Labeling 탭에서 제공하는 SRL 시각화를 해볼 것이다.

다음 순서를 따를 것이다.

1. SRL.ipynb를 열고 AllenNLP를 설치하고 각 샘플 실행하기
2. 실행 결과 원시 출력(raw output)을 표시하기
3. AllenNLP의 온라인 시각화 도구를 사용하여 출력을 시각화하기
4. AllenNLP의 온라인 텍스트 시각화 도구를 사용해 출력을 표시하기

이 장은 다른 장의 내용을 필요로 하지 않는다. 설명을 읽고 샘플을 실행하면 된다. 모델의 출력은 달라질 수도 있다. 이는 AllenNLP가 트랜스포머 모델을 지속적으로 업데이트하기 때문이다. 또한 학습에 사용하는 데이터셋이 변경될 수도 있기 때문이다. 마지막으로, 매번 동일한 결과를 생성하는 규칙 기반 알고리즘이 아니기 때문이다. 앞서 그림에서 표시된 것처럼 실행할 때마다 출력이 변경될 수 있다.

이제 몇 가지 SRL 실험을 실행해 보자.

10.2 BERT 기반 모델을 사용한 SRL 실험

이 챕터의 섹션 BERT SRL 환경 설정에서 설명한 순서로 실험을 실행할 것이다. 다양한 문장 구조를 가진 기본 예시들로 시작할 것이다. 그런 다음 시스템의 용량과 한계를 탐색하기 위해 좀 더 어려운 샘플을 도전해 보자.

SRL.ipynb를 열고 라이브러리 설치 셀(cell)을 실행하자.

```
!pip install allennlp==2.1.0 allennlp-models==2.1.0
```

그런 다음 태깅 모듈과 학습된 BERT 예측기(Predictor)를 불러오자.

```
from allennlp.predictors.predictor import Predictor
import allennlp_model_staging
import json

predictor = Predictor.from_path("https://storage.googleapis.com/allennlp-public-models/structured-prediction-srl-bert.2020.12.15.tar.gz")
```

함수 두 개를 추가하여 SRL BERT가 반환하는 JSON 객체를 표시할 때 사용하자. 첫 번째 문장은 서술어의 동사와 설명을 표시한다.

```
def head(prediction):
    # json을 순회해 예측값의 발췌문 표시하기
    for i in prediction['verbs']:
        print('Verb:', i['verb'], i['description'])
```

두 번째는 태그를 포함한 전체 응답을 표시한다.

```
def full(prediction):
    #전체 예측 값 표시하기
    print(json.dumps(prediction, indent = 1, sort_keys=True))
```

이 책을 집필하는 시점에는 SRL에 특화된 BERT 모델인 “SRL BERT” 모델을 사용했다. SRL BERT는 OntoNotes 5.0 데이터셋(<https://catalog.ldc.upenn.edu/LDC2013T19>)으로 학습됐다.

이 데이터셋에는 문장과 주석이 들어있다. 이 데이터셋은 문장에서 서술어(동사를 포함하는 문장의 일부)를 식별하고 동사에 대한 자세한 정보를 제공하는 단어를 식별하도록 설계됐다. 각 동사에는 동사에 대한 자세한 정보를 알려주는 “논항”이 함께 들어있다. “프레임”은 동사의 논항을 담고 있다.

따라서 SRL BERT는 특정작업을 수행하도록 학습된 특수 모델이며, **7장, GPT-3 엔진을 사용한 초인간 트랜스포머 등장**에서 살펴본 OpenAI GPT-3와 같은 파운데이션 모델이 아니다.

SRL BERT는 문장에 서술어가 포함되어 있는 한 허용 가능한 정확도로 SRL을 수행할 것이다.

이제 몇 가지 기본 예시로 시작해 보자.

10.3 기본 예시

기본 예시는 직관적으로 단순해 보이지만 분석하기 까다로울 수 있다. 복합 문장, 형용사, 부사, 조사는 전문가가 아닌 일반인도 식별하기 어렵다.

쉬운 예시로 시작해 보자.

10.3.1 예시 1

첫 예시는 길지만 트랜스포머에게는 비교적 쉽다.

Did Bob really think he could prepare a meal for 50 people in only a few hours?(밥은 정말 단 몇 시간 만에 50명을 위한 식사를 준비할 수 있다고 생각했을까?)

SRL.ipynb에서 Sample 1 셀을 실행하자.

```
prediction=predictor.predict(  
    sentence="Did Bob really think he could prepare a meal for 50 people in  
    only a few hours?"  
)  
head(prediction)
```

BERT SRL은 네 개의 서술어를 식별했다. head 함수로 각 서술어를 표시하면 다음과 같다.

Verb: Did [V: Did] Bob really think he could prepare a meal for 50 people in only a few hours ?

Verb: think Did [ARG0: Bob] [ARGM-ADV: really] [V: think] [ARG1: he could prepare a meal for 50 people in only a few hours] ?

Verb: could Did Bob really think he [V: could] [ARG1: prepare a meal for 50 people in only a few hours] ?

Verb: prepare Did Bob really think [ARG0: he] [ARGM-MOD: could] [V: prepare] [ARG1: a meal for 50 people] [ARGM-TMP: in only a few hours] ?

full 함수를 실행하면 전체 응답을 볼 수 있다.

- PropBank(Proposition Bank) 데이터셋의 요소들로, 이 문장을 think를 중심으로 표현하면 다음과 같다.
- V는 동사 think를 가리킨다.
- ARG0는 행위자를 가리킨다. 따라서 Bob은 행위자 또는 프로-에이전트(pro-agent)이다.²⁷
- ARGM-ADV는 really를 부사(ADV)로 간주한다. ARGM은 부가어(adjunct, 문장이나 절 혹은 구에서 선택적이거나 구조적으로 없어도 되는 부분)를 가리킨다. 따라서 번호를 매기지 않는다.

AllenNLP 온라인 인터페이스에서 샘플을 실행하면 각 동사에 대한 SRL 작업의 시각화 결과(프레임)를 얻을 수 있다 첫 번째 동사는 Did이다.

Did의 **Frames**:

Did
서술어

Bob really think he could prepare a meal for 50 people in only a few hours?

그림 10.2 동사 Did 식별하기

27 역주. 프로-에이전트는 행위자(agent)를 대신해 동작을 수행하는 역할을 하는 개체를 가리킨다. 예를 들면 He loves to swim에서 He는 프로-에이전트이다. John loves to swim에서의 John같은 실제 행위자를 대신하기 때문이다.

두 번째 동사는 think이다.

think의 Frames :



그림 10.3 동사 think 식별하기

이 표현을 자세히 살펴보면 몇가지 흥미로운 점이 있다.

- 동사 think를 감지했다.
- 주 동사로 해석될 수 있었던 prepare 함정을 피했다. 대신 prepare는 think의 논항의 일부로 남겨졌다.
- 부사(really)를 감지하고 라벨링을 했다.

세 번째 동사는 could이다.

could의 Frames :

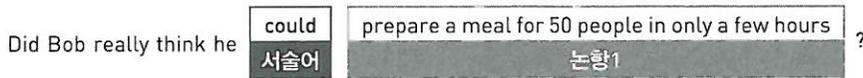


그림 10.4 동사 "could"와 논항 식별하기

그런 다음 트랜스포머는 동사 prepare로 이동하여 라벨링을 하고 문맥을 분석했다.

prepare의 Frames :

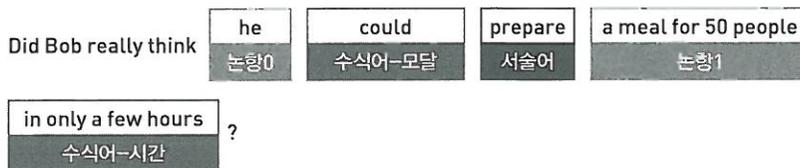


그림 10.5 동사 "prepare". 논항, 수식어 식별하기

다시 말하지만, 간단한 BERT 기반 트랜스포머 모델로 문장의 문법 구조에 대한 많은 정보를 감지했다.

- 동사 prepare를 발견하고 분리했다.
- 명사 he를 발견하고 논항으로 라벨링하고, a meal for 50 people에 대해서도 동일하게 수행했다. a meal for 50 people은 원시 수동자(proto-patient, 다른 참여자에 의한 수식을 수반하는 수동자)이다.
- in only a few hours가 시간적 수식어(ARGM-TMP) 임을 찾았다.
- modal modifier인 could를 발견했다.²⁸

비교적 긴 문장을 하나 더 분석해 보자.

10.3.2 예시 2

다음 문장은 쉬워 보이지만 여러 동사가 들어있다.

Mrs. and Mr. Tomaso went to Europe for vacation and visited Paris and first went to visit the Eiffel Tower.(토마소씨와 부인은 휴가에 유럽을 가서 에펠탑을 방문하기 위해 먼저 파리에 방문했다.)

이 헛갈리는 문장이 트랜스포머를 고민하게 만들까? SRL.ipynb의 Sample 2 셀을 실행해보자.

```
prediction=predictor.predict(
    sentence="Mrs. and Mr. Tomaso went to Europe for vacation and visited
    Paris and first went to visit the Eiffel Tower."
)
head(prediction)
```

출력을 보니, 트랜스포머가 문장의 동사를 올바르게 식별했음을 알 수 있다.

²⁸ 역주. modal modifier란, 동사나 형용사 앞에 올 수 있는 단어나 구문으로, 동사나 형용사의 의미를 변경하거나 강조하는 역할을 한다. 예를 들어 can, may, should, must 등이 modal modifier이다. "She can swim"이랑 "She must swim"은 뉘앙스가 다르다. 전자는 그녀가 수영을 할 수 있는 능력이 있다는 것을 나타내지만, 후자는 그녀가 반드시 수영을 해야 한다는 것을 강조한다. "The movie was surprisingly good."에서 surprisely는 형용사 good을 강조하는 modal modifier이다. "The weather is really cold today."에서 really는 cold를 강조하는 modal modifier이다.

Verb: went [ARG0: Mrs. and Mr. Tomaso] [V: went] [ARG4: to Europe] [ARGM-PRP: for vacation] and visited Paris and first went to visit the Eiffel Tower .

Verb: visited [ARG0: Mrs. and Mr. Tomaso] went to Europe for vacation and [V: visited] [ARG1: Paris] and first went to visit the Eiffel Tower .

Verb: went [ARG0: Mrs. and Mr. Tomaso] went to Europe for vacation and visited Paris and [ARGM-TMP: first] [V: went] [ARG1: to visit the Eiffel Tower].

Verb: visit [ARG0: Mrs. and Mr. Tomaso] went to Europe for vacation and visited Paris and first went to [V: visit] [ARG1: the Eiffel Tower].

AllenNLP 온라인에서 샘플을 실행한 결과, 네 개의 서술어가 식별되어 네 개의 프레임이 생성됐다. 첫 번째 프레임은 went에 대한 프레임이다.

went의 **Frames** :



그림 10.6 동사 "went", 논항, 수식어 식별하기

동사 went의 논항을 해석할 수 있다. Mrs. and Mr. Tomaso는 행위자이다. 트랜스포머는 동사의 주요 수식어가 여행의 목적인 유럽(to Europe)인 것을 발견했다. 스와 린이 고품질 문법 분석을 얻기 위해 간단한 BERT 모델 구축만 했다는 사실을 생각하면 아주 놀라운 결과이다.

went가 정확하게 Europe과 연관됐다는 것도 알 수 있다. 트랜스포머가 정확하게 동사 visited가 Paris와 관련된 것으로 식별했다.

visited의 **Frames** :



그림 10.7 동사 "visited", 논항 식별하기

트랜스포머는 동사 visited를 에펠탑(Eiffel Tower)과 직접 연결할 수 있었지만 그렇게 하지 않았다.

트랜스포머에게 요청한 다음 과제는 동사 went의 두 번째 사용 문맥을 파악하는 것이다. 두 번 사용된 동사 went와 관련된 모든 논항을 병합하는 함정에 빠지지 않았다. 즉, 시퀀스를 올바르게 분할하여 훌륭한 결과를 얻어냈다.

went의 Frames:

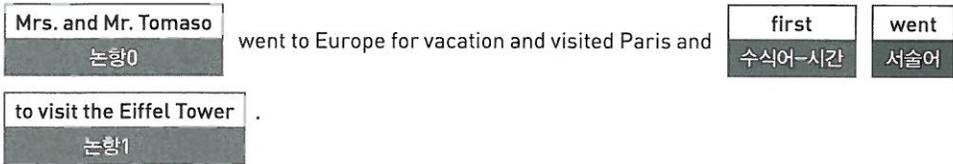


그림 10.8 동사 "went", 논항, 수식어 식별하기

동사 went는 두 번 사용됐지만 트랜스포머는 함정에 빠지지 않았다. 심지어 first가 동사 went의 시간적 수식어(ARGM-TMP, 수식어-시간)라는 것도 발견했다.

마지막으로, 동사 visited가 두번째로 사용됐고, SRL BERT는 그 용법을 정확하게 해석했다.

visite의 Frames:

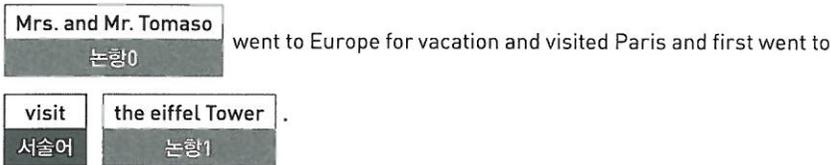


그림 10.9 동사 "visited", 논항 식별하기

이제 좀 더 헷갈리는 문장을 실행해 보자.

10.3.3 예시 3

트랜스포머에게 예시 3이 더 어려울 것이다. 왜냐하면 동사 drink의 변형을 네 번이나 포함하기 때문이다.

John wanted to drink tea, Mary likes to drink coffee but Karim drank some cool water and Faiza would like to drink tomato juice.(존은 차를 마시고 싶었고, 마리는 커피를 좋아하지만 카림은 시원

한 물을 마시고 싶었고, 파이자는 토마토 주스를 마시고 싶었다.)

SRL.ipynb 노트북에서 Sample 3 셀을 실행해 보자.

```
prediction=predictor.predict(  
    sentence="John wanted to drink tea, Mary likes to drink coffee but  
    Karim drank some cool water and Faiza would like to drink tomato juice."  
)  
head(prediction)
```

동사를 포함하는 출력 부분을 보니, 트랜스포머는 정확하게 수행했음을 알 수 있다.

```
Verb: wanted [ARG0: John] [V: wanted] [ARG1: to drink tea], Mary likes to  
drink coffee but Karim drank some cool water and Faiza would like to drink  
tomato juice.
```

```
Verb: drink [ARG0: John] wanted to [V: drink] [ARG1: tea], Mary likes to  
drink coffee but Karim drank some cool water and Faiza would like to drink  
tomato juice.
```

```
Verb: likes John wanted to drink tea, [ARG0: Mary] [V: likes] [ARG1: to  
drink coffee] but Karim drank some cool water and Faiza would like to  
drink tomato juice.
```

```
Verb: drink John wanted to drink tea, [ARG0: Mary] likes to [V: drink]  
[ARG1: coffee] but Karim drank some cool water and Faiza would like to  
drink tomato juice.
```

```
Verb: drank John wanted to drink tea, Mary likes to drink coffee but  
[ARG0: Karim] [V: drank] [ARG1: some cool water] and Faiza would like to  
drink tomato juice.
```

```
Verb: would John wanted to drink tea, Mary likes to drink coffee but  
Karim drank some cool water and [ARG0: Faiza] [V: would] like [ARG1: to  
drink tomato juice].
```

```
Verb: like John wanted to drink tea, Mary likes to drink coffee but Karim  
drank some cool water and [ARG0: Faiza] [ARGM-MOD: would] [V: like] [ARG1:  
to drink tomato juice].
```

Verb: drink John wanted to drink tea, Mary likes to drink coffee but Karim drank some cool water and [ARG0: Faiza] would like to [V: drink] [ARG1: tomato juice].

온라인 인터페이스에서 문장을 실행하면 여러 시각화를 얻을 수 있다. 그중 두 가지를 살펴보자.

첫 번째는 완벽하다. 동사 wanted를 식별하고 연관성을 올바르게 찾아냈다.

wanted의 **Frames** :

John	wanted	to drink tea
논항0	서술어	논항1

그림 10.10 동사 wanted, 논항 식별하기

동사 drank를 식별했을 때, 파이자를 정확하게 배제하고 오로지 some cool water만 논항으로 라벨링했다.

Karim	drank	some cool water
논항0	서술어	논항1

그림 10.11 동사 drank, 논항 식별하기

지금까지 BERT 기반 트랜스포머가 비교적 좋은 결과를 내는 것을 확인했다. 이제 좀 더 어려운 예시를 시도해 보자.

10.4 어려운 예시

이 섹션에서는 트랜스포머가 아직 해결하지 못한 문제를 갖고 있는 예시를 실행한다. 마지막으로 다루기 어려운 예시로 마무리할 것이다.

BERT 기반 트랜스포머가 분석할 수 있는 복잡한 예시를 시작해 보자.

10.4.1 예시 4

예시 4는 좀 더 까다로운 SRL 영역으로 안내한다. 이 예시에서는 Alice가 동사 “liked”와 분리되어 있어, whose husband went jogging every Sunday를 넘는 장기 의존성을 필요로 한다.

문장은 다음과 같다.

Alice, whose husband went jogging every Sunday, liked to go to a dancing class in the meantime. (앨리스는 남편이 매주 일요일마다 조깅을 하는 동안에 댄스 수업에 가는 것을 좋아했다.)

사람은 Alice를 분리하고 서술어를 찾을 수 있다.

Alice liked to go to a dancing class in the meantime. (앨리스는 그 동안에 댄스 수업에 가는 것을 좋아했다.)

BERT 모델도 우리와 같이 서술어를 찾을 수 있을까?

SRL.ipynb에서 코드를 실행하여 확인해 보자.

```
prediction=predictor.predict(
    sentence="Alice, whose husband went jogging every Sunday, liked to go
to a dancing class in the meantime."
)
head(prediction)
```

각 서술어에 대한 동사의 식별 결과와 각 프레임에 대한 라벨이 출력된다.

```
Verb: went Alice, [ARG0: whose husband] [V: went] [ARG1: jogging] [ARGM-TMP: every Sunday], liked to go to a dancing class in the meantime.

Verb: jogging Alice, [ARG0: whose husband] went [V: jogging] [ARGM-TMP: every Sunday], liked to go to a dancing class in the meantime.

Verb: liked [ARG0: Alice, whose husband went jogging every Sunday], [V: liked] [ARG1: to go to a dancing class in the meantime].

Verb: go [ARG0: Alice, whose husband went jogging every Sunday], liked to [V: go] [ARG4: to a dancing class] [ARGM-TMP: in the meantime].

Verb: dancing Alice, whose husband went jogging every Sunday, liked to go to a [V: dancing] class in the meantime.
```

모델이 서술어를 찾았는지 확인해 보자. 동사 liked를 찾았고, 동사 liked가 다른 서술어로 인해 Alice와 떨어져 있었음에도, liked를 잘 찾아냈다.

Verb: liked [ARG0: Alice, whose husband went jogging every Sunday]

이제 AllenNLP 온라인 UI에서 예시를 실행한 후 시각화 결과를 보자. 먼저 트랜스포머가 Alice의 남편(husband)을 찾는다.



그림 10.12 서술어 "went"가 식별된 모습

트랜스포머는 다음과 같이 설명한다.

- 서술어는 went이다.
- whose husband는 논항이다.
- jogging은 went와 관련된 또 다른 논항이다.
- every Sunday는 [AGM-TMP: every Sunday]로 표현된 것처럼 시간적 수식어이다.

그리고 트랜스포머는 Alice의 남편이 무엇을 하고 있는지 알아낸다.

jogging의 **Frames**:



그림 10.13 동사 "jogging"에 대한 SRL 탐지

동사 jogging이 식별되고 이 동사가 시간적 수식어 every Sunday와 함께 whose husband랑 연관되었다는 것을 알 수 있다. 트랜스포머는 여기서 멈추지 않고 Alice가 좋아하는 것을 감지한다.

liked의 **Frames**:



그림 10.14 동사 "liked" 식별하기

트랜스포머는 동사 go 또한 올바르게 감지하고 분석한다.

go의 **Frames**:

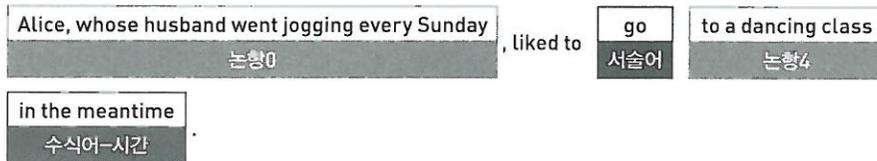


그림 10.15 동사 "go"와 그 논항 및 수식어 감지하기

시간적 수식어 the meantime도 식별된 것을 알 수 있다. SRL BERT가 단순히 시퀀스+동사를 학습한 것을 고려하면 상당한 성능이다.

마지막으로, 트랜스포머는 마지막 동사 dancing을 class와 관련된 것으로 식별한다.

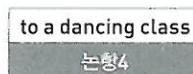


그림 10.16 논항 "class"를 동사 "dancing"에 관련짓기

예시 4의 결과는 꽤 설득력이 있다. 이제 트랜스포머 모델의 한계를 알아보자.

10.4.2 예시 5

예시 5는 동사를 여러 번 반복하지 않는다. 그러나 예시 5에는 여러 의미를 가진 단어가 들어있다. 단어 round의 의미와 문법적 기능은 다양하다. round는 명사, 형용사, 부사, 타동사 또는 자동사가 될 수 있다.

타동사 또는 자동사로서 round는 perfection과 completion을 달성할 수 있다.²⁹ 이런 의미에서 round는 off와 함께 사용할 수 있다.

다음 문장은 과거 시제 round를 사용한다.

The bright sun, the blue sky, the warm sand, the palm trees, everything round off.(밝은 태양, 푸른 하늘, 따뜻한 모래, 야자수 등 모든 것이 완벽하게 어우러진다.)

동사 round는 “완벽하게 만들다”는 의미로 사용됐다. 물론 문법적으로 “rounded”가 맞지만, 결과를 살펴보자.

SRL.ipynb의 예시 5를 실행해 보자.

```
prediction=predictor.predict(
    sentence="The bright sun, the blue sky, the warm sand, the palm trees,
    everything round off."
)
head(prediction)
```

출력에 동사가 표시되지 않았다. 트랜스포머가 서술어(predicate)를 식별하지 못했다. 사실, 트랜스포머는 동사(verb)를 전혀 찾지 못했다.

```
"verbs": []
```

²⁹ 역주, perfection은 자동사 동작이 완벽하게 수행됐거나 완전하게 발생했음을 나타내는 것을 가리킨다. 예를 들어 He slept soundly에서 slept이 자동사이며 soundly가 perfection이다. completion은 타동사 동작이 완료됐거나 끝났음을 나타내는 것을 가리킨다. 예를 들어 She finished her homework에서 finished가 타동사이고 her homework가 completion을 나타내는 목적어이다.

반면, 온라인 버전은 동사를 찾아냈다. 온라인 버전이 문장을 더 잘 해석하는 것 같다.

everything	round	off
논항1	서술어	수식어-단어

그림 10.17 동사 round, 논항 everything 탐지하기

SRL 트랜스포머에게 친절하게 round에 현재 시제를 나타내는 s를 덧붙여 문장을 바꿔보자.

The bright sun, the blue sky, the warm sand, the palm trees, everything rounds off.

현재 시제로 SRL.ipynb를 다시 실행해보자.

```
prediction=predictor.predict(
    sentence="The bright sun, the blue sky, the warm sand, the palm trees,
    everything rounds off."
)
head(prediction)
```

이번에는 다음과 같이 서술어를 발견했다.

```
Verb: rounds [ARG1: The bright sun, the blue sky, the warm sand, the
palm trees, everything] [V: rounds] [ARGM-PRD: off].
```

온라인에서 같은 문장을 실행해 보면 아래의 시각화 결과를 얻을 수 있다.

The bright sun, the blue sky, the warm sand, the palm trees, everything	rounds	off
논항1	서술어	수식어-단어

그림 10.18 단어 "rounds"를 동사로 탐지하기

현재 시제 rounds로 쓰여지자 BERT 기반 트랜스포머 모델이 단어 round를 잘 탐지했다.

BERT 모델은 처음에 우리가 기대했던 결과를 생성하지 못했다. 하지만 입력 문장에 약간의 변화를 주자 잘 해결했다.

다음의 사실을 알 수 있다.

- 모델의 버전이 진화함에 따라 결과는 달라질 수 있다.
- 트랜스포머에게 무엇을 해야 하는지 알려줄 수 있는 인지적 측면의 노력이 곧 4차 산업 시대의 사고방식이다.

라벨을 붙이기 어려운 다른 예시를 시도해보자.

10.4.3 예시 6

예시 6은 우리가 흔히 명사라고 생각하는 단어를 사용한다. 하지만 많은 단어가 명사이면서 동사일 수도 있다. 예를 들어 to ice는 하키에서 링크를 가로질러 상대의 골라인 너머까지 퍽(puck, 아이스하키에서 공처럼 치는 고무 원반)을 쏘는 데 사용되는 동사이다.³⁰

아이스하키 코치는 팀에게 아이싱을 훈련하라고 지시하는 것으로 하루를 시작할 수 있다. 코치가 소리 지르는 명령문을 생각해 보자.

Now, ice pucks guys!(자, 이제 아이싱 훈련해, 애들아!)

guys는 성별에 관계없이 사람을 의미할 수 있다.

예시 6 셸을 실행시켜 보자.

```
prediction=predictor.predict(  
    sentence="Now, ice pucks guys!"  
)  
head(prediction)
```

결과는 트랜스포머가 동사를 찾지 못했다. “verbs”: [].

트랜스포머는 많은 발전을 했지만, 개발자가 모델을 개선할 여지가 여전히 많다는 것을 알 수 있다. 트랜스포머에게 무엇을 해야 하는지 알려주기위해 여전히 사람의 개입이 필요하다.

³⁰ 역주. 아이싱을 한다는 의미로, 아이스하키에서의 전문 용어이다.

온라인 인터페이스는 pucks를 동사로 혼동하고 있다.³¹

pucks의 **Frames** :

Now 수식어-시간	ice 논항2	pucks 서술어	guys 수식어-내용
---------------	------------	--------------	----------------

그림 10.19 pucks을 동사로 잘못 라벨링 하는 모델

다른 모델로 이런 문제를 해결할 수 있지만, 그럴지언정 또 다른 문제에 직면하게 될 것이다. 심지어 GPT-3도 여러분이 해결해야 할 문제가 있을 것이다.

NOTE

전문 용어나 기술 용어가 포함된 커스텀 애플리케이션을 위해 트랜스포머를 구현하다 보면, 어느 순간 다루기 힘든 한계에 도달하게 될 것이다.

이런 한계의 존재로, 프로젝트를 성공시키기 위해선 전문 지식이 필요하다. 따라서, 프로젝트를 성공시키기 위해서는 전문 사전을 만들어야 한다. 이것은 개발자에게는 희소식이다. 팀원들이 좋아할 여러 분야를 넘나드는 인지 기술을 개발할 수 있다는 얘기다.

몇 가지 예시를 직접 만들어 SRL의 한계를 어떻게 극복할 수 있는지 살펴보자. 그런 다음 전처리 기능을 개발해 트랜스포머에게 커스텀 애플리케이션에서 수행해야 할 작업을 보여주는 방법을 살펴보자.

이 장을 마무리 하기 전에, SRL의 동기(motivation)에 대해 알아보자.

31 역주, 번역을 하는 시점(2024년 3월)에는 HanLP, AllenNLP Semantic Role Labeling의 온라인 인터페이스에서 앞의 문장을 입력하면 동사를 찾지 못했다.

실제 프로젝트에서 우리를 도와줄 사람은 없다. 우리가 알아서 해야 하고, 만족시켜야 할 사람은 그 프로젝트를 의뢰한 사람뿐이다.

실용주의가 우선한다. 기술적 이념은 그다음이다.

2020년대에는 이전 AI 이념과 새로운 이념이 공존한다. 10년 후쯤에는 전자의 일부와 후자의 일부가 합쳐진 단 하나의 이념이 존재할 것이다.

이 섹션에서는 두 가지 측면으로 SRL의 생산성과 동기에 대해 의문을 가져보자:

- 서술어 분석의 한계
- 의미(semantic)라는 용어 사용에 대한 의문

10.5.1 서술어 분석의 한계

SRL은 서술어에 의존한다. SRL BERT는 동사를 제공하는 경우에만 동작한다. 하지만 동사가 없는 문장도 수없이 많다.

<https://demo.allennlp.org/>의 “Semantic Role Labeling”에서 단언을 입력하면 SRL BERT가 잘 동작한다.³²

하지만, 단언이 어떠한 물음에 대한 대답이라면 어떤 일이 벌어질까?

사람 1: What would you like to drink, please?

사람 2: A cup of coffee, please

사람 2의 대답을 입력해도 SRL BERT는 아무것도 찾지 못한다.

³² 역주. allennlp의 데모 서비스가 종료되었다. 실험을 원한다면 <https://hanlp.hankcs.com/en/demos/srl.html>에서 실행해보자.

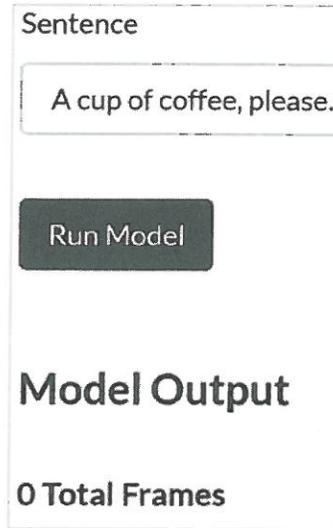


그림 10.20 프레임을 찾지 못한 경우

출력 프레임의 개수는 0개이다. 사람 2의 대답은 생략된 표현이 있어 SRL이 문장을 분석할 수가 없다. 이 경우, 서술어는 명시적(explicit)이지 않고 암시적(implicit)이다.

문장을 이해하는 데 필요하지 않은 단어는 문장에서 생략할 수 있다. 우리는 매일 생략한 문장을 사용한다. 하지만 SRL BERT는 이 모든 문장에 대해 프레임을 반환하지 못한다.

다음 what, where, how로 시작하는 질문(Q)에 대한 대답(A)은 모두 전체 프레임 개수가 0개이다.

Q: What would you like to have for breakfast?(아침 식사로 무엇을 드시겠습니까?)

A: Pancakes with hot chocolate.(핫초코를 곁들인 팬케이크요.)

(모델은 pancakes=고유 명사(proper noun), with=전치사(preposition), hot=형용사(adjective), chocolate=일반 명사(common noun)으로 추론한다.)

Q: Where do you want to go?(어디로 가고 싶으세요?)

A: London, please.(런던이요.)

(모델은 London=고유 명사, please=부사로 추론한다.)

Q: How did you get to work today?(오늘 어떻게 출근하셨나요?)

A: Subway.(지하철이요.)

(모델은 subway=고유 명사로 추론한다.)

NOTE

문장에 서술어가 없는, SRL BERT가 이해하지 못하는 예제를 수없이 만들 수 있다.

SRL BERT가 프레임을 하나도 얻지 못하는, 대화 중간에 발생하는 질문도 존재한다.

다음은 사람2가 커피를 원하지 않는다고 말한 이후의 대화이다.

Q: So, tea?(차 마실래요?)

A: No thanks.(아니요.)

Q: Ok, hot chocolate?(그럼 핫초코는요?)

A: Nope.(아니요.)

Q: A glass of water?(물 한 잔?)

A: Yup!(네!)

위 대화의 모든 질문은 프레임을 하나도 얻지 못한다.

소셜 미디어에 존재하는 영화, 콘서트, 전시회 리뷰에서도 프레임을 얻지 못하는 문장들을 볼 수 있다.

- Best movie ever!(역대 최고의 영화!)
- Worst concert in my life!(내 인생 최악의 콘서트!)
- Excellent exhibition!(훌륭한 전시회!)

이 섹션에서는 SRL 한계를 확인했다. 이제 SRL을 재정의하고 이를 구현하는 방법을 알아보자.

10.5.2 SRL 재정의하기

SRL BERT는 문장에 서술어가 포함됐다고 전제하지만, 많은 경우 이 가정을 따르지 않는다. 문장을 분석할 때 서술어 분석에만 의존할 수 없다.

서술어는 동사를 포함한다. 서술어는 주어에 대한 자세한 정보를 제공한다. 다음 서술어에는 동사와 추가 정보가 들어있다.

The dog ate his food quickly.(개가 음식을 빠르게 먹었다.)

ate ... quickly는 개가 먹는 방식에 대해 자세히 알려준다. 그러나 다음과 같이 동사만으로도 서술어가 될 수 있다.

Dogs eat.(개가 먹는다.)

문제는 “서술어”와 “동사”가 의미가 아닌 문법의 일부라는 것이다. 문법적, 기능적 관점에서 단어가 어떻게 서로 어울리는지 이해하는 것은 제한적이다.

전혀 의미가 없는 문장을 예를 들어 보자.

Globydisshing maccaked up all the tie.³³

SRL BERT는 아무 의미 없는 이 문장에 대해 “의미론적” 분석을 완벽하게 수행한다.



그림 10.21 의미 없는 문장 분석하기

몇 가지 결론을 도출할 수 있다.

- SRL 서술어 분석은 문장에 동사가 있을 때만 동작한다.
- SRL 서술어 분석은 생략된 내용을 식별할 수 없다.
- 서술어와 동사는 언어 구조 분석, 문법 분석의 일부이다.
- 서술어 분석은 구조를 식별하지만 문장의 의미는 식별하지 못한다.
- 문법 분석은 서술어 분석보다 그 이상의 것이 필요하다.

의미론(semantics)은 구문이나 문장의 의미에 초점을 맞춘다. 의미론은 문맥과 단어가 서로 연관되는 방식에 중점을 둔다.

문법 분석은 구문(syntax), 굴절(inflexion, 단어가 어떤 문법 범주를 나타내기 위해 모양을 바꾸는 현상, 예를 들어 동사가 시제에 따라 모양이 바뀌거나 명사가 주격, 목적격 등에 따라 모양을 바꾸는 것을 가리킴), 구(phrase) 또는 문장에서 단어의 기능 등을 포함한다. 의미역 결정(Semantic Role Labeling, SRL)이라는 용어는 오해의 소지가 있으므로 서술어 역할 결정(predicate role labeling)이라고 명명해야 한다.

인간은 서술어가 없는 문장이나 순서 구조를 넘어서는 문장도 완벽하게 이해한다.

감성 분석(sentiment analysis)은 서술어 분석 없이도 문장의 의미를 해독하고 출력을 제공할 수 있다.

33 역주. 의미를 알 수 없는 문장 예시, 존재하지 않는 단어들을 포함하고 있다.

감성 분석 알고리즘은 서술어의 존재 여부와 관계없이 “Best movie ever(역대 최고의 영화)”가 긍정적이라는 것을 완벽하게 이해한다.

NOTE

SRL만 사용해 언어를 분석하는 것은 제한적이다. AI 파이프라인에 도입하거나 다른 AI 도구와 함께 사용하면, 더 효과적으로 NLU를 발전시킬 것이다.

13장, 트랜스포머로 가짜 뉴스 분석하기에서 살펴볼 것처럼, 다른 AI 도구와 함께 SRL을 사용하는 것을 추천한다.

이제 SRL의 범위와 한계에 대한 탐구를 마무리짓겠다.

10.6 정리하기

이번 장에서는 SRL에 대해 살펴봤다. SRL 작업은 인간과 기계 모두에게 어렵다. 트랜스포머 모델은 많은 NLP 작업에서 인간의 기준선에 어느 정도 도달할 수 있음을 보여주었다.

간단한 BERT 기반 트랜스포머로 모호한 서술어 의미를 해소할 수 있었다. 어휘나 구문 라벨 없이 동사(서술어) 의미를 식별할 수 있는 간단한 트랜스포머를 실행해 봤다. 스와 린은 표준 “문장+동사” 입력 형태를 사용해 BERT 기반 트랜스포머를 학습시켰다. SRL만 사용해 언어를 분석하는 것은 제한적이다. AI 파이프라인이나 다른 AI 도구와 함께 SRL을 사용하면 NLU에 더 많은 지능을 추가할 수 있다.

필요한 것만 남긴 문장+서술어 입력으로 학습한 트랜스포머가 단순하거나 복잡한 문제를 해결할 수 있다는 것을 확인했다. 비교적 드문 동사 형태를 사용하면 한계에 도달했다. 하지만 이런 한계를 극복할 수 있다. 해결하지 못한 케이스를 학습 데이터셋에 추가하면, 모델을 개선할 수가 있다.

또한 인류를 위한 AI가 존재한다는 점도 알 수 있었다. 앨런 인공지능 연구소는 많은 무료 인공지능 리소스를 제공한다. 또한 사용자가 AI를 이해하는 데 도움을 주기 위해 NLP 모델의 출력 시각화 기능이 존재한다. AI를 설명하는 것은 AI를 실행하는 것만큼이나 중요하다는 것을 알았다. 시각화와 텍스트 표현을 통해 BERT 기반 모델의 잠재력을 명확하게 파악할 수 있었다.

마지막으로, 다른 AI 도구와 함께 사용하는 방법을 최적화하기 위해 SRL의 범위와 한계를 살펴봤다.

트랜스포머는 분산 아키텍처와 입력 형식으로 NLP 표준화를 지속적으로 개선해 나갈 것이다.

다음 11장, 데이터가 말하게 하기: 스토리, 질문, 답변에서 일반적으로 인간만이 잘 수행할 수 있는 작업에 대해 트랜스포머를 적용해 볼 것이다. 개체명 인식(NER, Named Entity Recognition)과 질의응답(question-answering) 작업에 트랜스포머의 잠재력을 살펴볼 것이다.

10.7 문제

01. 의미역 결정(SRL)은 텍스트 생성 작업이다. 참 / 거짓
02. 서술어는 명사이다. 참 / 거짓
03. 동사는 서술어이다. 참 / 거짓
04. 논항은 누가, 무엇을 하고 있는지를 설명할 수 있다. 참 / 거짓
05. 수식어는 부사가 될 수 있다. 참 / 거짓
06. 수식어는 위치가 될 수 있다. 참 / 거짓
07. BERT 기반 모델은 인코더 및 디코더 스택으로 이루어져있다. 참 / 거짓
08. BERT 기반 SRL 모델에는 표준 입력 형식이 있다. 참 / 거짓
09. 트랜스포머는 모든 SRL 작업을 해결할 수 있다. 참 / 거짓

10.8 참고 문헌

- Peng Shi and Jimmy Lin, 2019, Simple BERT Models for Relation Extraction and Semantic Role Labeling: <https://arxiv.org/abs/1904.05255>
- 앨런 인공지능 연구소: <https://allennlp.org/>

11장

데이터가 말하게 하기: 스토리, 질문, 답변

- 11.1 방법론
- 11.2 방법 0: 시행착오
- 11.3 방법 1: NER
- 11.4 방법 2: SRL
- 11.5 다음 단계
- 11.6 정리하기
- 11.7 문제
- 11.8 참고 문헌

독해력에는 많은 기술이 필요하다. 우리는 텍스트를 읽을 때 키워드와 주요 사건을 알아차리고 내용을 이해한다. 그런 다음 콘텐츠에 대한 지식과 표현을 사용하여 질문에 답한다. 또한 함정과 실수를 피하기 위해 각 질문을 검토한다.

트랜스포머가 아무리 강력해졌다고 해도 열린 질문에는 쉽게 답할 수 없다. 열린 환경이란 누군가가 어떤 주제에 대해 어떤 질문이든 할 수 있고, 트랜스포머가 올바르게 대답할 수 있는 환경을 의미한다. 이는 쉽지 않지만 이번 장에서 살펴볼 것처럼 GPT-3로 어느 정도 가능하다. 트랜스포머는 닫힌 질의응답 환경에서 일반적인 데이터셋을 사용하는 경우가 많다. 의로나 법률 같은 예민한 질문과 답변에는 추가적인 NLP 기능이 필요하다.

트랜스포머는 학습 환경이 전처리된 질문-답변 시퀀스로 닫혀 있는지 여부에 관계없이 어떤 질문에도 올바르게 답할 수 없다. 한 시퀀스에 두 개 이상의 주제와 복합 명제가 포함된 경우 트랜스포머 모델이 잘못된 예측을 할 수 있다.

이번 장에서는 다른 NLP 작업을 사용하여 텍스트에서 모호하지 않은 콘텐츠를 찾는 질문 생성기를 구축하는 방법에 중점을 둔다. 질문 생성기는 질문-답변을 구현하는 데 적용된 몇 가지 아이디어를 보여준다.

먼저 무작위로 질문을 던졌을 때 트랜스포머가 매번 잘 답변하는 것이 얼마나 어려운지 살펴본다.

다음으로 **NER(Named Entity Recognition, 개체명 인식)**로 의미 있는 질문을 생성하여 DistilBERT 모델이 질문에 답할 수 있도록 돕는다. 또한 트랜스포머를 위한 질문 생성기의 기반을 마련한다.

판별자(discriminator)로 사전 학습된 ELECTRA 모델을 질문-답변 토크스에 추가해 본다.

계속해서 텍스트 생성기의 청사진에 **SRL(Semantic Role Labeling, 의미역 결정)** 기능을 추가한다.

그리고 다음 단계 섹션에서는 헤이스택(Haystack) 프레임워크 구현을 포함하여 신뢰할 수 있는 질문-답변 솔루션을 구축하기 위한 아이디어를 살펴본다.

마지막으로, 열린 환경에서 질문-답변 작업을 살펴보기 위해 온라인 GPT-3 다빈치(Davinci) 엔진 인터페이스를 사용해 본다. 다시 말하지만, 개발, 교육, 준비 과정이 필요하지 않다!

이번 장이 끝나면 자신만의 멀티태스크(multi-task) NLP를 구축하거나, 질문-답변을 위해 Cloud AI 사용하는 방법을 알게 될 것이다.

이 장에서는 다음 주제들을 다룬다.

- 무작위 질문-답변의 한계
- NER로 의미 있는 질문 생성하기
- 트랜스포머용 질문 생성기 청사진 설계하기
- NER로 생성한 질문 테스트하기
- 판별자로 사전 학습된 ELECTRA 인코더 살펴보기
- 표준 질문으로 ELECTRA 모델 테스트하기
- 술어(predicate) 식별을 기반으로 SRL로 의미 있는 질문 생성하기
- 질문-답변 트랜스포머를 구현하기 위한 프로젝트 관리 지침 살펴보기
- SRL로 질문을 생성하는 방법 분석하기
- NER과 SRL을 이용한 질문 생성기의 청사진 정의하기
- RoBERTa와 헤이스택의 질문-답변 프레임워크 살펴보기
- 개발이나 준비가 필요 없는 GPT-3 인터페이스 사용하기

먼저 질문-답변 작업을 위해 질문을 생성하는데 적용할 방법론을 살펴보자.

질문-답변은 주로 트랜스포머와 바로 사용할 수 있는 질문-답변 데이터셋이 포함된 NLP 예제가 주어지고, 해당 질문에 답하는 방식으로 진행된다. 트랜스포머는 이 닫힌 환경에서 질문에 답하도록 학습된다.

그러나 더 복잡한 상황에서는 신뢰할 수 있는 트랜스포머 모델 구현을 위해 커스텀하는 방법이 필요하다.

11.1.1 트랜스포머와 방법

질문-답변이나 다른 NLP 작업을 위한 완벽하고 효율적인 범용 트랜스포머 모델은 존재하지 않는다. 프로젝트에 가장 적합한 모델은 해당 데이터셋과 작업에 가장 적합한 결과물을 생성하는 모델이다.

많은 경우 방법은 모델보다 성능이 뛰어나다. 예를 들어, 평균 모델을 사용하는 적절한 방법이 우수한 모델을 사용하는 결함이 있는 방법보다 더 효율적인 결과를 생성하는 경우가 많다.

이 장에서는 DistilBERT, ELECTRA 및 RoBERTa 모델을 실행해 본다. 어떤 모델은 다른 모델보다 더 나은 성능을 제공한다.

그러나 성능이 결과를 보장하지는 않는다. 예를 들어, 우주 로켓 및 우주선 제작 프로젝트에서 NLP 봇에게 질문을 하면 정확한 답변을 얻을 수 있다.

사용자가 로켓의 재생 냉각 노즐과 연소실 상태에 대한 수백 페이지 분량의 보고서에 대해 질문해야 한다고 가정해 보자. 질문은 냉각 상태가 신뢰할 수 있는지 여부와 같이 구체적일 수 있다. 이것이 바로 사용자가 NLP 봇에게 원하는 핵심 정보다.

요약하자면, NLP 봇이 통제 없이 그대로 통계적으로 답변하는 것은 너무 위험하기 때문에 실제로 사용할 수 없다. 신뢰할 수 있는 NLP 봇은 데이터와 규칙에 따라 규칙 기반 시스템을 실행하여 NLP 봇의 답변을 확인한다. NLP 트랜스포머 모델 봇은 사람의 목소리로 부드럽고 신뢰할 수 있는 자연어 답변을 생성한다.

모든 요구 사항에 맞는 범용 트랜스포머 모델과 방법은 존재하지 않는다. 각 프로젝트마다 특정 기능과 맞춤형 접근 방식이 필요하며 사용자의 기대에 따라 크게 달라질 수 있다.

이 장에서는 특정 트랜스포머 모델 선택을 넘어 질문-답변의 일반적인 제약 조건에 초점을 맞춘다. 이 장은 질문-답변 프로젝트 가이드가 아니라 질문-답변에 트랜스포머를 어떻게 사용할 수 있는지 다룬다.

먼저 대비되지 않은 열린 환경에서 질문-답변을 하는 데 중점을 둔다. 트랜스포머 모델을 사용하려면 다른 NLP 작업과 기존 프로그램의 도움이 필요하다. 프로젝트의 목표를 달성하기 위한 몇 가지 방법을 살펴보자.

- 방법 0 무작위로 질문을 던지는 시행착오 방식을 살펴본다.
- 방법 1 질문-답변 작업을 준비하는 데 도움이 되는 NER을 도입한다.
- 방법 2 기본 트랜스포머에 ELECTRA 트랜스포머 모델을 도입한다. 또한 트랜스포머가 질문을 준비하는 데 도움이 되는 SRL을 도입한다.

이 세 가지 방법은 단일 질문-답변 방법으로는 유명 기업 프로젝트에 충분하지 않다는 것을 보여준다. NER과 SRL을 추가하면 트랜스포머 솔루션의 언어 지능이 향상된다.

예를 들어, 항공 우주 기업은 전술적 방위 프로젝트에 대한 답변 신뢰도를 100% 보장하는 질문-답변을 구현하기 위해 다양한 NLP 방법을 결합했다.

이처럼 프로젝트 구현을 위해 다중 방법(multi-method) 솔루션을 설계할 수 있다.

시행착오 접근 방식부터 시작하자.

11.2 방법 0: 시행착오

질문-답변은 매우 쉬워 보인다. 정말 그런지 확인해 보자. 구글 코랩(Google Colab)에서 QA.ipynb를 열고 셀(cell) 단위로 실행해 보자.

첫 번째 셀을 실행하여 트랜스포머를 설치한다.

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

NOTE

참고: 허깅페이스 트랜스포머는 시장에 맞게 라이브러리와 모듈을 업데이트하며 지속적으로 발전하고 있다. 기본 버전이 작동하지 않는 경우, !pip install transformers=[노트북 셀이 정상 실행되는 버전]으로 고정해야 할 수도 있다.

이제 허깅페이스의 파이프라인을 가져온다. 여기에는 바로 사용할 수 있는 많은 트랜스포머 모델들이 포함되어 있다. 이 리소스는 다양한 작업을 수행할 수 있도록 높은 수준의 추상화 기능을 제공한다. 간단한 API를 통해 이러한 NLP 작업을 사용할 수 있다. 이 프로그램은 구글 코랩에서 만들어졌다. 무료 이메일 계정을 사용하여 구글 코랩 VM에서 실행하는 것이 좋다.

이제 pipeline을 불러온다.

```
from transformers import pipeline
```

이 작업이 완료되면 트랜스포머 모델과 작업을 인스턴스화 할 수 있는 옵션이 생긴다.

1. 기본 모델과 기본 토큰라이저를 사용하여 NLP 작업을 수행한다.

```
pipeline("<task_name>")
```

2. 사용자 정의 모델을 사용하여 NLP 작업을 수행한다.

```
pipeline("<task_name>", model="<model_name>")
```

3. 사용자 정의 모델과 사용자 정의 토큰나이저를 사용하여 NLP 작업을 수행한다.

```
pipeline("<task_name>", model="<model_name>", tokenizer="<tokenizer_name>")
```

기본 모델과 토큰나이저부터 살펴보자.

```
nlp_qa = pipeline('question-answering')
```

이제 트랜스포머에 질문할 텍스트를 작성한다.

```
sequence = "The traffic began to slow down on Pioneer Boulevard in Los Angeles, making it difficult to get out of the city. However, WBGO was playing some cool jazz, and the weather was cool, making it rather pleasant to be making it out of the city on this Friday afternoon. Nat King Cole was singing as Jo, and Maria slowly made their way out of LA and drove toward Barstow. They planned to get to Las Vegas early enough in the evening to have a nice dinner and go see a show."
```

(도시 밖으로 벗어나기 어려울 정도로 로스앤젤레스의 파이오니어 대로의 교통체증이 시작되고 느려졌다. 하지만 WBGO에서 멋진 재즈가 나오고 날씨도 선선했기 때문에 금요일 오후에 도시를 벗어나며 즐거울 수 있었다. 넷 킹 콜(Nat King Cole)이 노래를 부르고 있었고 조(Jo)와 마리아(Maria)는 천천히 LA를 벗어나 바스토로 향했다. 라스베가스에 일찍 도착해 멋진 저녁을 먹고 쇼를 보러 갈 계획이었다.)

시퀀스는 놀라울 정도로 간단하며, API에 질문하고 답변을 얻기만 하면 된다.

```
nlp_qa(context=sequence, question='Where is Pioneer Boulevard?(파이오니어 대로는 어디에 있는가?)')
```

결과는 완벽하다.

```
{'answer': 'Los Angeles.', 'end': 66, 'score': 0.988201259751591, 'start': 55}
```

방금 몇 줄의 코드로 질문-답변 트랜스포머 NLP 작업을 구현했다!

이제 텍스트, 질문, 답변이 포함된 바로 사용 가능한 데이터셋을 다운로드할 수 있다.

여기까지의 내용만으로도 질문-답변 작업을 위한 준비가 모두 완료됐다. 하지만 실제 구현은 결코 간단하지 않다. 사용자가 데이터베이스에 저장된 많은 문서를 질문할 수 있도록 질문-답변 트랜스포머

모델을 구현해야 한다고 가정해 보자. 여기에는 두 가지 중요한 제약 조건이 있다.

- 먼저 일련의 주요 문서로 트랜스포머를 실행하여 시스템이 작동한다는 것을 보여줘야 한다.
- 트랜스포머가 질문에 올바르게 답한다는 것을 어떻게 보장하는지 보여줘야 한다.

몇 가지 질문이 떠오른다.

- 시스템 테스트를 위한 질문은 누가 찾을 수 있을까?
- 전문가가 작업을 수행하기로 동의하더라도 많은 질문에서 잘못된 결과가 나오면 어떻게 될까?
- 결과가 만족스럽지 않은 경우 모델을 계속 학습시킬 것인가?
- 어떤 모델을 사용해도 일부 질문에 대한 답을 얻을 수 없다면 어떻게 할까?
- 제한된 샘플에서 작동은 하지만 프로세스가 너무 오래 걸리고 비용이 너무 많이 들어 확장할 수 없다면 어떻게 해야 할까?

전문가의 도움을 받아 어떤 것이 효과적인지 확인하려면 시간이 오래 걸릴 수 있다. 시행착오는 해결책이 아니다.

이번 장에서는 질문-답변 트랜스포머 모델을 구현하는 데 드는 비용을 절감할 수 있는 몇 가지 방법과 도구를 제공하는 것을 목표로 한다. 고객을 위해 새로운 데이터셋을 구축할 때 질문-답변에 적합한 질문을 찾는 것은 상당히 어려운 일이다.

트랜스포머는 인코더 전용 또는 디코더 전용 스택을 사용하여 원하는 대로 조립할 수 있는 레고 빌딩 블록 셋이라고 생각할 수 있다. 소형, 대형 또는 초대형(XL) 트랜스포머 모델 셋을 사용할 수 있다.

또한 이 책에서 살펴본 NLP 작업을 구현하는 프로젝트의 레고 빌딩 블록 셋이라고 생각할 수도 있다. 다른 소프트웨어 구현과 마찬가지로 목표를 달성하기 위해 두 개 이상의 NLP 작업을 조합할 수 있다. 시행착오를 거치며 질문을 찾는 방식에서 체계적인 접근 방식으로 전환해 보자.

이번 장에서는

- 각 섹션에 설명된 방법을 살펴보기 위해 QA.ipynb를 셀 단위로 계속 실행한다.
- 또한 AllenNLP NER 인터페이스를 사용하여 NER과 SRL 결과의 시각적으로 표현한다. 인터페이스에 문장을 입력한 다음 NER 또는 SRL을 선택하고 시퀀스를 입력할 수 있다. 이 장에서는 AllenNLP 모델을 사용한다.

먼저 NER 방식으로 질문-답변에 적합한 XL 트랜스포머 모델 질문을 찾아보자.

11.3 방법 1: NER

이 섹션은 NER을 사용하여 좋은 질문을 찾기 위한 아이디어를 보여준다. 트랜스포머 모델은 지속적으로 학습되고 업데이트된다. 또한 학습에 사용되는 데이터셋이 변경될 수 있다. 마지막으로, 이 방식은 매번 동일한 결과를 생성하는 규칙 기반 알고리즘이 아니다. 실행할 때마다 결과가 달라질 수 있다. NER은 사람, 위치, 조직 및 기타 개체를 순서대로 감지할 수 있다. 먼저 질문하기 위해 단락의 주요 부분을 찾아주는 NER 작업을 실행해 본다.

11.3.1 NER을 사용하여 질문 찾기

QA.ipynb를 셀 단위로 실행한다. 이제 프로그램은 기본 모델과 토큰라이저를 다루기 위해 NER 작업으로 파이프라인을 초기화한다.

```
nlp_ner = pipeline("ner")
```

이 챕터의 방법 0: 시행착오 섹션에서 실행한 시퀀스를 계속 사용한다.

```
sequence = "The traffic began to slow down on Pioneer Boulevard in Los Angeles, making it difficult to get out of the city. However, WBGO was playing some cool jazz, and the weather was cool, making it rather pleasant to be making it out of the city on this Friday afternoon. Nat King Cole was singing as Jo and Maria slowly made their way out of LA and drove toward Barstow. They planned to get to Las Vegas early enough in the evening to have a nice dinner and go see a show."
```

QA.ipynb에서 nlp_ner 셀을 실행한다.

```
print(nlp_ner(sequence))
```

NLP 작업의 결과를 출력한다. 점수는 소수점 이하 두 자리까지 반올림한다.

```
[{'word': 'Pioneer', 'score': 0.97, 'entity': 'I-LOC', 'index': 8},
 {'word': 'Boulevard', 'score': 0.99, 'entity': 'I-LOC', 'index': 9},
 {'word': 'Los', 'score': 0.99, 'entity': 'I-LOC', 'index': 11},
 {'word': 'Angeles', 'score': 0.99, 'entity': 'I-LOC', 'index': 12},
 {'word': 'W', 'score': 0.99, 'entity': 'I-ORG', 'index': 26},
 {'word': '##B', 'score': 0.99, 'entity': 'I-ORG', 'index': 27},
 {'word': '##G', 'score': 0.98, 'entity': 'I-ORG', 'index': 28},
 {'word': '##O', 'score': 0.97, 'entity': 'I-ORG', 'index': 29},
 {'word': 'Nat', 'score': 0.99, 'entity': 'I-PER', 'index': 59},
 {'word': 'King', 'score': 0.99, 'entity': 'I-PER', 'index': 60},
 {'word': 'Cole', 'score': 0.99, 'entity': 'I-PER', 'index': 61},
 {'word': 'Jo', 'score': 0.99, 'entity': 'I-PER', 'index': 65},
 {'word': 'Maria', 'score': 0.99, 'entity': 'I-PER', 'index': 67},
 {'word': 'LA', 'score': 0.99, 'entity': 'I-LOC', 'index': 74},
 {'word': 'Bar', 'score': 0.99, 'entity': 'I-LOC', 'index': 78},
 {'word': '##sto', 'score': 0.85, 'entity': 'I-LOC', 'index': 79},
 {'word': '##w', 'score': 0.99, 'entity': 'I-LOC', 'index': 80},
 {'word': 'Las', 'score': 0.99, 'entity': 'I-LOC', 'index': 87},
 {'word': 'Vegas', 'score': 0.9989519715309143, 'entity': 'I-LOC', 'index': 88}]
```

허깅페이스 문서에 사용된 라벨에 대한 설명이 나와 있다. 주요 라벨은 다음과 같다.

- I-PER, 사람 이름
- I-ORG, 조직명
- I-LOC, 위치명

결과가 정확하다. Barstow는 토큰 3개로 분할되었다.

NER 섹션의 AllenNLP에서 동일한 시퀀스를 실행하면 시퀀스를 시각적으로 표현할 수 있다.³⁴

34 역주. allennlp 데모의 서비스가 종료되었다. 대신 <https://corener-demoaiola-lab.com>의 ner 섹션을 참고하자.

The traffic began to slow down on **Pioneer Boulevard** in **Los Angeles**, making it difficult to get out of the city. However, **WBGO** was playing some cool jazz, and the weather was cool, making it rather pleasant to be making it out of the city on this Friday afternoon. **Nat King Cole** was singing as **Jo** and **Maria** slowly made their way out of **LA** and drove toward **Barstow**. They planned to get to **Las Vegas** early enough in the evening to have a nice dinner and go see a show.

그림 11.1 NER

NER이 질문-답변을 위한 질문을 만드는 데 사용할 핵심 개체를 강조했다.

트랜스포머에게 두 가지 유형의 질문을 해 보자.

- 위치와 관련된 질문
- 사람과 관련된 질문

위치 관련 질문부터 시작해 보자.

11.3.1.1 위치 개체 질문

QA.ipynb는 거의 20개에 달하는 개체를 생성했다. 특히 위치 개체가 흥미롭다.

```
[{'word': 'Pioneer', 'score': 0.97, 'entity': 'I-LOC', 'index': 8},
 {'word': 'Boulevard', 'score': 0.99, 'entity': 'I-LOC', 'index': 9},
 {'word': 'Los', 'score': 0.99, 'entity': 'I-LOC', 'index': 11},
 {'word': 'Angeles', 'score': 0.99, 'entity': 'I-LOC', 'index': 12},
 {'word': 'LA', 'score': 0.99, 'entity': 'I-LOC', 'index': 74},
 {'word': 'Bar', 'score': 0.99, 'entity': 'I-LOC', 'index': 78},
 {'word': '##sto', 'score': 0.85, 'entity': 'I-LOC', 'index': 79},
 {'word': '##w', 'score': 0.99, 'entity': 'I-LOC', 'index': 80},
 {'word': 'Las', 'score': 0.99, 'entity': 'I-LOC', 'index': 87},
 {'word': 'Vegas', 'score': 0.9989519715309143, 'entity': 'I-LOC', 'index': 88}]
```

11.3.1.2 휴리스틱 적용하기

휴리스틱(heuristics) 방법을 적용하여 QA.ipynb의 출력으로 질문을 만들 수 있다.

- 파서(parser)를 사용하여 위치를 원래 형식으로 다시 합치기.
- 템플릿(template) 적용하기

코드를 작성하는 것은 이 책의 범위를 벗어나므로 다음 의사 코드처럼 작업을 수행하는 함수를 작성한다.

```
for i in range beginning of output to end of the output:
    filter records containing I-LOC
    merge the I-LOCs that fit together
    save the merged I-LOCs for questions-answering
```

NER의 출력은 다음과 같다.

- I-LOC, Pioneer Boulevard
- I-LOC, Los Angeles
- I-LOC, LA
- I-LOC, Barstow
- I-LOC, Las Vegas

두 가지 템플릿을 사용하여 질문을 자동으로 생성한다. 예를 들어 무작위 함수를 적용할 수 있다. 다음 의사 코드처럼 작업을 수행하는 함수를 작성한다.

```
from the first location to the last location:
choose randomly:
    Template 1: Where is [I-LOC]?
    Template 2: Where is [I-LOC] located?
```

질문 5개가 자동으로 표시된다.

```
Where is Pioneer Boulevard?(파이오니어 대로는 어디에 있는가?)
Where is Los Angeles located?(로스앤젤레스가 있는 곳은 어디인가?)
Where is LA?(LA는 어디에 있는가?)
Where is Barstow?(바스토는 어디에 있는가?)
Where is Las Vegas located?(라스베가스가 있는 곳은 어디인가?)
```

우리가 만들었던 시퀀스로 답변할 수 없는 질문도 있다. 하지만 이를 자동으로 관리할 수도 있다. 질문은 다음 방법으로 자동 생성되었다고 가정한다.

- 시퀀스 입력하기
- NER 실행하기
- 자동으로 질문 생성하기

자동 생성된 질문에 답을 가정하고 실행해 보자.

```
nlp_qa = pipeline('question-answering')
print("Question 1.",nlp_qa(context=sequence, question='Where is Pioneer
Boulevard?'))
print("Question 2.",nlp_qa(context=sequence, question='Where is Los Angeles
located?'))
print("Question 3.",nlp_qa(context=sequence, question='Where is LA'))
print("Question 4.",nlp_qa(context=sequence, question='Where is Barstow?'))
print("Question 5.",nlp_qa(context=sequence, question='Where is Las Vegas
located ?'))
```

Question 1만 정답으로 출력되었다.

```
Question 1. {'score': 0.9879662851935791, 'start': 55, 'end': 67, 'answer': 'Los Angeles,'}
Question 2. {'score': 0.9875189033668121, 'start': 34, 'end': 51, 'answer': 'Pioneer Boulevard'}
Question 3. {'score': 0.5090435442006118, 'start': 55, 'end': 67, 'answer': 'Los Angeles,'}
Question 4. {'score': 0.3695214621538554, 'start': 387, 'end': 396, 'answer': 'Las Vegas'}
Question 5. {'score': 0.21833994202792262, 'start': 355, 'end': 363, 'answer': 'Barstow.'}
```

출력에는 score, 정답의 start 및 end 위치, answer가 표시된다. 이번 실행에서 Question 2의 score는 0.98점이지만, Pioneer Boulevard에 Los Angeles가 있다고 잘못 답변했다.

이제 프로젝트 관리로 트랜스포머를 통제하는 기능을 추가해 보자.

11.3.1.3 프로젝트 관리

트랜스포머를 관리하는 방법과 하드코딩된 함수 예제 네 가지를 살펴본다. 예제는 쉬운, 중간, 어려운, 매우 어려운 네 가지 수준으로 분류한다. 프로젝트 관리는 이 책의 범위를 벗어나므로 네 가지 범주를 간략하게 살펴본다.

1. **쉬운 프로젝트**로는 초등학교용 웹사이트를 예로 들 수 있다. 약간의 HTML 개발로 질문에 일정한 형식의 다섯 가지 답변을 제시하고 아래에 (True, False)를 추가한다. (예: Barstow는 캘리포니아에 있다) 선생님은 관리자 인터페이스로 객관식 질문의 정답을 클릭하면 된다.
2. **중간 프로젝트**는 API를 사용하여 답을 확인하고 자동으로 수정하는 프로그램에 트랜스포머의 질문과 답변을 캡슐화하는 것이다. 트랜스포머가 틀린 답변은 추가 분석을 위해 저장된다.
3. **어려운 프로젝트**는 중간 프로젝트에 세부적인 질문을 하는 챗봇을 구현하는 것이다. 예를 들어, 트랜스포머는 Pioneer Boulevard가 Los Angeles에 있다고 정확하게 알려준다. 챗봇 사용자는 near where in LA?(LA 어디 근처인가요?)와 같은 추가 질문을 할 수 있다. 이 부분은 더 많은 개발이 필요하다.
4. **매우 어려운 프로젝트**는 데이터셋의 수백만 개 레코드에서 I-LOC 개체를 인식하고 지도(map) API로 실시간 스트리밍 결과를 출력하도록 트랜스포머를 학습하는 연구 프로젝트이다.

좋은 소식은 우리가 구현한 것을 사용할 방법 있다는 것이고, 나쁜 소식은 실제 프로젝트에서 트

랜스포머나 다른 AI를 구현하려면 강력한 장비와 프로젝트 관리자, 주제별 전문가(Subject Matter Experts, SMEs), 개발자, 최종 사용자 간의 엄청난 팀워크가 필요하다는 것이다.

이제 사람 관련 질문을 해보자.

11.3.1.4 사람 개체 질문

쉬운 질문부터 시작해 보자.

```
nlp_qa = pipeline('question-answering')
nlp_qa(context=sequence, question='Who was singing?(노래하고 있던 사람은 누구인가?)')
```

출력된 정답이 맞다. 시퀀스에 누가 노래를 불렀는지 명시되어 있다.

```
{'answer': 'Nat King Cole,'
 'end': 277,
 'score': 0.9653632081862433,
 'start': 264}
```

이제 트랜스포머에게 명확하게 명시되지 않았기 때문에 약간의 생각이 필요한 질문을 해보자.

```
nlp_qa(context=sequence, question='Who was going to Las Vegas?(라스베가스에 가고 있던 사람은 누구인가?)')
```

문장을 분리하지 않고는 질문에 대답할 수 없는데, 트랜스포머는 이번에도 잘못 답변했다.

```
{'answer': 'Nat King Cole,'
 'end': 277,
 'score': 0.3568152742800521,
 'start': 264}
```

트랜스포머의 답변은 0.35점밖에 받지 못했다. 이 score는 계산마다 또는 트랜스포머 모델마다 다를 수 있다. 트랜스포머가 의미적 결정 문제에 직면했음을 알 수 있다. 사람 개체 질문에는 SRL 방식을 적용하여 더 나은 결과를 얻어 보자.

11.4 방법 2: SRL

트랜스포머는 Las Vegas로 가기 위해 운전하는 사람을 찾을 수 없었고 Jo와 Maria가 아닌 Nat King Cole이 운전하는 것으로 생각했다.

무엇이 잘못되었는지 그리고 트랜스포머가 생각한 내용에 대해 알기 위해 의미역 모델링을 살펴보자. 필요한 경우 잠시 시간을 내어 10장, BERT 기반 트랜스포머를 사용한 SRL을 다시 살펴본다.

SRL 섹션의 AllenNLP에서 동일한 시퀀스로 이전 장에서 사용한 SRL BERT 모델을 실행하여 동사 drove의 시각적 표현을 얻었다.³⁵

drove의 **Frames**:

The traffic began to slow down on Pioneer Boulevard in Los Angeles .

making it difficult to get out of the city . However , WBGO was playing

some cool jazz , and the weather was cool , making it rather pleasant to

be making it out of the city on this Friday afternoon . Nat King Cole was

singing as

Jo and Maria
논항

slowly
수식어-태도

 made their way out of LA and

drove
서술어

toward Barstow
수식어-방향

 . They planned to get to Las Vegas early

그림 11.2 SRL 실행 결과

SRL BERT는 프레임 19개를 발견했다. 이 섹션에서는 drove에 중점을 둔다.

NOTE

결과는 실행할 때마다 또는 AllenNLP가 모델 버전을 업데이트할 때마다 달라질 수 있다.

³⁵ 역주. Allennlp 데모의 서비스가 종료되었다. 실험을 원한다면 <https://hanlp.hankcs.com/en/demos/srl.html>에서 실행하자.

동사 drove의 주어를 Jo and Maria로 잘못 파악했다.

NOTE

트랜스포머 모델은 계속 진화하고 있다는 점을 명심한다. 출력은 달라질 수 있지만 개념은 동일하다.

사실인지 QA.ipynb에서 질문해 보자.

```
nlp_qa(context=sequence, question='Who are they?(그들은 누구인가?)')
```

출력이 예상했던 대로 나온다.

```
{'answer': 'Jo and Maria',  
  'end': 305,  
  'score': 0.8486017557290779,  
  'start': 293}
```

정답을 얻기 위해 질문하는 방법을 찾을 수 있을까? 질문을 의역하여 시도해 보자.

```
nlp_qa(context=sequence, question='Who drove to Las Vegas?(라스베가스에 운전해 갔던 사람은 누구인가?)')
```

다소 더 나은 결과를 얻을 수 있다.

```
{'answer': 'Nat King Cole was singing as Jo and Maria',  
  'end': 305,  
  'score': 0.35940926070820467,  
  'start': 264}
```

이제 트랜스포머는 Nat King Cole이 노래를 부르고 있었고 Jo와 Maria가 그 사이에 무언가를 하고 있었다는 것을 이해한다.

이제 더 나은 질문을 할 수 있는 방법을 찾아야 한다.

다른 모델을 시도해 보자.

11.4.1 ELECTRA로 질문-답변하기

다른 모델을 시도하기 전에 먼저 어떤 모델을 사용 중인지 알아야 한다.

```
print(nlp_qamodel)
```

출력은 질문-답변에 학습된 DistilBERT 모델을 사용했음을 보여준다.

```
DistilBertForQuestionAnswering((distilbert): DistilBertModel(
```

모델에는 출력에서 총 6에 표시된 것처럼 총 6개와 피쳐 768개가 있다(각 층은 0부터 n까지 번호가 매겨져 있다).

```
(5): TransformerBlock(  
  (attention): MultiHeadSelfAttention(  
    (dropout): Dropout(p=0.1, inplace=False)  
    (q_lin): Linear(in_features=768, out_features=768, bias=True)  
    (k_lin): Linear(in_features=768, out_features=768, bias=True)  
    (v_lin): Linear(in_features=768, out_features=768, bias=True)  
    (out_lin): Linear(in_features=768, out_features=768, bias=True)
```

이제 ELECTRA 트랜스포머 모델을 사용해 보자. 2020년 클라크(Clark et al.)는 **Masked Language Modeling(MLM, 마스크드 언어 모델링)**의 사전 학습 방법을 개선한 트랜스포머 모델을 설계했다.

3장, BERT 모델 미세 조정하기의 마스크된 언어 모델링 섹션에서 BERT 모델이 학습 과정에서 무작위로 [MASK] 토큰을 삽입하는 것을 보았다.

클라크는 단순히 무작위 토큰 대신 생성자(generator)를 사용하는 방법을 소개했다. BERT 모델은 (마스크된) 손상된 토큰을 예측하도록 학습된다. 클라크는 마스크된 토큰이 생성된 토큰인지 아닌지를 예측하는 판별자(discriminator)로 ELECTRA 모델을 학습시켰다. 그림 11.3은 ELECTRA가 어떻게 학습되는지 보여준다.

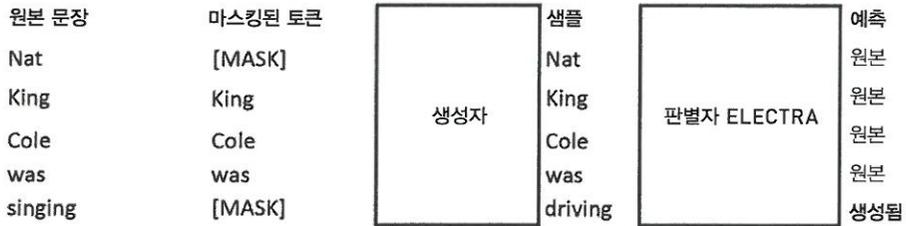


그림 11.3 ELECTRA는 판별자로 학습된다

그림 11.3은 원본 시퀀스가 생성자를 통과하기 전에 마스킹되는 것을 보여준다. 생성자는 무작위 토큰이 아닌 수용 가능한(acceptable) 토큰을 삽입한다. ELECTRA 트랜스포머 모델은 토큰이 원본 시퀀스에서 나왔는지 혹은 생성되었는지 예측하도록 학습된다.

ELECTRA 트랜스포머 모델의 아키텍처와 대부분의 하이퍼파라미터는 BERT 트랜스포머 모델과 동일하다.

이제 더 나은 결과를 얻을 수 있는지 알아보자. QA.ipynb에서 ELECTRA-small-generator가 있는 질문-답변 셀을 실행한다.

```
nlp_qa = pipeline('question-answering', model='google/electra-smallgenerator', tokenizer='google/electra-small-generator')
nlp_qa(context=sequence, question='Who drove to Las Vegas?')
```

결과물이 우리가 기대했던 것과 다르다.

```
{'answer': 'to slow down on Pioneer Boulevard in Los Angeles, making it difficult to',
 'end': 90,
 'score': 2.5295573154019736e-05,
 'start': 18}
```

출력은 트랜스포머 모델에 따라 변경될 수 있지만 아이디어는 동일하다.

학습 메시지도 출력된다.

- This IS expected if you are initializing ElectraForQuestionAnswering from the checkpoint of a model trained on another task or with another architecture..
- This IS NOT expected if you are initializing ElectraForQuestionAnswering from the checkpoint of a model that you expect to be exactly identical..

이러한 경고 메시지가 마음에 들지 않을 수도 있고 나쁜 모델이라고 결론 내릴 수도 있다. 하지만 가능한 모든 방법을 살펴봐야 한다. 물론 ELECTRA는 더 많은 학습이 필요할 수도 있다. 중요한 것은 새로운 아이디어를 찾기 위해 가능한 많이 실험해 봐야 모델을 더 학습시키거나 다른 모델로 넘어갈지 결정할 수 있다는 것이다.

이제 다음 단계로 넘어가보자.

11.4.2 프로젝트 관리 제약사항

기본 DistilBERT 모델과 ELECTRA 트랜스포머로 기대했던 결과를 얻지 못했다.

이제 세 가지 대안이 있다.

- 추가 데이터셋을 사용하여 DistilBERT 및 ELECTRA 또는 다른 모델을 학습한다. 실제 프로젝트에서 데이터셋 학습에는 비용이 많이 든다. 새로운 데이터셋을 구축하고 하이퍼파라미터를 변경해야 하는 경우 몇 달 동안 학습해야 될 수도 있다. 하드웨어 비용도 고려해야 한다. 또한 결과가 만족스럽지 않으면 프로젝트 관리자가 프로젝트를 중단할 수도 있다.
- 요구 사항에 맞지 않을 수도 있지만 허깅페이스 모델처럼 바로 사용할 수 있는 트랜스포머를 사용해 볼 수도 있다. (<https://huggingface.co/transformers/usage.html#extractive-question-answer>)
- 질문-답변 모델을 돕기 위해 추가 NLP 작업을 사용하여 더 나은 결과를 얻을 수 있는 방법을 찾아 본다.

이 장에서는 기본 DistilBERT 모델에 도움이 되는 추가 NLP 작업을 찾는 데 중점을 둔다.

SRL을 사용하여 술어(predicate)와 논항(argument)을 추출해 보자.

11.4.3 SRL을 사용하여 질문 찾기

AllenNLP는 10장, BERT 기반 트랜스포머를 사용한 SRL에서 SRL.ipynb에 구현한 BERT 기반 모델을 사용한다.

SRL 섹션의 AllenNLP에서 시퀀스를 다시 실행하여 시퀀스의 술어를 시각적으로 표현해 보자.³⁶

작업 중인 시퀀스를 입력한다.

```
The traffic began to slow down on Pioneer Boulevard in Los Angeles, making it difficult to get out of the city. However, WBGO was playing some cool jazz, and the weather was cool, making it rather pleasant to be making it out of the city on this Friday afternoon. Nat King Cole was singing as Jo and Maria slowly made their way out of LA and drove toward Barstow. They planned to get to Las Vegas early enough in the evening to have a nice dinner and go see a show.
```

BERT 기반 모델이 몇 가지 술어를 발견했다. 우리의 목표는 문장의 동사를 기반으로 질문을 자동으로 생성할 수 있는 SRL 결괏값을 찾는 것이다.

먼저 BERT 모델에서 생성된 술어 후보를 나열한다.

```
verbs={"began," "slow," "making"(1), "playing," "making"(2), "making"(3), "singing",..., "made," "drove," "planned," go," see"}
```

프로그램을 작성해야 한다면 아래 의사 코드처럼 동사 카운터(counter)를 적용할 수 있다.

```
def maxcount:
    for in range first verb to last verb:
        for each verb
            counter +=1
            if counter>max_count, filter verb
```

36 역주. allennlp 데모의 서비스가 종료되었다. 대신 <https://hanlp.hankcs.com/en/demos/srl.html>에서 같은 테스트를 실행해 볼 수 있다.

카운터가 허용 가능한 발생 횟수 (max_count)를 초과하면 해당 동사는 제외된다. 동사의 논항을 명확히 구분하기 위해 작업이 더 필요하다.

make의 과거형인 made도 목록에서 제외해 보자.

이제 목록은 다음과 같다.

```
verbs={"began," "slow," "playing," "singing," "drove," "planned," "go," "see"}
```

동사를 필터링하는 함수에서 논항의 길이가 긴 동사를 찾을 수 있다. 동사 began에는 매우 긴 논항이 있다.

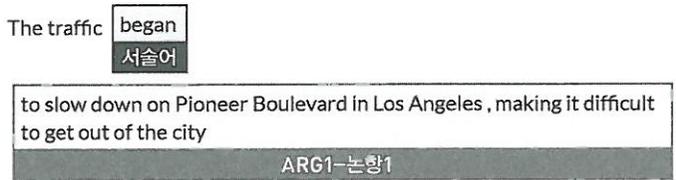


그림 11.4 동사 "began"에 적용된 SRL

"began"의 논항이 너무 길어서 스크린샷에 맞지 않는다. 텍스트 버전은 "began"의 논항을 해석하는 것이 얼마나 어려운지 보여준다.

```
began: The traffic [V: began] [ARG1: to slow down on Pioneer Boulevard in Los Angeles, making it difficult to get out of the city]. However, WBGO was playing some cool jazz], and the weather was cool, making it rather pleasant to be making it out of the city on this Friday afternoon. Nat King Cole was singing as Jo and Maria slowly made their way out of LA and drove toward Barstow. They planned to get to Las Vegas early enough in the evening to have a nice dinner and go see a show.
```

최대 길이를 초과하는 논항이 포함된 동사를 필터링하는 함수를 추가한다.

```
def maxlength:  
    for in range first verb to last verb:  
        for each verb  
            if length(argument of verb)>max_length, filter verb
```

논항 길이가 최대 길이(max_length)를 초과하는 경우, 해당 동사는 제외된다. 지금은 목록에서 began 만 제외한다.

이제 목록은 다음과 같다.

```
verbs={ "slow", "playing", "singing", "drove", "planned", "go", "see" }
```

작업 중인 프로젝트에 따라 더 많은 규칙을 추가할 수 있다. 또한 매우 제한적인 max_length 값으로 자동 질문 생성자에 사용할 흥미로운 후보를 추출할 수도 있다. 가장 짧은 논항을 가진 동사 후보를 질문으로 변환할 수 있다. 동사 slow는 시퀀스에서 한 번만 등장하고, 논항이 너무 길지 않으며, 시퀀스에서 가장 짧은 논항을 포함하고 있다는 세 가지 규칙에 부합한다.

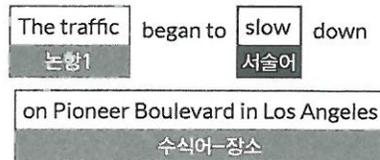


그림 11.5 동사 "slow"에 적용된 SRL

텍스트 출력을 쉽게 파싱해 볼 수 있다.

```
slow: [ARG1: The traffic] began to [V: slow] down [ARG1: on] [ARGM-ADV: Pioneer
Boulevard] [ARGM-LOC: in Los Angeles], [ARGM-ADV: making it difficult to get out
of the city].
```

출력 결과는 계속 진화하는 트랜스포머 모델에 따라 달라질 수 있지만 아이디어는 동일하게 유지된다. 동사 slow가 식별되며 이것이 SRL 출력의 핵심이다.

what 템플릿을 자동으로 생성할 수 있다. 논항 중 어떤 것도 I-PER(사람)으로 지정되지 않았으므로 who 템플릿은 생성하지 않는다. 다음 의사 코드처럼 두 가지 경우를 처리할 함수를 작성할 수 있다.

```
def whowhat:
    if NER(ARGi)==I-PER, then:
        template=Who is [VERB]
    if NER(ARGi)!=I-PER, then:
        template=What is [VERB]
```

이 함수를 사용하려면 동사 형태와 수식어를 처리하는 데 더 많은 작업이 필요하지만 이번 실험에서는 함수를 그대로 적용하여 다음과 같은 질문을 생성해 보자.

What is slow?(무엇이 느린가?)

다음 셀처럼 기본 pipeline을 실행해 보자.

```
nlp_qa = pipeline('question-answering')
nlp_qa(context= sequence, question='What was slow?')
```

결과가 만족스럽다.

```
{'answer': 'The traffic',
 'end': 11,
 'score': 0.4652545872921081,
 'start': 0}
```

DistilBERT 모델이 교통이라고(The traffic) 정답을 맞췄다.

자동 질문 생성자는 다음을 수행할 수 있다.

- 자동으로 NER 실행하기
- 기존 코드로 결과 파싱하기
- 개체를 이용하여 질문 생성하기
- 자동으로 SRL 실행하기
- 규칙 기반으로 결과 필터링하기
- NER로 사용할 템플릿을 정하여 SRL 질문 생성하기

이 솔루션은 결코 완전하지 않다. 더 많은 작업이 필요하며 추가 NLP 작업과 코드가 필요할 수도 있다. 하지만 AI를 구현하는 데 얼마나 많은 노력이 필요한지 알 수 있다.

다음으로 동사 `playing`으로 필터링해 보자. 다음 출력에서 논항이 `WBGO`와 `some cool jazz`임을 알 수 있다.

`playing`의 **Frames**:

The traffic began to slow down on Pioneer Boulevard in Los

Angeles , making it difficult to get out of the city . However ,

`WBGO` was `playing` `some cool jazz` , and the weather

그림 11.6 동사 `playing`에 적용된 SRL

텍스트 버전을 쉽게 파싱할 수 있다.

```
playing: The traffic began to slow down on Pioneer Boulevard in Los Angeles ,
making it difficult to get out of the city . [ARGM-DIS: However] , [ARGO: WBGO] was
[V: playing] [ARG1: some cool jazz]
```

NOTE

이 결과와 다음 출력은 계속 진화하는 트랜스포머 모델에 따라 달라질 수 있지만 동사와 그 논항을 식별한다는 아이디어는 동일하게 유지된다.

`whowhat` 함수를 실행하면 논항에 `I-PER`이 없다는 것을 알 수 있다. `what` 템플릿이 선택되고 다음 질문이 자동으로 생성된다.

```
What is playing?(무엇이 연주되고 있는가?)
```

다음 셀에서 기본 파이프라인을 실행해 보자.

```
nlp_qa = pipeline('question-answering')
nlp_qa(context=sequence, question='What was playing')
```

출력도 만족스럽다.

```
{'answer': 'cool jazz,.'  
  'end': 153,  
  'score': 0.35047012837950753,  
  'start': 143}
```

singing은 좋은 후보이며, whowhat 함수는 I-PER 템플릿을 찾아 다음 질문을 자동으로 생성한다.

Who is singing?(누가 노래를 부르는가?)

우리는 이번 장에서 이미 해당 질문을 성공적으로 테스트했다.

다음 동사는 트랜스포머가 해결할 수 없었던 drove다.

동사 go는 좋은 후보이다.

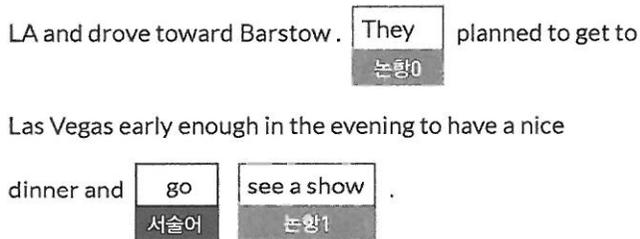


그림 11.7 동사 "go"에 적용된 SRL

올바른 동사 형태를 가진 템플릿을 생성하려면 추가 개발이 필요하다. 작업이 완료되었다고 가정하고 모델에게 다음과 같이 '누가 쇼를 보는가?'라고 질문해 보자.

```
nlp_qa = pipeline('question-answering')  
nlp_qa(context=sequence, question='Who sees a show?')
```

출력에 잘못된 답변을 받게된다.

```
{'answer': 'Nat King Cole,'  
  'end': 277,  
  'score': 0.5587267250683112,  
  'start': 264}
```

이것으로 복잡한 시퀀스에서 Nat King Cole과 Jo와 Maria가 같은 시퀀스에 존재하면 트랜스포머 모델과 모든 NLP 모델에 모호성 문제가 발생한다는 것을 알 수 있다. 더 많은 프로젝트 관리와 연구가 필요한 때이다.

11.5

다음 단계

질문-답변이나 지름길(shortcut)을 구현하는 쉬운 방법은 없다. 그래서 우리는 질문을 자동으로 생성할 수 있는 방법을 구현하기 시작했다. 자동 질문 생성은 자연어 처리의 핵심 요소이다.

더 많은 트랜스포머 모델을 NER, SRL 및 질문-답변 문제가 포함된 멀티태스크 데이터셋으로 사전 학습시켜야 한다. 또한 프로젝트 관리자는 질문-답변과 같은 특정 작업을 해결하는 데 도움이 되도록 여러 NLP 작업을 결합하는 방법을 배워야 한다.

상호 참조 해결(coreference resolution)은 모델이 작업한 시퀀스에서 주요 항목을 식별하는 데 도움이 될 수 있다. AllenNLP³⁷로 생성된 결과가 흥미롭다.

The traffic began to slow down on Pioneer Boulevard in 0 Los Angeles , making it difficult to get out of 0 the city . However , WBGO was playing some cool jazz , and the weather was cool , making it rather pleasant to be making it out of 0 the city on this Friday afternoon . Nat King Cole was singing as 1 Jo and Maria slowly made 1 their way out of 0 LA and drove toward Barstow . 1 They planned to get to Las Vegas early enough in the evening to have a nice dinner and go see a show .

그림 11.8 시퀀스의 상호 참조 해결

상호 참조 해결을 추가하여 프로그램을 계속 발전시킬 수 있다.

```
Set0={'Los Angeles', 'the city', 'LA'}
Set1=[Jo and Maria, their, they]
```

37 역주. allennlp 데모의 서비스가 종료되었다. 대신 <https://corener-demoaiola-lab.com>의 Coreference Resolution 항목을 참조하자.

상호 참조 해결을 사전 학습 작업으로 추가하거나 질문 생성자의 사후 처리(post-processing) 작업으로 추가할 수 있다. 어떤 경우든 사람의 행동을 시뮬레이션하면 질문-답변 작업의 성능을 크게 향상시킬 수 있다. 질문-답변 모델의 사전 학습 프로세스에 더 많은 맞춤형 추가 NLP 작업을 포함시켜 볼 것이다.

물론 이번 장에서 사용한 모델에 DistilBERT 및 ELECTRA와 같은 새로운 전략을 사용하여 사전 학습한 후 사용자가 원하는 질문을 하도록 할 수도 있다. 두 가지 접근 방식을 모두 권장한다.

- 질문-답변 작업을 위한 질문 생성자에 작업한다. 이러한 질문은 교육 목적으로 사용하거나, 트랜스포머를 학습하거나, 실시간으로 사용자에게 아이디어를 제공할 수 있다.
- 특정 NLP 작업을 포함하여 트랜스포머 모델을 사전 학습시켜 질문-답변 성능을 입증할 수 있다. 질문 생성기를 사용하여 추가로 학습한다.

11.5.1 RoBERTa 모델로 헤이스택 살펴보기

헤이스택은 흥미로운 기능을 갖춘 질문-답변 프레임워크이다. 주어진 프로젝트의 요구 사항에 맞는지 살펴볼 가치가 있다.

이 섹션에서는 앞서 다양한 모델과 방법을 사용하여 실험한 문장으로 질문-답변을 실행해 본다.

Haystack_QA_Pipeline.ipynb를 연다.

첫 번째 셀은 헤이스택을 실행하는 데 필요한 모듈을 설치한다.

```
# 헤이스택을 설치한다.  
!pip install farm-haystack==0.6.0  
# 코랩 호환성을 위해 특정 버전의 urllib와 토치를 설치한다.  
preinstalled versions on Colab  
!pip install urllib3==1.25.4  
!pip install torch==1.6.0+cu101-f https://download.pytorch.org/whl/torch_stable.html
```

이 노트북은 RoBERTa 모델을 사용한다.

```
# 로컬 또는 허깅페이스 허브의 QA 모델을 불러온다 (https://huggingface.co/models)
from haystack.reader.farm import FARMReader
reader = FARMReader(model_name_or_path="deepset/roberta-base-squad2", use_
gpu=True, no_ans_boost=0, return_no_answer=False)
```

4장, RoBERTa 모델 처음부터 사전 학습하기로 돌아가서 RoBERTa 모델에 대한 일반적인 설명을 참고할 수 있다.

노트북의 나머지 셀에서는 이번 챕터에서 자세히 살펴본 질문에 대한 답변을 얻을 수 있다.

```
text = "The traffic began to slow down on Pioneer Boulevard in... have a nice
dinner and go see a show."
```

얻은 답변을 이전 섹션의 결과와 비교하여 어떤 트랜스포머 모델을 구현할지 결정할 수 있다.

11.5.2 GPT-3 엔진으로 Q&A 살펴보기

이번 섹션에서는 학습, 미세 조정, 서버에 프로그램 불러오기, 데이터셋 사용을 피하는 대신 OpenAI 계정에 연결하여 대화형 교육 인터페이스를 사용한다.

GPT-3 엔진의 온라인 교육 인터페이스는 다음처럼 E(설명)와 T(텍스트)로 충분히 좋은 답변을 제공한다.

E = Answer question from this text(다음 질문에 답변하라.)

T = The traffic began to slow down on Pioneer Boulevard in(중략) have a nice dinner and go see a show.(도시밖으로 벗어나기 어려울 정도로 로스앤젤레스의 파이오니어 대로의 (중략)멋진 저녁을 먹고 쇼를 보러 갈 계획이었다.)

다음은 질문-답변 형식의 예시이다.

- Who is going to Las Vegas?: Jo and Maria(누가 라스베가스로 가고 있는가?: 조(Jo)와 마리아(Maria))
- Who was singing?: Nat King Cole(누가 노래하고 있었는가?: 넷 킹 콜(Nat King Cole))
- What kind of music was playing?: Jazz(어떤 종류의 음악이 재생되고 있었는가?: 재즈)
- What was the plan for the evening?: to have a nice dinner and go see a show(저녁 계획은 무엇이었는가?: 멋진 저녁을 먹고 쇼를 보러 간다.)

이것만 있으면 GPT-3 엔진이 있는 API 없이도 대화형 인터페이스를 통해 다양한 교육용 NLP 작업을 온라인으로 실행할 수 있다.

GPT-3에 기대하는 질문-답변을 변경하며 끝없는 상호 작용을 만들 수 있다. 차세대 NLP가 탄생했다. 4차 산업 시대 개발자, 컨설턴트 또는 프로젝트 관리자는 인지적 접근 방식, 언어학, 심리학 및 기타 여러 학문 분야에 걸치는 새로운 기술을 습득해야 한다. 필요한 경우 시간을 내어 **7장, GPT-3 엔진을 사용한 초인간 트랜스포머 등장**을 다시 살펴보는 것도 좋다.

지금까지 트랜스포머를 사용한 질의-답변의 중요한 측면을 살펴보았다. 이제 살펴본 내용을 요약해보자.

이번 장에서 우리는 질문-답변 작업이 생각만큼 쉽지 않다는 사실을 발견했다. 트랜스포머 모델을 구현하는 데는 몇 분밖에 걸리지 않는다. 하지만 실제로 동작하려면 몇 시간 또는 몇 달이 걸릴 수 있다!

먼저 허깅페이스 파이프라인의 기본 트랜스포머에 몇 가지 간단한 질문에 대한 답변을 얻었다. 기본 트랜스포머 DistilBERT는 간단한 질문에 꽤 잘 대답했다. 하지만 실생활에서 사용자들은 온갖 종류의 질문을 한다. 트랜스포머가 혼동하여 잘못된 출력을 생성할 수 있다.

그런 다음 무작위 질문에 대해 무작위 답변을 얻을 것인지, 아니면 더 생산적인 질문 생성기를 설계할지 결정했다.

먼저 유용한 콘텐츠를 찾기 위해 NER을 사용했다. NER의 출력으로 질문을 자동으로 생성하는 기능을 설계했다. 품질은 괜찮았지만 더 많은 작업이 필요했다.

ELECTRA 모델을 사용해 보았지만 기대했던 결과를 얻지 못했다. 트랜스포머 모델을 학습에 많은 리소스를 투입할지, 아니면 질문 생성기를 설계할지도 고민했다.

질문 생성기에 SRL을 추가하고 질문을 테스트했다. 또한 NER을 추가하여 몇 가지 의미 있는 질문을 생성했다. 또한 헤이스택 프레임워크도 도입하여 RoBERTa로 질문 답변을 처리하는 방법을 모색했다.

마지막으로 API 없이 OpenAI 교육용 대화형 인터페이스에서 직접 GPT-3 엔진을 사용하여 예제를 실행해 보았다. 클라우드 AI 플랫폼의 성능과 접근성이 점점 더 향상되고 있다.

실험을 통해 얻은 한 가지 결론은 멀티태스크 트랜스포머가 특정 작업으로 학습된 트랜스포머보다 복잡한 NLP 작업에서 성능이 더 낫다는 것이다. 트랜스포머를 구현하려면 잘 준비된 멀티태스크 학습, 고전적인 코드를 이용한 휴리스틱, 질문 생성기가 필요하다. 질문 생성기는 질문을 학습 데이터로 사용하거나 독립형 솔루션으로 사용할 수 있다.

다음 챕터인 12장, **고객 감정을 감지해 예측하기**에서는 소셜 미디어 피드백의 감성을 분석하는 방법을 살펴보자.

11.7 문제

01. 학습된 트랜스포머 모델은 어떤 질문에도 답할 수 있다. 참 / 거짓
02. 질문-답변은 더 이상 연구가 필요하지 않다. 참 / 거짓
03. NER은 의미 있는 질문을 찾을 때 유용한 정보를 제공할 수 있다. 참 / 거짓
04. SRL은 질문을 생성할 때 필요 없다. 참 / 거짓
05. 질문 생성기는 질문을 생성하는 훌륭한 방법이다. 참 / 거짓
06. 질문-답변에는 신중한 프로젝트 관리가 필요하다. 참 / 거짓
07. ELECTRA 모델은 GPT-2와 동일한 아키텍처를 가지고 있다. 참 / 거짓
08. ELECTRA 모델은 BERT와 동일한 아키텍처를 가지고 있지만 판별자로 학습했다. 참 / 거짓
09. NER은 위치를 인식하여 I-LOC로 라벨을 지정할 수 있다. 참 / 거짓
10. NER은 사람을 인식하여 I-PER로 라벨을 지정할 수 있다. 참 / 거짓

11.8 참고 문헌

- 옐런 인공지능 연구소: <https://allennlp.org/>
- 옐런 인공지능 연구소의 독해력 관련 자료: <https://demo.allennlp.org/reading-comprehension>
- Kevin Clark, Minh-Thang Luong, Quoc V. Le, Christopher D. Manning, 2020, ELECTRA: Pretraining Text Encoders as Discriminators Rather Than Generators: <https://arxiv.org/abs/2003.10555>
- 허깅페이스 파이프라인: https://huggingface.co/transformers/main_classes/pipelines.html
- 헤이스택의 깃허브 저장소: <https://github.com/deepset-ai/haystack/>

12장

고객 감정을 감지해 예측하기

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

감성 분석(sentiment analysis)은 합성성의 원리(principle of compositionality)³⁸를 따른다. 문장의 일부를 이해할 수 없을 때 문장 전체를 이해할 수 있을까? 감성 분석이 NLP 트랜스포머 모델로 가능할까? 이 장에서는 몇 가지 트랜스포머 모델을 사용해 감성 분석을 시도해 보자.

먼저 **SST(Stanford Sentiment Treebank)** 데이터셋을 사용해 보자. SST 데이터셋은 복잡하고 분석하기 쉽지 않다. The movie was great(그 영화는 훌륭했다.)와 같은 문장을 분석하는 것은 쉽다. 하지만 Although the movie was a bit too long, I really enjoyed it(영화가 조금 길긴 했지만, 정말 재미있게 봤다.)과 같이 복잡한 문장의 경우는 어떻게 될까? 이 문장은 분할할 수 있다. 분할할 수 있는 문장을 트랜스포머가 이해하려면 문장의 구조와 논리 형식을 이해할 수 있어야 한다.

그다음 복잡한 문장과 간단한 문장으로 여러 트랜스포머 모델을 테스트해 보자. 어떤 모델을 사용하든 충분한 학습을 하지 않으면 동작하지 않는다는 것을 알게 될 것이다. 트랜스포머 모델은 사람과 같은 면이 있다. 모델은 실제 인간의 기준선에 도달하기 위해 노력해야 하는 학생과 같다.

DistilBERT, RoBERTa-large, BERT-base, MiniLM-L12-H384-uncased, 그리고 BERT 기반 다국어 모델(BERT-base multilingual model)을 실행하는 것은 재밌다. 그러나 이들 중 일부는 더 많은 학습이 필요하다는 것을 알게 될 것이다.

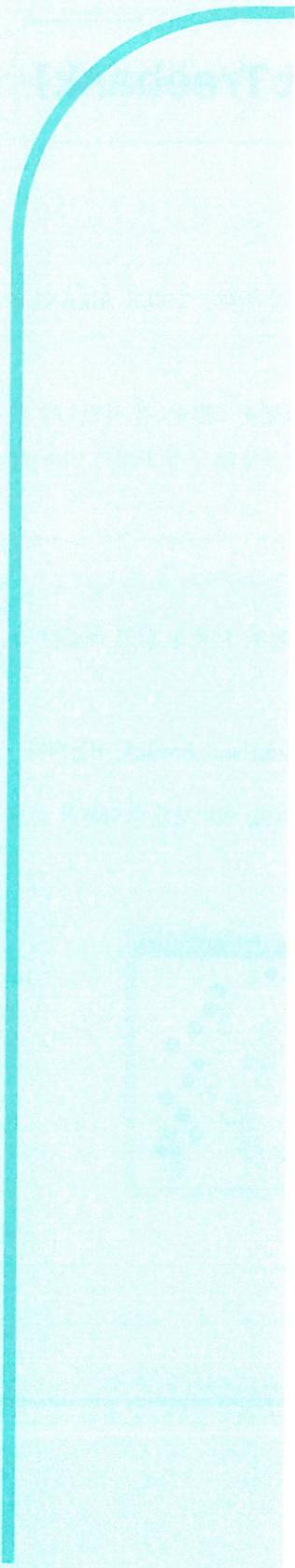
이 과정에서 감성 분석 결과를 사용해 고객 관계(customer relationships)를 개선하는 방법을 살펴보고 웹사이트에 구현할 수 있는 멋진 별 다섯 개 인터페이스를 살펴볼 것이다.

마지막으로, 감성 분석을 위해 OpenAI의 GPT-3의 온라인 인터페이스를 사용할 것이다. AI 개발이나 API가 전혀 필요하지 않다!

이번 장은 다음 주제들을 다룬다.

- 감성 분석을 위한 SST
- 긴 시퀀스의 합성성(compositionality) 정의
- AllenNLP(RoBERTa)를 사용한 감성 분석
- 복잡한 문장을 실행해 트랜스포머의 새로운 영역 살펴보기
- 허깅페이스(Hugging Face) 감성 분석 모델 사용하기
- 감성 분석을 위한 DistilBERT

38 역주. NLP 모델이 복잡한 문장의 구성 표현과 이를 결합하는 규칙을 검토해 시퀀스의 의미를 이해해야 한다는 것을 의미

- 
- MiniLM-L12-H384-uncased 실험하기
 - RoBERTa-large-mnli 살펴보기
 - BERT 기반 다국어 모델 조사하기
 - GPT-3을 사용한 감성 분석

SST를 살펴보는 것으로 이 장을 시작해 보자.

12.1

SST(Stanford Sentiment Treebank)

이 장에서는 먼저 트랜스포머가 감성 분석에서 학습할 데이터셋 SST를 살펴본다. 그리고 AllenNLP를 사용해 RoBERTa-large 트랜스포머를 실행할 것이다.

2013년 소처(Socher et al.)은 긴 구문에 대한 의미론적 단어 공간을 설계했다. 그들은 긴 시퀀스에 적용하는 합성성의 원리를 정의했다. 합성성의 원리는 NLP 모델이 복잡한 문장의 구성 표현과 이를 결합하는 규칙을 검토해 시퀀스의 의미를 이해해야 한다는 것을 의미한다.

합성성의 원리를 파악하기 위해 SST에서 샘플을 가져와보자.

이 섹션과 챗터는 독립적으로 구성됐다. 섹션 내 작업을 직접 실행해 보든지, 내용을 읽고 제공된 스크린샷을 보든지 선택할 수 있다.

상호 작용할 수 있는 감성 트리뱅크(<https://nlp.stanford.edu/sentiment/treebank.html>)로 이동하자.

여러분들은 원하는 것을 선택할 수 있다. 감성 트리 그래프가 표시될 것이다. 이미지를 클릭하면 감성 트리를 볼 수 있다.

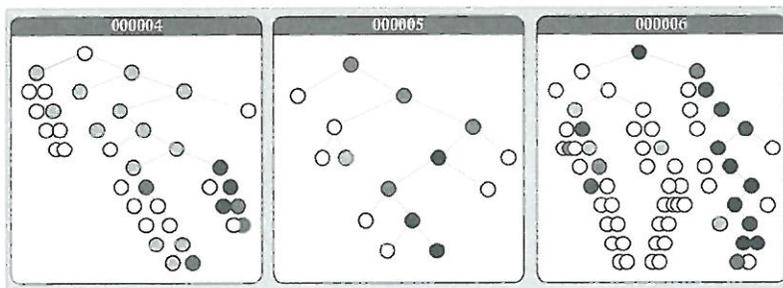


그림 12.1 감성 트리 그래프

이 예에서는 Jacques Derrida, a pioneer in deconstruction theories in linguistics(자크 데리다 (Jacques Derrida), 언어학의 해체주의 이론의 선구자)라는 문장을 포함하는 그래프 6을 클릭했다. 길고 복잡한 문장이 나타난다.

Whether or not you're enlightened by any of Derrida's lectures on the other and the self, Derrida is an undeniably fascinating and playful fellow.(자아와 타자에 대한 데리다(Derrida)의 강연으로 깨달음을 얻었든 아니든, 데리다가 매혹적이고 유쾌한 동료임은 부정할 수 없다.)

소처는 벡터 공간에서의 합성성과 논리 형식에 대해 연구했다.

예를 들어 자크 데리다 샘플을 구성하는 논리 형식을 정의한다는 것은 다음을 이해한다는 것을 의미한다.

- Whether, or, 그리고 not이라는 단어와 Whether 구문을 구분하는 쉼표를 해석하는 방법
- 또 다른 and를 갖는, 문장의 두 번째 부분(쉼표 뒤에 부분)을 이해하는 방법

벡터 공간을 정의한 후, 소처는 합성성의 원리를 나타내는 복잡한 그래프를 생성했다.

그래프를 섹션별로 볼 수 있다. 첫 번째 섹션은 문장의 Whether 세그먼트³⁹이다.

39 역주. NLP에서 세그먼트(segment)란, 문장의 부분적인 구조를 나타낸다. 단어, 구, 구문 등과 같은 작은 단위로 구성하는 것을 가리킨다.

소처가 고안한 방법에서 몇 가지 결론을 내릴 수 있다.

- 감성 분석은 문장에서 긍정 단어와 부정 단어를 세는 작업으로 단순화할 수 없다.
- 트랜스포머 모델 또는 어느 NLP 모델이든 복잡한 문장의 구성 요소가 논리 형식에 어떻게 부합하는지 이해하기 위해 합성성의 원리를 학습할 수 있어야 한다.
- 트랜스포머 모델은 복잡한 문장의 미묘한 차이를 해석하기 위해 벡터 공간을 구축할 수 있어야 한다.

이제 이 이론을 RoBERTa-large 모델을 사용해 실습해 보자.

12.1.1 RoBERTa-large를 사용한 감성 분석

이 섹션에서는 AllenNLP 리소스를 사용해 RoBERTa-large 트랜스포머를 실행할 것이다. 2019년 리우(Liu et al.)는 기존 BERT 모델을 분석한 결과, 예상만큼 학습하지 않았다는 사실을 발견했다. 모델 학습 시간을 생각해 보면 그리 놀라운 일은 아니었다. 이들은 BERT 모델의 사전 학습을 개선해 RoBERTa(Robustly Optimized BERT Pretraining Approach)를 개발했다.

그럼, `SentimentAnalysis.ipynb`에서 RoBERTa-large 모델을 실행해 보자.

첫 번째 셀을 실행해 `allennlp-models`를 설치하자.

```
!pip install allennlp==1.0.0 allennlp-models==1.0.0
```

이제 `Jacque Derrida` 샘플을 실행해 보자.

```
!echo '{"sentence": "Whether or not you're enlightened by any of Derrida's lectures on the other and the self, Derrida is an undeniably fascinating and playful fellow."}' | \W  
allennlp predict https://storage.googleapis.com/allennlp-public-models/sst-roberta-large-2020.06.08.tar.gz -
```

먼저 24개 층과 16개 어텐션 헤드를 가진 RoBERTa-large 모델의 아키텍처가 출력된다.

```

"architectures": [
  "RobertaForMaskedLM"
],
"attention_probs_dropout_prob": 0.1,
"bos_token_id": 0,
"eos_token_id": 2,
"hidden_act": "gelu",
"hidden_dropout_prob": 0.1,
"hidden_size": 1024,
"initializer_range": 0.02,
"intermediate_size": 4096,
"layer_norm_eps": 1e-05,
"max_position_embeddings": 514,
"model_type": "roberta",
"num_attention_heads": 16,
"num_hidden_layers": 24,
"pad_token_id": 1,
"type_vocab_size": 1,
"vocab_size": 50265
}

```

필요하다면 **3장, BERT 모델 미세 조정하기**의 BERT 모델 설정하기 섹션에서 설명한 BERT 아키텍처를 살펴보고 모델 파라미터를 수정해 활용할 수 있다.

감성 분석 결과는 0과 1사이 값을 갖는다. 0에 가까울수록 부정, 1에 가까울수록 긍정을 뜻한다.

로짓(logits)과 확률(probs)을 포함한 감성 분석 결과도 출력된다.

```

prediction: {"logits": [3.646597385406494, -2.9539334774017334], "probs":
[0.9986421465873718, 0.001357800210826099]}

```

NOTE

모델은 확률적 알고리즘을 사용한다. 따라서 실행할 때마다 결과는 달라질 수 있다.

또한, 사용한 토큰 ID와 최종 예측 라벨이 출력된다.

```
"token_ids": [0, 5994, 50, 45, 47, 769, 38853, 30, 143, 9, 6113, 10505, 281, 25798, 15, 5, 97, 8, 5, 1403, 2156, 211, 14385, 4347, 16, 41, 35559, 12509, 8, 23317, 2598, 479, 2], "label": "1",
```

모델이 지속적으로 업데이트되기 때문에 사용한 토큰 ID는 그림 12.2,3과 다를 수 있다.

사용한 토큰 자체도 출력된다.

```
"tokens": [{"<s>", "Wu0120Whether", "Wu0120or", "Wu0120not", "Wu0120you", "Wu0120re", "Wu0120enlightened", "Wu0120by", "Wu0120any", "Wu0120of", "Wu0120Der", "rid", "as", "Wu0120lectures", "Wu0120on", "Wu0120the", "Wu0120other", "Wu0120and", "Wu0120the", "Wu0120self", "Wu0120,", "Wu0120D", "err", "ida", "Wu0120is", "Wu0120an", "Wu0120undeniably", "Wu0120fascinating", "Wu0120and", "Wu0120playful", "Wu0120fellow", "Wu0120.", "</s>"}]
```

몇 가지 샘플을 입력해 사전 학습된 RoBERTa 모델을 직접 실험해 보는 것도 좋다.

다른 트랜스포머 모델을 사용해 감성 분석으로 고객 행동을 어떻게 예측하는지 알아보자.

12.2

감성 분석으로 고객 행동 예측하기

이번 섹션에서는 허깅페이스 트랜스포머 모델로 감성 분석 작업을 실행하고 어떤 모델이 가장 좋은 결과를 생성하는지를 확인한다.

먼저 허깅페이스 DistilBERT 모델을 사용해 보자.

12.2.1 DistilBERT를 사용한 감성 분석

DistilBERT로 감성 분석 작업을 수행한 결과로 고객 행동을 예측하는 방법을 살펴보자.

SentimentAnalysis.ipynb을 열어 트랜스포머를 설치하고 필요한 모듈을 불러오자.

```
!pip install -q transformers
from transformers import pipeline
```

시퀀스를 받아 모델을 실행하는 classify 함수를 정의하자.

```
def classify(sequence,M):
    #DistilBertForSequenceClassification(기본 모델)
    nlp_cls = pipeline('sentiment-analysis')
    if M==1:
        print(nlp_cls.model.config)
    return nlp_cls(sequence)
```

함수에 M=1을 전달하면 현재 사용 중인 DistilBERT의 설정이 출력된다.

```

DistilBertConfig {
  "activation": "gelu",
  "architectures": [
    "DistilBertForSequenceClassification"
  ],
  "attention_dropout": 0.1,
  "dim": 768,
  "dropout": 0.1,
  "finetuning_task": "sst-2",
  "hidden_dim": 3072,
  "id2label": {
    "0": "NEGATIVE",
    "1": "POSITIVE"
  },
  "initializer_range": 0.02,
  "label2id": {
    "NEGATIVE": 0,
    "POSITIVE": 1
  },
  "max_position_embeddings": 512,
  "model_type": "distilbert",
  "n_heads": 12,
  "n_layers": 6,
  "output_past": true,
  "pad_token_id": 0,
  "qa_dropout": 0.1,
  "seq_classif_dropout": 0.2,
  "sinusoidal_pos_embs": false,
  "tie_weights_": true,
  "vocab_size": 30522
}

```

층 개수가 6, 헤드 개수가 12이다. 설정 파라미터 중 몇몇은 이름만으로 정의를 알 수 있다.

이제 classify 함수에 보낼 시퀀스 목록을 만들어보자.

```

seq=3
if seq==1:
    sequence="The battery on my Model9X phone doesn't last more than 6 hours
and I'm unhappy about that."# 내 Model9X 핸드폰 배터리는 6시간도 버티지 못한다. 불만족스럽다.
if seq==2:
    sequence="The battery on my Model9X phone doesn't last more than 6 hours
and I'm unhappy about that. I was really mad! I bought a Moel10x and things
seem to be better. I'm super satisfied now."# 내 Model9X 핸드폰 배터리는 6시간도 버티지
못한다. 불만족스럽다. 정말 화가 났다! Moe110x는 나은 것 같다. 이번에는 정말 만족하고 있다.
if seq==3:
    sequence="The customer was very unhappy" #고객이 정말 불만족했다.
if seq==4:
    sequence="The customer was very satisfied"#고객이 정말 만족했다.
print(sequence)
M=0 #모델 구성 출력=1, 기본값=0
CS=classify(sequence,M)
print(CS)

```

원한다면 시퀀스를 추가해서 실험해도 좋다.

seq=3으로 세 번째 시퀀스의 결과를 보자. 예상하듯 부정 결과를 얻는다.

```

[{'label': 'NEGATIVE', 'score': 0.9997098445892334}]

```

이 결과에서 고객 행동을 예측하는 함수를 정의하는 데에 도움이 될 실마리를 얻을 수 있다.

- 예측 결과를 고객 관리 데이터베이스에 저장한다.
- 고객이 특정 기간(주, 월, 년) 동안 서비스나 제품에 대해 불만을 제기한 횟수를 센다. 불만을 자주 제기하는 고객은 더 나은 제품이나 서비스를 받기 위해 경쟁업체로 옮길 수 있다.
- 부정적인 피드백 메시지가 계속 발생하는 제품 및 서비스를 감지한다. 제품이나 서비스의 결함을 발견해 개선할 수 있다.

허깅페이스 트랜스포머를 실험해 보자.

12.2.2 허깅페이스 모델을 사용한 감성 분석

이번 섹션에서는 허깅페이스의 트랜스포머 모델 목록을 살펴보고 몇 가지 샘플로 결과를 평가한다. 프로젝트를 진행할 때 여러 모델을 테스트해 가장 적합한 모델을 사용하는 것이 좋다.

허깅페이스 모델을 실행할 것이다.(<https://huggingface.co/models>)

허깅페이스에서 제공하는 문서에서 우리가 사용할 각 모델에 대한 설명을 볼 수 있다.(<https://huggingface.co/transformers/>)

여러 모델을 테스트할 것이다. 모델을 구현할 경우 실행하려는 NLP 작업에 맞게 미세 조정하거나 사전 학습이 필요할 수도 있다. 각 경우 다음을 참고하면 된다.

- 미세 조정의 경우, 3장, BERT 모델 미세 조정하기를 참조
- 사전 학습의 경우, 4장, RoBERTa 모델 처음부터 사전 학습하기를 참조

먼저 허깅페이스에서 제공하는 모델(<https://huggingface.co/models>)을 살펴보고 Tasks 창에서 Text Classification을 선택하자.

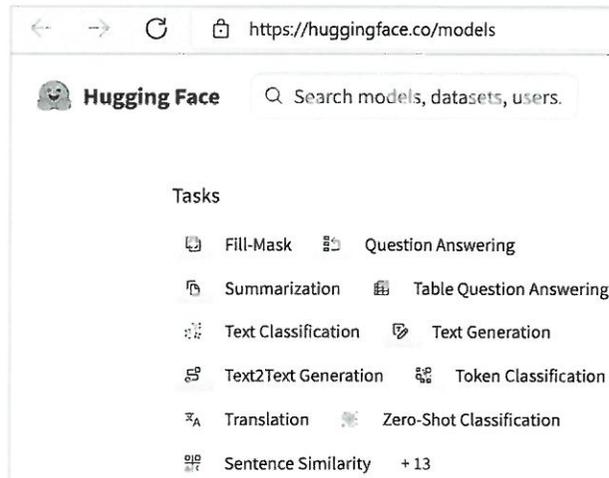


그림 12.4 텍스트 분류 모델 선택하기

텍스트 분류를 사전 학습한 트랜스포머 모델 목록

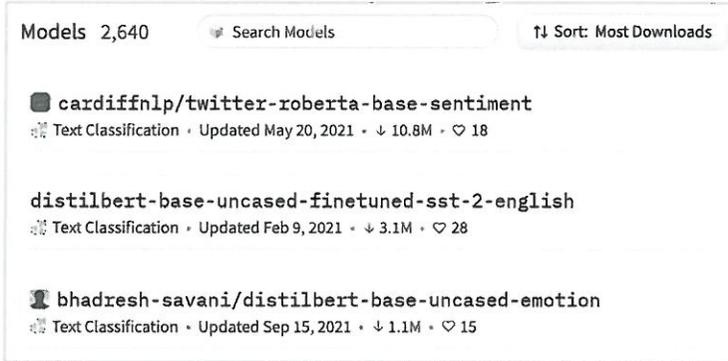


그림 12.5 텍스트 분류를 사전 학습한 허깅페이스 모델들

다운로드 수가 가장 많은(Most downloads) 것부터 나타난다. 이제 온라인에서 테스트할 수 있는 트랜스포머 모델을 살펴볼 것이다.

먼저 DistilBERT부터 시작해 보자.

12.2.2.1 SST 용도의 DistilBERT

distilbert-base-uncased-finetuned-sst-2-english 모델은 SST 데이터셋으로 미세 조정된 모델이다.

합성성의 원리를 잘 이해해야 하는 예를 들어보자.

Though the customer seemed unhappy, she was, in fact satisfied but thinking of something else at the time, which gave a false impression.(고객은 정말 불만족스러워 보였지만, 사실 그녀는 만족했으며 당시에 다른 생각을 하고 있어서 잘못된 인상을 주었다.)

이 문장은 트랜스포머가 분석하기 어렵고 논리 형식 학습을 요구한다.

모델 결과는 거짓 음성(false negative)이다.

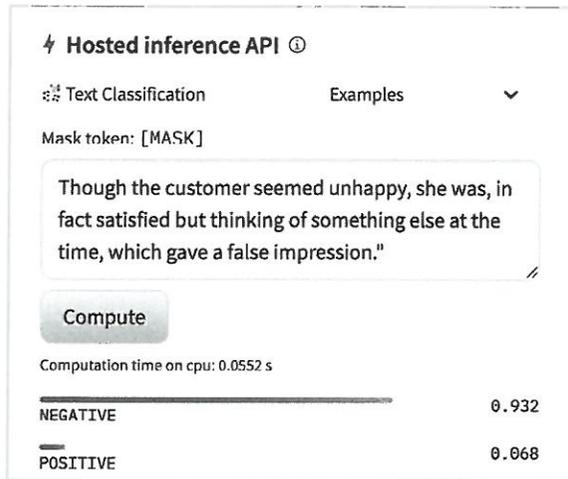


그림 12.6 복잡한 문장 분류 작업의 모델 결과

결과가 거짓 음성이라고 해서 모델이 제대로 작동하지 않는다는 의미는 아니다. 추가 학습이 필요하다는 의미일 수도 있다.

필자가 책을 집필하는 시점에 BERT 계열 모델은 GLUE 및 SuperGLUE 순위표에서 높은 순위를 기록하고 있다. 순위는 계속 바뀌겠지만 트랜스포머의 기본 개념은 바뀌지 않을 것이다.

이 예제는 실제 프로젝트에서 도움될 중요한 교훈을 준다. 모델이 프로젝트의 모든 문장을 정확하게 분류할 수는 없다. 즉, 고객이 불만을 제기한 횟수를 추정할 때면 거짓 음성과 거짓 양성(false positive)이 발생할 것이다. 따라서, 앞으로 몇 년 동안은 사람의 정기적인 개입이 필수적이다.

MiniLM 모델을 사용해 보자.

12.2.2.2 MiniLM-L12-H384-uncased

Microsoft/MiniLM-L12-H384-uncased 모델은 다른 BERT 기반 모델과는 다르게 마지막 셀프어텐션(self-attention) 층의 크기를 최적화하여 더 나은 성능을 얻은 모델이다. 12개 층, 3,300만 개 파라미터를 가지고 BERT-base보다 2.7배 빠르다.

합성성의 원리를 이해할 수 있는지 앞의 문장을 테스트해보자.

다음 그림과 같이 모델이 긍정과 부정 각각에 점수를 비슷하게 나눠 부여한 점이 흥미롭다.

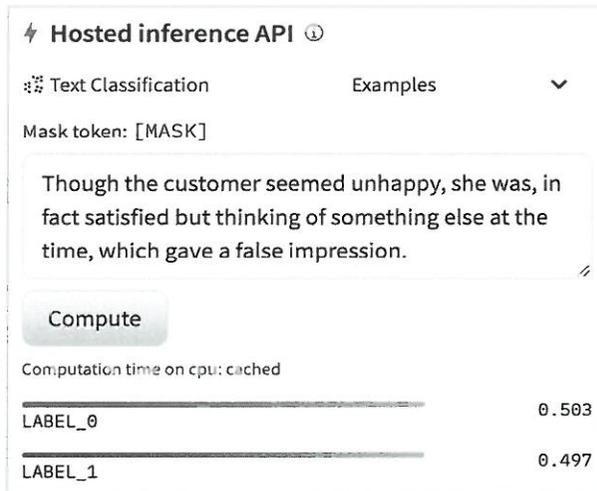


그림 12.7 복잡한 문장의 감성 분석 결과

긍정으로 나와야 하는 문장인데, 약 0.5점으로 긍정/부정에 결정적이지 않은 결과를 보여주는 것을 알 수 있다.

함의(entailment)와 관련된 모델을 사용해 보자.

12.2.2.3 RoBERTa-large-mnli

다중 장르 자연어 추론(Multi-Genre Natural Language Inference, MultiNLI) 작업(<https://cims.nyu.edu/~sbowman/multinli/>)은 고객이 의미하는 바를 파악할 때, 복잡한 문장 해석에 도움이 될 수 있다. 이 작업은 어떤 시퀀스가 다음 시퀀스를 함의하는지 여부를 판단하는 작업이다.

이를 위해서 입력 형식을 지정하고 시퀀스 분할 토큰으로 시퀀스를 분할해야 한다.

Though the customer seemed unhappy</s></s> she was, in fact satisfied but thinking of something else at the time, which gave a false impression

모델 결과는 중립(neutral)이다.

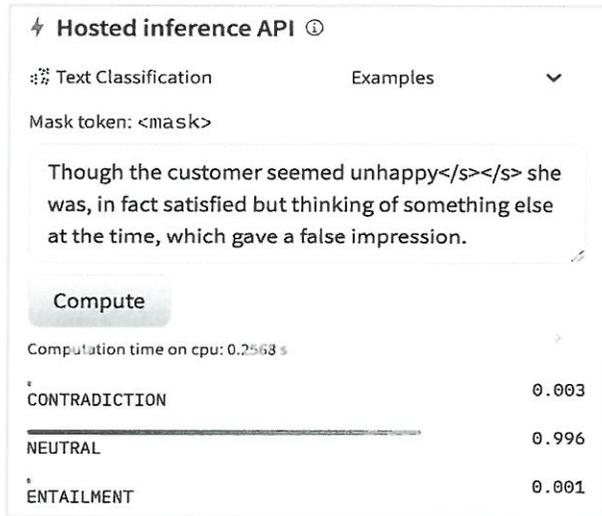


그림 12.8 약간 긍정적인 문장의 함의 추론 모델 결과

모델 결과에 오류는 없다. 두 번째 시퀀스를 첫 시퀀스로 추론할 수 없기 때문이다. 모델 결과는 정확하다.

마지막으로, 긍정 감성 문장으로 BERT 기반 다국어 모델을 실험해 보자.

12.2.2.4 BERT 기반 다국어 모델

마지막 실험으로 BERT 기반 다국어 모델(`nlptown/bert-base-multilingual-uncased-sentiment`.)을 실행해 보자.

이 모델은 아주 잘 설계된 모델이다.

친근하고 긍정적인 영어 문장을 테스트해 보자.

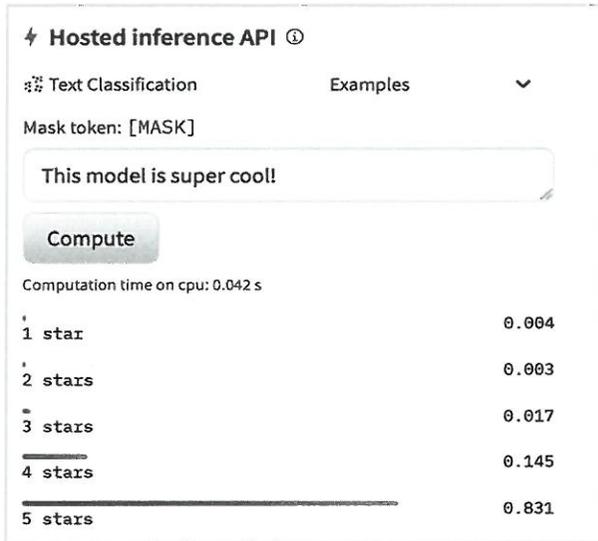


그림 12.9 영어 문장 감성 분석 결과

프랑스어로 “이 모델은 정말 멋져요”를 뜻하는 문장(“Ce modèle est super bien!”)을 테스트해 보자.

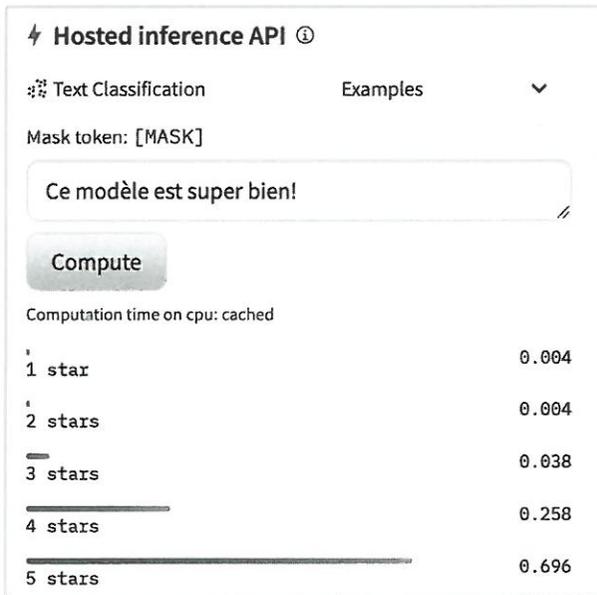


그림 12.10 프랑스어 문장 감성 분석 결과

허깅페이스에서 이 모델의 url은 <https://huggingface.co/nlptown/bert-base-multilingual-uncased-sentiment>이다.

다음 코드를 사용해 웹사이트에 구현할 수 있다.

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification
tokenizer = AutoTokenizer.from_pretrained("nlptown/bert-base-
multilingualuncased-sentiment")
model = AutoModelForSequenceClassification.from_pretrained("nlptown/
bertbase-multilingual-uncased-sentiment")
```

기다리는 시간이 약간 필요하지만, 좋은 결과를 얻을 수도 있다.

웹사이트에 이 모델을 구현해 전 세계 고객의 평균 만족도를 구할 수 있다. 또한 고객 서비스를 개선하고 고객 반응을 예측하기 위한 지속적인 피드백으로 사용할 수 있다.

이제 GPT-3는 어떻게 감성 분석을 하는지 알아보자.

12.3

GPT-3를 사용한 감성 분석

이번 섹션의 예제를 실행하려면 OpenAI 계정이 필요하다. 교육용 인터페이스는 API, 개발 또는 학습이 없이도 사용 가능하다. 사용자가 문장을 입력하기만 하면 된다. 몇 개의 트윗(tweet)을 입력해 감성 분석을 해보자.

Tweet: I didn't find the movie exciting, but somehow I really enjoyed watching it!(영화가 정말 재밌었던 것은 아니지만, 어쨌든 볼만했다!)

Sentiment: Positive(긍정)

Tweet: I never ate spicy food like this before but find it super good!(이렇게 매운 음식은 처음이었지만 정말 좋았다!)

Sentiment: Positive(긍정)

만족스러운 결과를 얻을 수 있었다. 좀 더 어려운 시퀀스를 입력해 보자.

Tweet: It's difficult to find what we really enjoy in life because of all of the parameters we have to take into account.(고려해야 할 변수들이 너무 많기 때문에 우리는 인생에서 진정으로 좋아하는 것을 찾기 쉽지 않다.)

Sentiment: Positive(긍정)

이번에 모델은 틀렸다! 입력한 문장은 전혀 긍정적이지 않다. 문장은 삶의 어려움을 토로하고 있다. “즐거다”를 뜻하는 단어 enjoy가 GPT-3에 편견을 유도했다.

단어 enjoy 대신 are으로 바꾸면 모델 결과는 부정이 된다.

Tweet: It's difficult to find what we really are in life because of all of the parameters we have to take into account.(고려해야 할 변수들이 너무 많기 때문에 우리는 인생에서 진정한 자기 자신을 찾기 쉽지 않다.)

Sentiment: Negative(부정)

하지만, 이 결과도 틀린 것이다! 인생을 파악하기 어렵다해서 트윗 시퀀스가 부정적이라고 판단할 수는 없다. 모델의 정확한 결과는 중립적(neutral)이어야 한다. 이후 다른 NLP 작업에서 GPT-3에게 왜 인생을 파악하기 어려운지 설명해 달라고 요청할 수도 있다.

NLP 작업과 관련없는 사용자가 NLP 작업을 실행한다는 것은 4차 산업혁명이 어디로 가고 있는지를 보여준다. 즉, 사람의 개입은 줄고 자동화된 기능은 강화하는 방향으로 가고 있다는 것을 알 수 있다. 하지만, 트랜스포머가 예상한 것과 다른 결과를 생성할 때, 전처리 기능을 설계하는 것과 같은 새로운 기술이 필요한 상황도 있다. 인간은 아직 쓸모가 있다!

바로 사용할 수 있는(ready-to-use) 코드를 사용한 트윗 분류 예시는 **7장, GPT-3 엔진을 사용한 초인간 트랜스포머 등장**의 OpenAI GPT-3 작업 실행하기 섹션에서 확인할 수 있다. 이번 섹션의 예제를 해당 코드로 구현할 수도 있다.

이제 우리가 아직 가치있다는 것을 어떻게 증명할 수 있을지 알아보자.

허깅페이스 트랜스포머를 사용해 감성 분석 결과가 중립으로 나온 문장이 있었다.

필자는 해당 문장을 중립으로 라벨링하는 것이 불편했다. OpenAI GPT-3가 더 잘할 수 있는지 궁금했다. 왜냐하면 GPT-3는 학습하지 않은 것들도 잘 할 수 있는 파운데이션 모델이기 때문이다.

필자는 12.2 챕터에 나왔던 다음 문장을 다시 살펴봤다.

Though the customer seemed unhappy, she was, in fact, satisfied but thinking of something else at the time, which gave a false impression.

문장을 자세히 보니, customer가 she인 것을 알 수 있었다. 더 들여다보면 그녀(he)가 실제로는 만족(in fact satisfied)했음을 알 수 있다. 효과가 있는 모델을 찾을 때까지 맹목적으로 모델 테스트를 하지 않기로 결심했다. 이런 방식은 생산적이지 않다.

필자는 문제의 근원 파악에 논리와 실험이 필요했다. 자동으로 원인을 파악해 주는 알고리즘에 의존하고 싶지 않았다. 때때로 우리는 우리의 뉴런을 사용해야 한다.

기계가 she를 customer로 식별하기 어렵다는 게 문제였을까? 10장, BERT 기반 트랜스포머를 사용한 SRL에서 했던 것처럼 SRL BERT를 사용해 보자.

12.4.1 SRL을 사용해 대명사 식별 알아보기

10장은 SRL과 다른 도구를 함께 사용하는 것을 권장하며 끝났다. 이번 섹션에서 그렇게 해볼 것이다.

먼저 She was satisfied를 Semantic Role Labeling 인터페이스에서 실행해 봤다.⁴⁰

결과는 정확했다.

⁴⁰ 역주. allennlp 데모의 서비스가 종료되었다. 대신 <https://hanlp.hankcs.com/en/demos/srl.html>에서 같은 테스트를 실행해볼 수 있다.

think의 Frames :

she	was	satisfied
논항1	수식어	논항2

그림 12.11 간단한 문장의 SRL 결과

술어 was 프레임의 분석 결과는 명확하다. was가 동사, She가 논항1, satisfied가 논항2이다.

문장을 조금 더 복잡하게해도 동일한 결과를 얻을 수 있을까? 동일한 결과가 나오는 것을 알 수 있다.

was의 Frames :

she	was	in fact	satisfied but thinking of something else at the time
논항1	서술어	수식어-영향	논항2

for vacation
수식어-영향

그림 12.12 다른 단어와 합쳐진 동사 "satisfied", 잘못된 결과

satisfied는 여전히 논항2로 분류돼, 정확하게 분석됐다.

was 의미를 수정하는 수식어-영향에 주목해보자. 수식어-영향는 thinking을 갖는 논항2와 연관있어서 단어 false는 오해의 소지가 있다.

술어 thinking은 false impression을 주었다. 하지만 thinking은 이 문장에서 술어로 식별되지 않았다. 10장, BERT 기반 트랜스포머를 사용한 SRL에서 보았던 것처럼, 생략된 she was가 식별되지 않은게 아닐까?

생략없이 전체 문장을 입력해 보자.

Though the customer seemed unhappy, she was, in fact, satisfied but she was thinking of something else at the time, which gave a false impression.

10장에서 본 것처럼 SRL을 사용할 때 표현 일부를 생략하면 문제가 발생한다는 것을 여기서도 확인할 수 있다. 전체 문장의 SRL 결과를 보면 다섯 개의 정확한 술어를 가진 프레임을 확인할 수 있다.

- she was의 생략

생략하지 않은 문장의 결과를 보자.

Though the customer seemed unhappy, she was, in fact, satisfied but she was thinking of something else at the time, which gave a false impression
 결과는 여전히 부정이다.

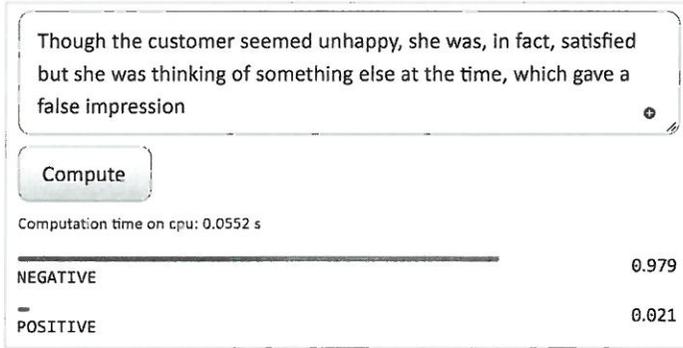


그림 12.16 거짓 음성

- 단어 false가 없다면 긍정적인 문장

문장에서 false를 제거하되 원래 문장처럼 she was를 생략하자.

Though the customer seemed unhappy, she was, in fact, satisfied but thinking of something else at the time, which gave an impression
 드디어 긍정 결과를 얻었다!

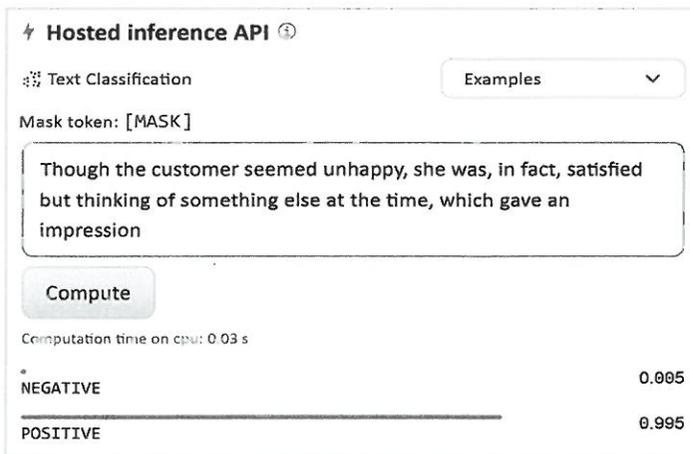


그림 12.17 참 양성

일부 표현을 생략한 문장에서 단어 false가 SRL에 혼란을 야기한다는 것을 확인했다.

또한 false가 허깅페이스 감성 분석 모델에도 혼란을 야기한다는 것을 확인했다.

GPT-3가 더 잘할 수 있을까? GPT-3에서도 테스트해보자.

12.4.3 GPT-3를 사용한 감성 분석하기

OpenAI Playground(플레이그라운드)의 트윗 분류기(tweet classifier) 예제를 수정해 문장을 분석해보자. 세 단계로 진행된다.

- 1단계: GPT-3에 기대하는 것 보여주기.

Sentence: The customer was satisfied(문장: 그 고객은 만족했어.)

Sentiment: Positive(감성: 긍정)

Sentence: The customer was not satisfied(문장: 그 고객은 불만족했어.)

Sentiment: Negative(감성: 부정)

Sentence: The service was 🍷(문장: 그 서비스는 🍷)

Sentiment: Positive(감성: 긍정)

Sentence: This is the link to the review(문장: 다음은 리뷰 링크입니다.)

Sentiment: Neutral(감성: 중립)

- 2단계: 원하는 출력 형식 예시 보여주기.

1. "I loved the new Batman movie!"(새로 나온 배트맨 영화가 아주 좋았어!)

2. "I hate it when my phone battery dies"(휴대폰 배터리가 방전되는 게 너무 싫어)

3. "My day has been 🍷"(나의 하루는 🍷였어)

4. "This is the link to the article"(다음은 기사 링크입니다.)

5. "This new music video blew my mind"(새로운 뮤직비디오는 내 마음을 사로 잡았어)

Sentence sentiment ratings.(문장 감성 평가)

1: Positive(긍정)

2: Negative(부정)

3: Positive(긍정)

4: Neutral(부정)

5: Positive(중립)

- 3단계: 분석할 문장 입력하기(아래 세 번째 문장이 관심 문장)

1. "I can't stand this product"(이 제품 정말 싫어.)

2. "The service was bad! 🍷"(서비스가 별로야! 🍷)

3. "Though the customer seemed unhappy she was in fact satisfied but thinking of something else at the time, which gave a false impression"(고객은 정말 불만족스러워 보였지만, 사실 그녀는 만족했으며 당시에 다른 생각을 하고 있어서 잘못된 인상을 주었다.)
4. "The support team was ♥♥"(지원 팀은 ♥♥)
5. "Here is the link to the product."(제품 링크입니다.)

Sentence sentiment ratings.(문장 감성 평가)를 입력하면 다음 결과를 얻는다.

- 1: Negative(부정)
- 2: Positive(긍정)
- 3: Positive(긍정)
- 4: Positive(긍정)
- 5: Neutral(중립)

세 번째 문장의 결과가 긍정으로 정확한 결과를 얻었다. 이 결과를 신뢰할 수 있을까? OpenAI Playground에서 예제를 입력해 실험해볼 수 있다. 여기서는 코드를 사용한 결과를 확인해 보자.

12.4.3.1 GPT-3 코드를 사용한 분석

OpenAI Playground에서 View code를 클릭하고 curl, 파이썬, node.js, json 등 여러 라이브러리 환경의 코드를 볼 수 있다. 파이썬을 선택 후 보이는 코드를 복사해 SentimentAnalysis.ipynb에 붙여 넣고 응답의 "choices"만 출력하는 코드를 한 줄 추가해 실행해 보자.

```
response = openai.Completion.create(
    engine="davinci",
    prompt="This is a Sentence sentiment classifier\nSentence: W\"The customer was satisfied\n\nSentiment: Positive\n\n\n\nSentence: W\"The customer was not satisfied\n\n\nSentiment: Negative\n\n\n\nSentence: W\"The service was 🍷\n\n\nSentiment: Positive\n\n\n\nSentence: W\"This is the link to the review\n\n\nSentiment: Neutral\n\n\n\nSentence text\n\n\n\n1. W\"I loved the new Batman movie!\n\n\n2. W\"I hate it when my phone battery dies\n\n\n3. W\"My day has been 🍷\n\n\n4. W\"This is the link to the article\n\n\n5. W\"This new music video blew my mind\n\n\n\n\nSentence sentiment ratings:\n n1: Positive\nn2: Negative\nn3: Positive\nn4: Neutral\nn5: Positive\n\n\n\n\n\n\nSentence text\n\n\n\n1. W\"I can't stand this product\n\n\n2. W\"The service was bad! 🤬\n\n\n3. W\"Though the customer seemed unhappy she was in fact satisfied but thinking of something else at the time, which gave a false impression\n\n\n4. W\"The support team was ♥♥\n\n\n5. W\"Here is the link to the product.\n\n\n\n\n\nSentence sentiment ratings:\n\"
```

```

temperature=0.3,
max_tokens=60,
top_p=1,
frequency_penalty=0,
presence_penalty=0,
stop=["###"]
)
r = (response["choices"])[0]
print(r["text"])

```

출력된 내용을 보니, 응답이 일정하지 않다.

- 첫 번째 실행: 우리 문장(3)은 중립이다.
 - 1: Negative(부정)
 - 2: Negative(부정)
 - 3: Neutral(중립)
 - 4: Positive(긍정)
 - 5: Positive(긍정)
- 두 번째 실행: 우리 문장(3)은 긍정이다.
 - 1: Negative(부정)
 - 2: Negative(부정)
 - 3: Positive(긍정)
 - 4: Positive(긍정)
 - 5: Neutral(중립)
- 세 번째 실행: 우리 문장(3)은 긍정
- 네 번째 실행: 우리 문장(3)은 부정

이를 통해 아래 결론을 얻을 수 있다.

- SRL은 문장이 간단하고 완전한 경우(생략이 없는 경우) 신뢰할 수 있는 감성 분석 결과를 얻을 수 있다.
- SRL은 문장이 적당히 어려운 경우 신뢰할 수 있는 결과를 얻을 수도, 아닐 수도 있다.
- SRL은 문장이 복잡한 경우(생략이 있거나, 여러 명제를 포함하거나, 모호한 구문이 많은 경우 등)에는 결과가 안정적이지 않아 신뢰할 수 없다.

따라서 앞으로 개발자의 역할은 다음과 같을 것이다.

- 클라우드 AI와 바로 사용 가능한 모듈로 인해 AI 개발이 덜 필요할 것이다.
- 더 많은 설계 기술이 필요할 것이다.
- AI 알고리즘을 공급하고, 제어하고, 결과를 분석하는 고전적인 파이프라인 개발에 사고력과 목표 지향적인 개발이 필요할 것이다.

이 장에서는 사상가(thinker), 설계자, 파이프라인 개발자로서 개발자의 미래를 보여준다!

이제 12장의 내용을 정리하고 트랜스포머의 새로운 지평을 살펴보자.

12.5 정리하기

이번 장에서는 몇 가지 고급 이론을 살펴봤다. 합성성의 원리는 직관적이지 않다. 합성성의 원리는 트랜스포머 모델이 전체 문장을 이해하기 위해서는 문장의 모든 부분을 이해해야 한다는 것을 의미한다. 여기에는 문장 세그먼트 간의 연결을 제공하는 논리 형식 규칙을 포함한다.

감성 분석의 이론적 난이도 때문에 많은 양의 트랜스포머 모델 학습, 인력과 컴퓨팅 리소스가 필요하다. 많은 트랜스포머 모델이 다양한 작업을 학습했지만, 특정 작업에 대해서는 더 많은 학습이 필요한 경우가 많다.

우리는 RoBERTa-large, DistilBERT, MiniLM-L12-H384-uncased, 그리고 우수한 BERT 기반 다국어 모델을 테스트했다. 일부 모델은 흥미로운 결과를 보였지만 더 많은 학습이 필요한 것을 확인할 수 있었다.

감성 분석은 매우 복잡한 시퀀스와 문장에 대한 깊은 이해를 필요로 한다. 따라서 감성 분석이 아닌 RoBERTa-large-mnli 모델을 사용해 문장 간 합의 여부도 추론해 보았다. 여기서 얻은 교훈은 한 가지 결과만을 사용하지 말자는 것이다. 다양한 작업에 다양한 모델을 사용해보자. 트랜스포머의 유연성 덕분에 동일 모델에 다양한 작업, 혹은 다수 모델에 동일한 작업을 시도할 수 있다.

이 과정에서 고객 관계를 개선하기 위한 아이디어를 얻기도 했다. 고객의 불평이 잦다면 경쟁 업체로 떠날 수 있다. 여러 고객이 제품이나 서비스에 대해 불만을 제기하면 향후 발생할 수 있는 문제를 예측하고 제품이나 서비스를 개선해야 한다. 또한 고객의 피드백을 트랜스포머 모델로 실시간 분석해 서비스 품질을 파악할 수도 있다.

마지막으로, GPT-3 온라인 인터페이스만으로 감성 분석을 했다. 놀라울 정도로 효과적이었지만, 어려운 시퀀스를 풀기 위해서는 여전히 사람이 필요하다는 것을 알 수 있었다. SRL이 복잡한 시퀀스에서 문제를 식별하는 데 어떻게 도움이 되는지 확인했다.

사상가, 설계자, 파이프라인 개발자로서 개발자의 미래는 매우 밝다는 결론을 내릴 수 있다.

다음 13장 트랜스포머로 가짜 뉴스 분석하기에서는 감성 분석을 사용해 가짜 뉴스에 대한 감정 반응(emotional reactions)을 분석해 볼 것이다.

12.6 문제

01. 감성 분석을 위해 트랜스포머를 사전 학습할 필요는 없다. 참 / 거짓
02. 문장은 항상 긍정적이거나 부정적이다. 중립적일 수는 없다. 참 / 거짓
03. 합성성의 원리는 트랜스포머가 문장의 모든 부분을 이해해야 한다는 것을 의미한다. 참 / 거짓
04. RoBERTa-large는 트랜스포머의 사전 학습 과정을 개선하기 위해 설계됐다. 참 / 거짓
05. 트랜스포머는 고객이 만족하는지 여부를 알려줄 수 있다. 참 / 거짓
06. 제품이나 서비스의 감성 분석 결과가 지속적으로 부정적이라면, 분석 결과는 제품이나 서비스를 개선하기 위한 적절한 결정을 내리는 데 도움이 된다. 참 / 거짓
07. 모델 결과가 좋지 않다면, 모델을 바꾸기 전에 더 학습해야 한다. 참 / 거짓

12.7 참고 문헌

- Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher Manning, Andrew Ng, and Christopher Potts, Recursive Deep Models for Semantic Compositionality over a Sentiment Treebank: https://nlp.stanford.edu/~socherr/EMNLP2013_RNTN.pdf
- 허깅페이스의 파이프라인과 모델 관련 문서:
https://huggingface.co/transformers/main_classes/pipelines.html
<https://huggingface.co/models>
<https://huggingface.co/transformers/>
- Yinhan Liu, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov, 2019, RoBERTa: A Robustly Optimized BERT Pretraining Approach: <https://arxiv.org/pdf/1907.11692.pdf>
- 앨런 인공지능 연구소: <https://allennlp.org/>
- 앨런 인공지능 연구소의 독해력 관련 자료: <https://demo.allennlp.org/sentiment-analysis>
- RoBERTa-large에 기여한 자오펑 우(Zhaofeng Wu): <https://zhaofengwu.github.io/>
- 스탠퍼드 감성 트리: <https://nlp.stanford.edu/sentiment/treebank.html>

13장

트랜스포머로 가짜 뉴스 분석하기

- 13.1 가짜 뉴스에 대한 감정 반응
- 13.2 가짜 뉴스에 대한 이성적 접근법
- 13.3 마치기 전에
- 13.4 정리하기
- 13.5 문제
- 13.6 참고 문헌

우리 모두는 처음 태어났을 때 지구가 평평하다고 생각하게 된다. 아기 때 평평한 바닥에서 기어 다녔으며, 유치원 때는 평평한 놀이터에서 놀았다. 초등학교에서는 평평한 교실에 앉아 있었다. 그런 다음, 부모님과 선생님들이 지구가 둥글고, 지구 반대편에 사는 사람들이 뒤집힌 상태라고 말해주었다. 우리가 왜 지구에서 떨어지지 않는지 이해하는 데는 꽤 오랜 시간이 걸렸다. 심지어 오늘날에도 아름다운 일몰을 볼 때, 우리는 여전히 “해가 지는 것”을 보고, 지구가 태양을 회전하는 것을 직접 관찰할 수는 없다!

가짜 뉴스와 그렇지 않은 것을 파악하는 데는 시간과 노력이 필요하고, 우리는 가짜 뉴스와 그렇지 않은 것을 구별해나가야 한다.

이 장에서는 의견이 대립하는 몇 가지 주제를 다룰 것이다. 우리는 기후 변화, 총기 통제, 그리고 도널드 트럼프의 트윗과 같은 주제들에 대해 사실이 무엇인지 확인할 것이다. 이를 위해 트윗, 페이스북 포스트, 그리고 다른 정보 소스들을 분석에 활용할 것이다.

우리의 목표는 결코 누군가나 무언가를 판단하는 것이 아니다. 가짜 뉴스는 의견과 사실 모두를 포함한다. 뉴스는 종종 현지 문화에 의한 사실 인식에 따라 달라진다. 따라서, 주제에 대한 더 많은 정보를 수집하고, 매일 받는 정보의 정글에서 길을 찾는 데 도움이 될 아이디어와 도구를 활용할 것이다.

트랜스포머의 성능이 아닌, 윤리적인 방법에 초점을 맞출 것이다. 이런 이유로 GPT-3 엔진을 사용하지 않을 것이다. 인간의 판단을 대체하지 않는 대신, 우리는 사람들이 직접 판단을 내릴 수 있는 도구를 제공할 것이다. GPT-3 엔진은 많은 작업에 대해 인간 수준의 성능을 달성했다. 그러나, 도덕적이고 윤리적인 결정은 인간에게 맡겨두어야 한다.

우선, 가짜 뉴스에 대해 감정적이고 이성적으로 반응하는 방식을 정의하는 것부터 시작해 볼 것이다.

그다음으로, 트랜스포머와 휴리스틱을 이용해 가짜 뉴스를 식별하는 몇 가지 방법을 정의할 것이다.

이전 장에서 구축한 자원을 이용해 가짜 뉴스를 이해하고 설명할 것이다. 의견에 대한 판단은 하지 않을 것이다. 또한, 뉴스를 설명하는 트랜스포머 모델을 제공할 것이다. 이를 이용해 원한다면 메시지가 가짜 뉴스임을 탐지하는 강력한 트랜스포머 모델을 만들어 볼 수 있을 것이다.

교육 목적으로 트랜스포머 모델을 사용하는 것이며, 뉴스의 사실 여부와는 무관하다.

이 장은 다음의 주제들을 다룬다.

- 인지적 불일치
- 가짜 뉴스에 대한 감정적 반응
- 가짜 뉴스에 대한 행동적 특징
- 가짜 뉴스에 대한 이성적 접근
- 가짜 뉴스 해결 로드맵
- 소셜 미디어에 감성 분석 트랜스포머 적용하기
- NER와 SRL을 이용해 총기 규제에 관한 의견 분석하기
- 트랜스포머에 의해 추출된 정보를 사용하여 신뢰할 수 있는 웹사이트 찾기
- 트랜스포머 결과를 사용해 교육에 활용하기
- 객관적이지만 비판적인 시각으로 전 대통령 트럼프의 트윗을 읽는 방법

첫 번째 단계는 가짜 뉴스에 대한 감정적이고 이성적인 반응을 탐색하는 것이다.

인간의 행동은 우리의 사회적, 문화적, 경제적 결정에 엄청난 영향을 끼친다. 인간의 감정은 이성적 사고만큼이나, 그보다 더 많이, 우리의 경제에 영향을 미친다. 행동 경제학은 감정이 우리의 의사 결정과정에 미치는 영향을 설명한다. 물건을 구입할 때 실제로 필요한 물건을 구입하기도 하지만 때때로 감정적 욕구를 만족시키는 소비재를 구입하기도 한다. 때로는 예산을 초과하더라도 순간의 충동에 휩쓸려 스마트폰을 사게 될 수도 있다.

가짜 뉴스에 대한 감정적이고 이성적인 반응은 입력되는 정보에 대해 느리게 생각하느냐, 빠르게 반응하느냐에 따라 달라진다. 다니엘 카너먼(Daniel Kahneman)은 이 과정을 그의 연구와 책, 『생각에 관한 생각(Thinking, Fast and Slow)(2013)』에서 설명했다.

그는 버논 스미스(Vernon L. Smith)와 함께 행동경제학 연구로 노벨 경제학상을 수상했다. 우리는 행동할 때 이전에 합리적이라고 생각했던 것들을 바탕으로 결정을 한다. 하지만 불행히도, 이러한 결정 중 많은 부분은 이성이 아닌 감정에 기반한다.

이러한 개념을 행동 플로우 차트로 가짜 뉴스에 적용해 보자.

13.1.1 인지부조화가 일으키는 감정적 반응

인지부조화는 가짜 뉴스를 트위터, 페이스북 및 기타 소셜 미디어 플랫폼의 최상위 순위로 이끈다. 트윗의 내용에 서로 이견이 없다면 논쟁이 일어나지 않을 것이다. 예를 들어, 만일 누군가 “기후 변화가 중요하다”라고 트윗을 쓴다면 다른 사람들은 별다른 반응이 없을 것이다.

우리 마음속에서 상반된 생각들 사이에 긴장이 쌓이면 인지부조화 상태에 진입한다. 결과적으로 우리는 토스터기에서의 합선이 생기는 것처럼 불안정한 상태가 된다.

생각해 볼 만한 많은 예시가 있다. 우리가 야외에 있을 때 코로나19로 인한 마스크 착용이 필요한가? 록다운은 좋은 것인가 나쁜 것인가? 코로나바이러스 백신은 효과적인가? 아니면 코로나바이러스 백신은 위험한가? 인지부조화는 간단한 노래를 연주하면서 계속해서 실수를 저지르는 음악가와 같다.

가짜 뉴스 증후군은 인지부조화를 기하급수적으로 증가시킨다. 한 전문가는 백신이 안전하다고 주장하고, 다른 전문가는 조심해야 한다고 말한다. 한 전문가는 야외에서 마스크를 착용하는 것이 쓸모없다고 말하고, 다른 한 전문가는 뉴스 채널에서 착용해야 한다고 주장한다. 각 진영은 상대방을 가짜 뉴스라고 비난하기도 한다.

한 진영에게는 상당한 부분의 가짜 뉴스가 상대 진영의 진실인 것처럼 보일 것이다.

한 가지 예로, 2022년에 미국의 공화당과 민주당은 2020년 대통령 선거 규칙이나 예정된 선거의 조직에 대해 합의하지 못한 상태였다.

우리는 한 신문을 펼쳐보고 다른 반대 의견을 읽기만 해도 더 많은 주제를 찾아볼 수 있다. 그럼에도 불구하고, 이 챕터에서는 다음 예시들로부터 몇 가지 상식적인 전제를 얻을 수 있다.

- 가짜 뉴스를 자동으로 감지하는 트랜스포머 모델을 찾는 것은 의미가 없다. 소셜 미디어와 다문화적 표현의 세계에서 각 그룹에게는 진실이 다른 그룹에게는 가짜 뉴스이기도 하다.
- 한 문화에서 진리로 받아들여지는 것을 다른 문화에서의 진리로 표현하려는 것은 의미가 없다. 글로벌 시대에서 각 나라마다, 각 대륙마다 그리고 소셜 미디어의 어디에서든 문화는 다양하다.
- 가짜 뉴스가 절대적이라는 것은 미신이다.
- 우리는 가짜 뉴스에 대한 더 나은 정의를 찾아야 한다.

필자의 의견은 (물론 사실은 아니지만!)가짜 뉴스는 인지적인 갈등상태로, 인지적 추론에 의해서만 해소될 수 있는 것이라는 것이다. 그렇기 때문에 가짜 뉴스 문제를 해결하는 것은 두 당사자 간 또는 우리 자신의 마음 안에서 갈등을 해소하는 것과 같다.

필자가 권장하고 싶은 것은 트랜스포머 모델로 각각의 갈등 같은 긴장을 해체하고 아이디어를 분석하는 것이다. “가짜 뉴스와 싸우기”, “내면의 평화 찾기”와 같은 것을 하려는 것이 아니며 트랜스포머를 사용하는 척하며 “가짜 뉴스에 대항하기 위해 절대적인 진실 찾기”를 하려는 것도 아니다.

우리는 단어 시퀀스(메시지)에 대한 더 깊은 이해를 얻고 주제에 대해 보다 심오하고 광범위한 의견을 형성하기 위해 트랜스포머를 사용할 것이다.

더 많은 사람들이 트랜스포머를 사용해 더 나은 시야와 의견을 얻을 수 있으면 하는 바람이다.

이를 위해 이 챕터를 직접 실행시켜 가며 이해할 수 있도록 설계했다. 트랜스포머는 언어 시퀀스에 대한 이해를 깊게 하고 보다 광범위한 의견을 형성하며 우리의 인지 능력을 개발하는 훌륭한 방법이다.

이제, 누군가 갈등의 여지가 있는 트윗을 게시할 때 무슨 일이 벌어지는지 알아보자.

13.1.1.1 갈등을 일으키는 트윗 분석하기

다음 트윗은 내가 일부 변형한 트위터에 게시된 메시지다. 이 챗터에서 보이는 트윗들은 트위터 인터페이스 표시 없이 원시 데이터만 나타내고 있다. 만약 주요 정치인이나 유명 배우가 아래와 같은 내용을 트윗으로 올린다면 많은 사람들이 그 내용에 동의하지 않을 것이다.

```
Climate change is bogus. It's a plot by the liberals to take the economy  
down.  
(기후 변화는 허위이다. 자유주의자들이 경제를 무너뜨리려는 음모이다.)
```

이것은 감정적인 반응을 일으킬 것이다. 트윗이 여기저기서 쌓일 것이다. 이것은 바이러스성을 가지고 트렌드가 되어 퍼져나갈 것이다.

트윗을 트랜스포머 도구에서 실행하여 이 트윗이 어떻게 누군가의 마음속에 인지부조화 폭풍을 일으킬 수 있는지 이해해 보자.

이 섹션에서 사용할 Fake_News.ipynb 노트북을 열어보자.

NOTE

```
구글 코랩에서 Fake_News.ipynb 노트북을 실행하는 데 문제가 있는 경우, Fake_News_Analysis_with_ChatGPT.ipynb 노트북을 실행해 보자.
```

앨런 인공지능 연구소로 시작해 보자. 우리는 12장, 고객 감정을 감지해 예측하기에서 감정 분석에 사용한 RoBERTa 트랜스포머 모델을 실행할 것이다.

먼저 allennlp-models를 설치해야 한다.

```
!pip install allennlp==1.0.0 allennlp-models==1.0.0
```

AllenNLP는 계속해서 버전을 업데이트하고 있다. 작성 당시에는 2.4.0 버전까지 존재하지만, 이번 장의 예제에서는 낮은 버전으로도 충분하다. 업데이트된 확률적 알고리즘이나 모델은 서로 다른 결과를 생성할 수 있다.

그런 다음 우리는 트윗의 출력을 자세히 분석하기 위해 다음 셀을 Bash로 실행한다.(모델 및 출력에 대한 정보는 다음과 같다)

```
!echo '{"sentence":"Climate change is bogus. It is a plot by the liberals to  
take the economy down."}' | W  
allennlp predict https://storage.googleapis.com/allennlp-public-models/sst-  
roberta-large-2020.06.08.tar.gz -
```

출력 결과는 트윗이 부정적임을 보여준다. 긍정적인 값은 0이고, 부정적인 값은 거의 1에 가깝다.

```
"probs": [0.0008486526785418391, 0.999151349067688]
```

트랜스포머의 확률적인 알고리즘으로 인해 출력 결과는 실행마다 다를 수 있다.

이제 <https://allennlp.org/>로 이동하여 분석 결과를 시각적으로 표현해 볼 것이다.

트랜스포머 모델은 지속적으로 학습되고 업데이트되므로 결괏값은 실행 시점마다 다를 수 있다. 이 장에서는 트랜스포머 모델의 추론에 초점을 맞출 것이다.

감성 분석(<https://demo.allennlp.org/sentiment-analysis>)을 선택하고 **RoBERTa large 모델**을 사용하여 분석을 실행한다.⁴¹

마찬가지로 트윗 내용이 부정적이라는 결과를 얻었다. 여기서 더 나아가, RoBERTa의 결정에 어떤 단어들이 영향을 미쳤는지 볼 수 있다.

모델 해석(Model interpretations)으로 이동한다. 모델을 해석하면 결과 도출 방식에 대한 통찰력을 얻을 수 있다. 한 가지를 선택해 살펴보거나 세 가지 옵션을 자세히 들여다 볼 수 있다.

41 역주. allennlp 데모의 서비스가 종료되었다. 모델 해석 기능은 제공하지 않지만, <http://text-processing.com/demo/sentiment/>에서 다양한 문장의 감성 분석 결과를 확인할 수 있다. 문장의 감성 분석 결과를 확인할 수 있다.

- **간단한 기울기 시각화(Simple Gradient Visualization):** 이 접근 방식은 두 가지 시각화를 제공한다. 첫 번째 시각화는 입력과 관련된 클래스 점수의 기울기를 계산한다. 두 번째 시각화는 클래스와 입력으로부터 추론된 중요한 특징 (saliency map)을 제공한다.
- **통합 기울기 시각화(Integrated Gradient Visualization):** 이 모델은 신경망 변경이 필요 없다. 이 시각화의 아이디어는 신경망의 입력으로부터 예측 결과를 생성하는 데 사용된 기울기의 기여도를 묘사하는 것이다.
- **부드러운 기울기 시각화(Smooth Gradient Visualization):** 이 접근 방식은 출력 예측과 입력을 사용하여 기울기를 계산한다. 목표는 입력의 특징을 식별하는 것이다. 그러나 해석을 개선하기 위해 노이즈가 추가된다.

이 섹션에서는 모델 해석으로 이동한 다음, 간단한 기울기 시각화를 클릭하고 예측 해석을 클릭하여 다음 표현을 얻어보자.

```
<s> Climate Change is bogus . It is plot by the liberals to take the  
economy down . </s>
```



Visualizing the top 3 most important words.

그림 13.1 상위 3개의 가장 중요한 단어 시각화

부정 예측에 큰 영향을 미치는 것은 is + bogus + plot이다.

이 시점에서, 우리가 인지부조화를 설명하기 위해 이렇게 간단한 예를 살펴보는 이유에 대해 궁금해할 수 있다. 아래 트윗에서 이에 대해 설명할 것이다.

강경파 공화당원이 앞의 트윗을 작성했다. 이 구성원을 Jaybird65라고 부르겠다. 놀랍게도, 다른 공화당원이 다음 트윗을 했다.

```
I am a Republican and think that climate change consciousness is a great  
thing!  
(저는 공화당원이며 기후 변화에 대한 인식은 훌륭한 일이라 생각합니다)
```

이 트윗은 Hunt78이라는 구성원이 작성했다. 이 문장을 Fake_News.ipynb에서 실행해 보자.

```
!echo '{"sentence":"I am a Republican and think that climate change
consciousness is a great thing!"}' | W
allennlp predict https://storage.googleapis.com/allennlp-public-models/
sst-roberta-large-2020.06.08.tar.gz -
```

출력은 당연하게도 “긍정”이다.

```
"probs": [0.9994876384735107, 0.0005123814917169511]
```

Jaybird65의 머릿속에 인지부조화의 폭풍우가 몰아치고 있다. 그는 Hunt78을 좋아하지만 의견이 다르다. 그의 정신적 폭풍우는 더욱 거세지게 된다. Jaybird65와 Hunt78 사이에서 일어나는 이후의 트윗을 읽는다면, Jaybird65의 감정을 상처받게 할 몇 가지 놀라운 사실을 발견할 수 있을 것이다.

Jaybird65와 Hunt78은 분명 서로를 알고 있다.

- 각각의 트위터 계정을 방문해 보면, 둘 다 사냥꾼임을 알 수 있다.
- 둘 다 강경파 공화당원임을 알 수 있다.

Jaybird65의 초기 트윗은 뉴욕 타임즈 기사에 대한 반응으로 기후 변화가 지구를 파괴하고 있다고 언급했다.

Jaybird65는 꽤 혼란스럽다. 그는 Hunt78이 자신과 같이 공화당원이며, 또한 사냥꾼이라는 것을 알 수 있다. 그렇다면 Hunt78이 기후 변화를 믿는다는 것은 어떻게 이해해야 할까?

이 트위터 스레드는 엄청난 수의 격렬한 트윗으로 이어진다.

하지만 가짜 뉴스 토론의 근원은 뉴스에 대한 감정적인 반응에 있다는 것을 알 수 있다. 기후 변화에 대한 합리적인 접근은 간단히 다음과 같을 것이다.

- 원인이 무엇이든 간에, 기후는 변화하고 있다.
- 인류를 변화시키기 위해 경제를 망가뜨릴 필요는 없다.
- 전기 자동차를 계속해서 만들고, 대도시에서 보다 많은 보행 공간을 마련하며, 더 나은 농업 습관을 가져야 한다. 수익을 창출할 가장 가능성이 높은 새로운 방식으로 사업을 진행하기만 하면 된다.

그러나 감정이 인간에게 미치는 영향은 강하다.

뉴스에서 감정적인 반응과 합리적인 반응으로 이어지는 과정을 표현해 보자.

13.1.1.2 가짜 뉴스의 행동적 표현

가짜 뉴스는 감정적인 반응으로 시작되어 종종 개인 공격으로 이어진다.

그림 13.2는 인지부조화가 사고 흐름을 방해할 때 가짜 뉴스를 향한 감정적인 반응 3 단계를 나타낸다.

13.1.1.3 1단계: 수신된 뉴스

두 명 또는 그룹의 사람들은 각각의 미디어를 통해 얻은 뉴스에 반응한다. 페이스북, 트위터, 다른 소셜 미디어, TV, 라디오, 웹사이트. 각 정보 소스는 편향된 의견을 포함하고 있다.

13.1.1.4 2단계: 합의

두 명 또는 그룹의 사람들은 동의하거나 동의하지 않을 수 있다. 만약 동의하지 않는다면, 우리는 3단계에 진입하게 되며, 그동안 충돌이 격화될 수 있다.

만약 그들이 동의한다면, 합의는 열이 쌓이는 것을 막고, 그 소식은 진짜 뉴스로 받아들여진다. 그러나, 모든 당사자들이 받고 있는 그 소식이 가짜가 아니라고 믿는다고 해서 그게 가짜가 아닌 것은 아니다. 다음은 가짜가 아닌 소식으로 라벨링된 소식이 가짜일 수 있는 몇 가지 내용이다.

- 12세기 초, 유럽 대부분의 사람들은 지구가 우주의 중심이고 태양계가 지구를 중심으로 회전한다고 동의했다.
- 1900년, 대부분의 사람들은 대양을 넘어 날아다니는 비행기 같은 것은 절대 없을 것이라고 믿었다.
- 2020년 1월, 대부분의 유럽인들은 COVID-19가 중국에만 영향을 주는 바이러스로, 세계적인 팬데믹이 아니라고 믿었다.

중요한 것은 두 당사자나 전체 사회 간의 합의가 들어오는 소식이 진실이거나 거짓이라는 것을 의미하지 않는다는 것이다. 만약 두 당사자가 의견이 다르다면, 이는 충돌로 이어질 것이다.

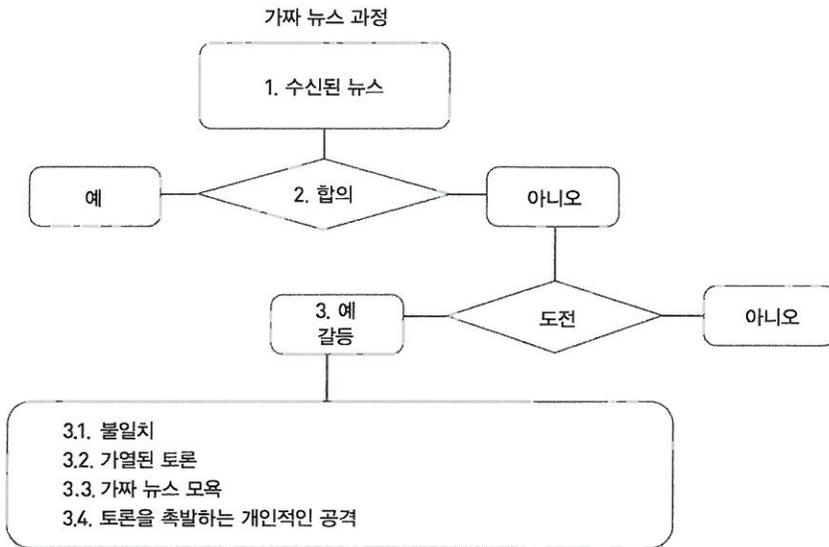


그림 13.2 뉴스에서 가짜 뉴스 갈등까지 이어지는 경로의 표현

소셜 미디어에서 회원들은 일반적으로 같은 생각을 가진 사람들과 모이게 되고, 어떤 일이 있어도 자신의 생각을 바꾸기 힘들어한다. 이는 사람이 트윗에서 표현한 의견을 종종 고수하며, 누군가가 그들의 메시지에 도전하자마자 충돌이 격화된다는 것을 보여준다.

13.1.1.5 3단계: 갈등

그림 13.2와 같이 가짜 뉴스 갈등은 네 가지 단계로 나눌 수 있다.

- 3.1: 갈등은 의견 충돌로 시작된다. 각 당사자는 트위터나 페이스북 또는 다른 플랫폼에 메시지를 게시할 것이다. 몇 차례 교류한 후, 갈등은 양 당사자가 해당 주제에 관심이 없어져서 식을 수도 있다.
- 3.2: Jaybird65와 Hunt78 사이의 기후 변화 논의로 돌아가보면, 상황이 심각해질 수 있다는 것을 알 수 있다. 대화가 고조된다!
- 3.3: 어느 순간, 한 당사자의 주장은 반드시 가짜 뉴스로 전락한다. Jaybird65는 화를 내며 여러 개의 트윗에서 인간의 기후 변화가 가짜 뉴스라고 주장할 것이다. Hunt78는 화를 내며 인간이 기후 변화에 기여하는 것을 부인하는 것이 가짜 뉴스라고 말할 것이다.
- 3.4: 이러한 토론은 종종 개인적인 공격으로 끝난다. 가끔은 우리가 어떻게 거기에 이르게 되었는지 알 수 없더라도, Godwin의 법칙이 대화에 들어온다. Godwin의 법칙은 대화 중 한 당사자가 상대방을 설명하는 최악의 비유를 찾을 것이라고 주장한다. 때로는 “당신 자유주의자들은 기후 변화

로 우리 경제를 무너뜨리려고 하는 히틀러 같은 사람들이다”라는 메시지로 나타날 수 있다. 이러한 유형의 메시지는 트위터, 페이스북 및 기타 플랫폼에서 볼 수 있다. 심지어 기후 변화에 관한 대통령 연설 중 실시간 채팅에도 나타날 수 있다.

이러한 토론에 합리적인 접근 방식이 있어 양측을 달래고 중간 지점에서 합의를 이루어 전진할 수 있는 방법이 있을까?

트랜스포머와 휴리스틱을 활용하여 합리적인 접근 방식을 만들어 보려고 해보자.

트랜스포머는 역사상 가장 강력한 NLP 도구이다. 이번 챕터에서는 가짜 뉴스로 인한 갈등에 참여하는 두 당사자를 감정적인 수준에서 합리적인 수준으로 이끌 수 있는 방법을 먼저 정의한다.

그런 다음 트랜스포머 도구와 휴리스틱을 사용할 것이다. 총기 통제와 전 대통령인 트럼프의 코로나 바이러스-19 대유행 중 트윗을 대상으로 트랜스포머 샘플을 실행할 것이다. 또한 기존 함수로 구현할 수 있는 휴리스틱에 대해서도 설명할 것이다.

이러한 트랜스포머 NLP 작업이나 다른 작업들을 의사 결정에 활용해 볼 수 있을 것이다. 로드맵과 방법론을 활용해 선생님들, 부모님들, 친구들, 동료들, 진실을 찾고자 하는 누구에게든 도움이 될 수 있다.

트랜스포머를 사용해 가짜 뉴스에 합리적으로 접근하기 위한 로드맵부터 시작해 보자.

13.2.1 가짜 뉴스 해결 로드맵 정의하기

그림 13.3은 가짜 뉴스를 합리적으로 분석하는 과정을 정의한다. 이 과정에는 트랜스포머 NLP 작업과 전통적인 기능이 포함된다.

합리적인 가짜 뉴스 해결 과정 로드맵

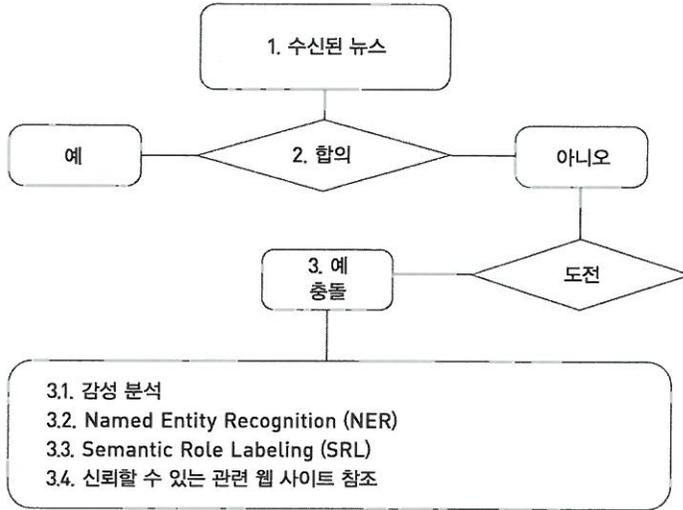


그림 13.3 가짜 뉴스에 대한 감정적인 반응에서 이성적인 표현으로 전환하는 과정

우리는 합리적인 사고과정이 거의 항상 감정적인 반응이 시작된 후에 시작된다는 것을 알 수 있다.

감정으로 반응하기 전에 합리적인 사고과정을 하도록 노력해야 한다. 이렇게 함으로써 토론이 중단되는 상황을 피할 수 있다. 그림 13.3의 단계 3에는 네 가지 도구가 포함되어 있다.

- **3.1:** 감성 분석을 통해 “감정적”으로 상위권인 긍정, 부정 단어를 분석한다. AllenNLP 자원을 사용하여 Fake_News.ipynb 노트북에서 RoBERTa 대형 트랜스포머를 실행할 것이다. 또한 AllenNLP의 시각 도구를 사용하여 키워드와 설명을 시각화할 것이다. 이전에 12장, 고객 감정을 감지해 예측하기에서 감성 분석에 대해 설명했었다.
- **3.2:** 소셜 미디어 메시지에서 개체를 추출하기 위한 **Named Entity Recognition (NER)**을 3.4단계에 사용한다. 11장, 데이터가 말하게 하기: 스토리, 질문, 답변에서 이에 관한 내용을 설명했다. 이 작업에는 허깅페이스의 BERT 트랜스포머 모델을 사용할 것이다. 또한 AllenNLP.org의 시각 도구를 사용하여 개체와 설명을 시각화할 것이다.
- **3.3:** 소셜 미디어 메시지에서 동사에 라벨을 지정하기 위한 **Semantic Role Labeling(SRL)**을 3.4 단계에 사용한다. 10장, BERT 기반 트랜스포머를 사용한 SRL에서 이에 대해 설명했다. Fake_News.ipynb에서 AllenNLP의 BERT 모델을 사용할 것이다. 또한 AllenNLP.org의 시각 도구를 사용하여 라벨링 작업의 출력을 시각화할 것이다.
- **3.4:** 신뢰할 수 있는 관련 웹 사이트를 보여줄 것이다. 일반적인 프로그래밍도 여전히 필요하다.

이제 총기 규제에 대해 살펴보자.

13.2.2 총기 규제 논쟁

미국 헌법의 제2 개정은 다음과 같은 권리를 주장한다.

잘 규율된 민병대는 자유로운 주(State)의 안보에 필수적이므로, 무기를 소장하고 휴대하는 인민의 권리는 침해될 수 없다.

수십 년 동안 미국은 이 문제에 대해 분열되어 왔다.

- 많은 사람들은 무기를 소지하는 것이 자신들의 권리이며, 총기 통제를 감수하고 싶지 않다고 주장한다. 그들은 무기를 소지하는 것이 폭력을 일으킨다는 주장은 가짜 뉴스라고 주장한다.
- 반면에, 많은 사람들은 총기를 소지하는 것이 위험하며, 총기 통제 없이 미국은 폭력적인 나라로 남을 것이라고 주장한다. 그들은 무기를 가지고 다니는 것이 위험하지 않다는 주장은 가짜 뉴스라고 주장한다.

감성 분석으로 시작해보자.

13.2.2.1 감성 분석

만약 트위터 트윗이나 페이스북 메시지, 연설 중에 유튜브 채팅이나 다른 어떤 소셜 미디어를 읽는다면, 당사자들이 격렬한 싸움을 벌이는 것을 볼 수 있을 것이다. TV 쇼는 필요하지 않다. 트윗 전투가 당사자들을 뒤죽박죽으로 만들 때, 팝콘을 먹으면서 그저 지켜보기만 해도 된다.

한쪽에서 온 트윗과 반대편 페이스북 메시지를 하나씩 살펴보자. 멤버들의 이름을 변경하고 텍스트의 욕설을 제거하도록 수정했다. 우선 총기 소지 찬성 의견을 살펴보자.

13.2.2.2 총기 소지 찬성 의견 분석

이 트윗은 한 사람의 솔직한 의견이다.

Afirst78: I have had rifles and guns for years and never had a problem. I raised my kids right so they have guns too and never hurt anything except rabbits.
(Afirst78: 나는 여러 해 동안 소총과 총을 가지고 있었고 문제가 없었습니다. 나는 내 아이들을 올바르게 키웠기 때문에 그들도 총을 가지고 있었고 토끼 외에는 아무것도 상처 입히지 않았습니다.)

Fake_News.ipynb에서 이것을 실행해 보자.

```
lecho '{"sentence": "I have had rifles and guns for years and never had  
a problem. I raised my kids right so they have guns too and never hurt  
anything except rabbits."}' | W  
allennlp predict https://storage.googleapis.com/allennlp-public-models/  
sst-roberta-large-2020.06.08.targz -
```

예측값은 긍정이다.

```
prediction: {"logits": [1.9383275508880615, -1.6191326379776], "probs":  
[0.9722791910171509, 0.02772079035639763]}
```

AllenNLP에서 결과를 시각화한다. 단순 기울기 시각화는 예측값에 대한 설명을 제공한다.

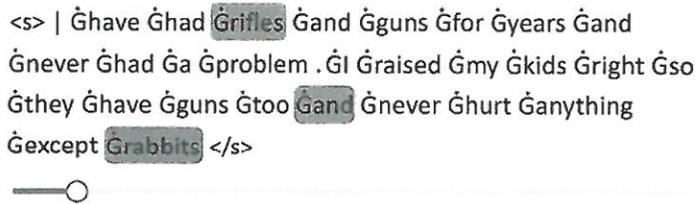


그림 13.4 입력 문장의 단순한 기울기 시각화

해당 설명은 Afirst78의 트윗에 대한 감성 분석 결과로, 소총(rifles) 그리고(and), 토끼(rabbit)를 강조한다.

NOTE

결과는 실행마다 또는 시간이 지남에 따라 달라질 수 있다. 이는 트랜스포머 모델이 지속적으로 학습되고 업데이트되기 때문이다. 그러나 이 챕터에서의 초점은 특정 결과가 아닌 프로세스에 있다.

각 단계마다 아이디어와 기능을 소개할 것이다. Fake_News_FUNCTION_1은 이 섹션에서 정의하게 되는 첫 번째 함수이다.

Fake_News_FUNCTION_1: 소총 + 그리고 + 토끼(rifles + and + rabbits)를 추출하고 추가 분석을 위해 기록해 두자. 이 예제에서 “소총(rifles)”은 “위험한(dangerous)” 것이 아님을 알 수 있다.

이제 NYS99의 총기를 통제해야 한다는 견해를 분석해 보겠다.

13.2.2.3 총기 규제 찬성 의견 분석

```
NYS99: I have heard gunshots all my life in my neighborhood,  
have lost many friends, and am afraid to go out at night.  
(NYS99: "나는 평생 동안 내 동네에서 총성을 항상 들었고, 많은 친구들을 잃었고, 밤에 나가기가 무서워.)
```

Fake_News.ipynb에서 먼저 분석을 실행해 보자.

```
!echo '{"sentence": "I have heard gunshots all my life in my neighborhood,  
have lost many friends, and am afraid to go out at night."}' | W  
allennlp predict https://storage.googleapis.com/allennlp-public-models/  
sst-roberta-large-2020.06.08.tar.gz -
```

예측값은 부정이다.

```
prediction: {"logits": [-1.3564586639404297, 0.5901418924331665],  
"probs": [0.12492450326681137, 0.8750754594802856]}
```

AllenNLP 온라인을 사용하여 키워드를 찾아보자. 샘플을 실행하고 부드러운 기울기 시각화에서 다음을 강조했다.

<s> Ġhave Ġheard Ġgunshots Ġall Ġmy Ġlife Ġin Ġmy Ġneighborhood , Ġhave Ġlost Ġmany Ġfriends , Ġand Ġam Ġafraid Ġto Ġgo Ġout Ġat Ġnight . </s>

그림 13.5 입력 문장의 부드러운 기울기 시각화

이 섹션의 기능 2에서는 “afraid”라는 키워드가 돋보인다. 이제 “두려움(afraid)”이 “총기(guns)”와 관련되어 있다는 것을 알 수 있다.

모델이 이러한 인지부조화를 해석하는 데 문제가 있다는 것을 알 수 있다. 따라서 인간의 비판적 사고는 여전히 필요하다.

Fake_News_FUNCTION_2: 두려움과 총기를 추출해 이후 분석에 사용할 것이다.

이제 두 함수를 나란히 놓으면, 두 정당이 서로 싸우는 이유를 명확히 이해할 수 있다.

- **Fake_News_FUNCTION_1:** 소총 + 그리고 + 토끼(rifles + and + rabbits)
Afirst78은 아마도 미국의 중서부 주에 살고 있을 것이다. 이러한 주 중 많은 곳들은 인구가 적고 매우 조용하며 범죄율이 낮다. Afirst78은 대도시를 여행해 본 적이 없을 수도 있으며 시골에서 조용한 삶의 즐거움을 즐기고 있다.
- **Fake_News_FUNCTION_2:** 두려움 + 총기(afraid + the topic guns)
NYS99는 아마도 대도시나 주요 미국 도시의 광역지역에 살고 있을 것이다. 범죄율은 종종 높고 폭력은 일상적인 현상이다. NYS99는 중서부 주에서 Afirst78의 생활 방식을 보지 못했을 수도 있다.

이 챗터에서 설명된 것과 같은 해결책을 사용해 위와 같은 의견의 대답을 줄일 수 있을 것이다.

더 나은 정보를 제공하는 일이 가짜 뉴스 전투를 줄이는 열쇠이다. 이제 개체명 인식을 적용해 보자.

13.2.2.4 개체명 인식(NER)

이번 챗터는 여러 가지 방법으로 트랜스포머를 사용하여 하나의 메시지를 다양한 관점에서 분석하는 방법을 보여준다. 챗터에 등장하는 방법들을 하나의 웹페이지로 구성하고 실제 운영환경에 적용해 보는 것도 좋다.

이제 트윗과 페이스북 메시지를 분석해 보자. 메시지에 개체가 없다는 것을 알 수 있지만, 프로그램은 그 사실을 알지 못한다. 첫 번째 메시지에 대해서만 분석을 수행해 볼 것이다.

먼저 허깅페이스의 transformers를 설치해야 한다.

```
!pip install -q transformers
from transformers import pipeline
from transformers import AutoTokenizer,
AutoModelForSequenceClassification,AutoModel
```

이제 다음 코드를 실행할 수 있다.

```
nlp_token_class = pipeline('ner')
nlp_token_class('I have had rifles and guns for years and never had a
problem. I raised my kids right so they have guns too and never hurt
anything except rabbits.')
```

개체명이 없으므로 결과가 생성되지 않는다. 그러나 이는 개체명 분석이 파이프라인에서 제외되어야 한다는 의미는 아니다. 다른 문장에는 해당 지역의 이름이 포함될 수 있으며, 해당 지역의 문화에 대한 단서를 제공하는 위치 정보를 제공할 수 있다.

계속 진행하기 전에 사용 중인 모델을 확인해 보자.

```
nlp_token_class.model.config
```

모델이 9개의 라벨과 1,024개의 피처를 사용하여 어텐션 층 구성한다는 것을 알 수 있다.

```
BertConfig {
  "_num_labels": 9,
  "architectures": [
    "BertForTokenClassification"
  ],
  "attention_probs_dropout_prob": 0.1,
  "directionality": "bidi",
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 1024,
  "id2label": {
    "0": "O",
    "1": "B-MISC",
    "2": "I-MISC",
    "3": "B-PER",
    "4": "I-PER",
    "5": "B-ORG",
    "6": "I-ORG",
    "7": "B-LOC",
    "8": "I-LOC"
  },
}
```

우리는 BERT 24-layer 트랜스포머 모델을 사용하고 있다. 만약 아키텍처를 탐색하고 싶다면 `nlp_token_class.model`을 실행해보자.

이제 메시지에 SRL을 실행해 볼 차례다.

13.2.2.5 SRL(의미역 결정)

이제 Fake_News.ipynb 파일을 노트북에 있는 순서대로 셀 단위로 실행할 것이다. 양쪽 시각을 모두 검토할 것이다.

먼저, 총기 소지 찬성 관점부터 시작해 보자.

13.2.2.6 총기 소지 찬성 의견의 SRL

Fake_News.ipynb에서 다음 셀을 먼저 실행하겠다.

```
!echo '{"sentence": "I have had rifles and guns for years and never had  
a problem. I raised my kids right so they have guns too and never hurt  
anything except rabbits."}' | W  
allennlp predict https://storage.googleapis.com/allennlp-public-models/  
bert-base-srl-2020.03.24.tar.gz -
```

매우 상세한 결과를 얻을 수 있으며, 라벨을 자세히 분석하는 데 유용할 것이다.

```
prediction: {"verbs": [{"verb": "had", "description": "[ARG0: I] have [V:  
had] [ARG1: rifles and guns] [ARGM-TMP: for years] and never had a problem  
...
```

이제 SRL 분석을 진행해 보겠다. 먼저 이 메시지에 대해 SRL 작업을 실행한다. 첫 번째 동사인 “had”는 Afirst78이 경험 많은 총 소유주임을 나타낸다.

had의 Frames :

I have had rifles and guns for years and never had a problem . I raised my kids
 right so they have guns too and never hurt anything except rabbits .

had의 Frames :

I have had rifles and guns for years and never had a problem . I raised my
 kids right so they have guns too and never hurt anything except rabbits .

그림 13.6 동사 "had"에 대한 SRL

첫 번째 프레임의 I + rifles and guns와 guns + for year는 Artist78이 수년간 총을 소지했던 경험을 말해준다.

두 번째 프레임에서는 I + never + had + a problem 단어들이 총기를 소지했음에도 문제가 없었다는 정보를 추가로 보여준다.

단어 raised는 Artist78의 부모로서의 경험을 보여준다.

raised의 Frames :

I have had rifles and guns for years and never had a
 problem . I raised my kids
 right
 so they have guns too and never hurt anything
 except rabbits

그림 13.7 raised에 대한 SRL 동사

다음 단어들이 총기 소지에 찬성하는 입장을 설명한다. 나의 아이들 + 총을 소지했었고 아무에게도 해를 끼친 적이 없다(my kids + ...have guns too and never hurt anything)

실행마다 또는 모델이 업데이트될 때마다 결과가 다를 수 있지만 과정은 동일하다.

여기에서 찾은 내용을 일부 구문 분석과 함께 함수 컬렉션에 추가할 수 있다.

- Fake_news_FUNCTION_3: 나 + 소총과 총 + 수년 간(+ rifles and guns + for years)
- Fake_news_FUNCTION_4: 나의 아이들 + 소지하고 있고 아무에게도 해를 끼친 적이 없다.
(my kids + have and never hurts anything)

이제 총기 소지 반대 메시지를 탐구해 보자.

13.2.2.7 총기 규제 SRL

먼저 Fake_News.ipynb에서 페이스북 메시지를 실행할 것이다. 노트북에서 생성된 순서대로 셀을 하나씩 실행해 보자.

```
!echo '{"sentence": "I have heard gunshots all my life in my neighborhood, have lost many friends, and am afraid to go out at night."}' | \W
allennlp predict https://storage.googleapis.com/allennlp-public-models/bert-base-srl-2020.03.24.tar.gz -
```

핵심 동사들이 자세히 라벨링되어 있는 것을 확인할 수 있다.

```
prediction: {"verbs": [{"verb": "heard", "description": "[ARG0: I] have [V: heard] [ARG1: gunshots all my life in my neighborhood]"
```

이제 AllenNLP(HanLP)로 이동한 다음 SRL 섹션을 실행해 보자. 문장을 입력하고 트랜스포머 모델을 실행할 것이다. 이 메시지의 동사 heard를 통해 심각한 현실을 확인할 수 있다.

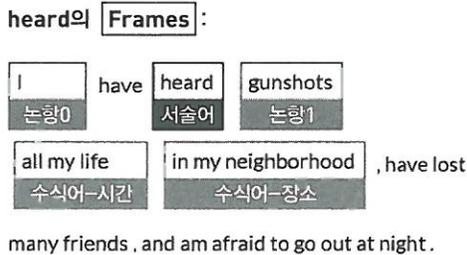


그림 13.5 동사 heard에 대한 SRL

13.2.2.9 총기 소지 찬성 의견과 총기 규제 찬성 의견

총기 소지 찬성 주장은 미국의 주요 도시에서 무슨 일이 일어나고 있는지에 대한 정보가 부족하다는 것을 보여준다.

- 총기 소지 찬성: Fake_News_FUNCTION_1: 절대 + 문제 + 총(never + problems + guns)
- 총기 소지 찬성: Fake_News_FUNCTION_3: 나 + 소총과 총 + 년(I + riffles and guns + years)
- 총기 소지 찬성: Fake_News_FUNCTION_4: 나의 아이들 + 총을 소지하다 + 아무에게도 해를 끼친 적이 없다(my kids + have guns + never hurt anything)

총기 규제 찬성 주장은 미드웨스트에 많은 얼마나 많은 조용한 지역들이 있는지에 대한 정보가 부족하다는 것을 보여준다.

- 총기 규제 찬성: Fake_News_FUNCTION_2: 듣다 + 두려운 + 총(heard + afraid + guns)
- 총기 규제 찬성: Fake_News_FUNCTION_5: 듣다 + 총성 + 평생동안(heard + gunshots + all my life)
- 총기 규제 찬성: Fake_News_FUNCTION_6: 잃다 + 많은 + 친구들(lost + many + friends)

서로 반대 의견을 가진 사람들에게 이러한 사실을 알려주는 데 각 함수를 활용할 수 있다.

예를 들어, FUNCTION1을 의사 코드로 표현해 보자면,

FUNCTION 1:

FUNCTION 2, 5, 6의 키워드들을 종합해 "afraid guns lost many friends gunshots"라는 내용으로 구글 검색을 수행한다.

이 프로세스의 목표는 다음과 같다.

- 먼저, 메시지를 해석하기 위해 트랜스포머 모델을 실행한다. NLP 트랜스포머는 계산기와 같은 역할을 한다. 결괏값을 해석하는 것은 사람의 몫이다!
- 그런 다음 숙련된 NLP 사용자에게 능동적으로 정보를 검색하고 더 잘 읽도록 요청한다.

트랜스포머 모델은 사용자가 메시지를 더 깊이 이해하는 데 도움을 준다. 하지만 사용자에게 특정한 의견을 강요하지는 않는다.

각 함수의 결과를 적절히 파싱해 주어야 할 것이다. 수백 개가 넘는 메시지를 처리할 수 있도록 적절히 자동화해야 한다.

구글 검색 결과는 달라질 수 있지만, 가장 상단의 링크들은 총기 소지자들에게 다양한 반대 의견의 근거가 있다는 것을 보여줄 수 있을 것이다.

www.amnesty.org > arms-control ▾ Traduire cette page

Gun violence - key facts | Amnesty International

When people are **afraid** of **gun** violence, this can also have a negative impact on people's right to ... How **many** people are injured by **gunshots** worldwide? ... We created March For Our Lives because our **friends** who **lost** their lives would have ...

everytownresearch.org > impact-gun... ▾ Traduire cette page

The Impact of Gun Violence on Children and Teens ...

29 mai 2019 - They are also harmed when a **friend** or family member is killed with a **gun**, when ... **Gun** homicides, non-fatal **shootings**, and exposure to **gun** violence stunt ... **worried** some or **a lot** of the time that they might get killed or die.³⁵.

www.hsph.harvard.edu > magazine ▾ Traduire cette page

Guns & Suicide | Harvard Public Health Magazine | Harvard ...

Gun owners and their families are **much** more likely to kill themselves than are ... Zachary may have been **afraid** of **losing** his commercial driver's license, a great ... In public health lingo, these potentially lifesaving **friends** and colleagues are ... other natural allies such as hunting groups, **shooting** clubs and **gun** rights groups.

www.pbs.org > extra > student-voices ▾ Traduire cette page

How teens want to solve America's school shooting problem ...

14 févr. 2019 - It's not having students practice lock-downs out of **fear** that an attack like ... The problem America has is that we give everyone a **gun** without **any** mental health testing. ... After the Florida school **shooting** my **friends** and I were having a ... We can't have more innocent lives **lost** just because of one person's ...

그림 13.10 총과 폭력에 관한 검색 결과

다음 의사 코드로 총기 규제에 찬성하는 사람을 찾는다고 상상해 보자.

FUNCTION 2:

FUNCTION 1, 3, 4의 키워드들을 종합해 "never problem guns for years kids never hurt anything"이라는 내용으로 구글 검색을 수행한다.

구글 검색은 총기 소지에 찬성하는 사람들의 의견을 보여주지는 않았다. 검색 결과는 흥미롭게도 대부분 교육적이고 중립적인 것들이었다.

kidshealth.org > parents > gun-safety ▾ Traduire cette page

Gun Safety - Kids Health

But every year, **guns** are used to kill or **injure** thousands of Americans. ... Even if you **have** talked to them many times about **gun safety**, they can't truly understand how ... Teens should **never** be able to get to a **gun** and bullets without an adult being there. ... Is there a **gun** or **anything** else dangerous he might get into?

www.healthychildren.org > Pages ▾ Traduire cette page

Guns in the Home - HealthyChildren.org

12 juin 2020 - **Did** you know that roughly a third of U S homes with **children have guns**? ... Parents can reduce the chances of **children being injured**, however, by ... about pets, allergies, supervision and other **safety issues** before your **child** visits ... Remind your **kids** that if they **ever** come across a **gun**, they must stay away ...

그림 13.11 총기 안전에 관한 검색 결과

네이버의 서점, 잡지 및 기타 교육 자료에서도 자동 검색을 실행할 수 있을 것이다.

가장 중요한 것은 의견이 다른 사람들이 싸움을 벌이지 않고 서로 대화하는 것이다. 서로를 이해하는 것이 양측에 공감을 발전시키는 가장 좋은 방법이다.

소셜미디어의 내용은 그럴듯하다. 하지만, 제3자를 무조건적으로 신뢰하지 말자. 트랜스포머로 메시지를 분석하며 주도적으로 생각하자!

이 주제에 대해서 총기 소지에 관한 안전 지침을 따르는 방식으로 합의할 수 있을 것이다. 예를 들어, 집에 총기를 두지 않거나 안전하게 보관하는 것으로 합의할 수 있으며, 이렇게 함으로써 어린이가 총기에 접근하지 못하도록 할 수 있다.

이제 COVID-19와 전 대통령 트럼프의 트윗에 대해 이야기해 보겠다.

13.2.3 코로나19와 전 대통령 트럼프의 트윗

정치적인 의견과는 무관하게 도널드 트럼프가 말한 내용들이다. 도널드 트럼프에 대해 이야기되는 내용이 너무 많아서 모든 정보를 분석하는 데에는 책 한 권이 필요할 정도다. 이 책은 정치적 내용과 무관한 기술적인 책이므로 트윗을 과학적으로 분석할 것이다.

우리는 총기 규제 찬성 섹션에서 가짜 뉴스에 대한 교육적으로 접근하는 방식을 설명했다. 여기서 같은 작업을 반복하지는 않을 것이다.

대신, AllenNLP의 BERT 모델을 사용하여 Fake_News.ipynb 노트북에서 SRL 분석을 수행했다.

이 섹션에서는 가짜 뉴스의 논리에 초점을 맞출 것이다. 이를 위해, BERT 모델을 SRL에 적용하고 그 결과를 AllenNLP의 웹사이트에서 시각화할 것이다.

이제 몇 가지 대통령 트윗을 살펴보겠다.

13.2.3.1 SRL

SRL은 우리 모두에게 훌륭한 교육 도구다. 우리는 종종 트윗을 수동적으로 읽거나 다른 사람들의 의견을 받아들인다. SRL을 사용해 메시지를 분해하면 가짜 정보와 정확한 정보를 구별하기 위한 소셜 미디어 분석 기술을 개발할 수 있다.

교실에서 교육 목적으로 SRL 트랜스포머를 사용하는 것을 추천한다. 학생들은 트랜스포머를 활용해 트윗의 각 동사 및 연결된 단어들에 대해 분석할 수 있다. 이는 어린 세대가 소셜 미디어에서 능동적인 독자가 되는 데 도움이 될 수 있다.

먼저 비교적 다툼이 없는 트윗을 분석한 후 갈등을 겪고 있는 트윗들을 분석해 보겠다.

이 책을 쓰는 동안 발견된 가장 최신 트윗을 분석해 보겠다. “검은 미국인”이라는 사람을 언급하는 트윗에서 해당 인물의 이름을 제외하고 전 대통령의 일부 텍스트를 다시 표현했다.

X is a great American, is hospitalized with coronavirus, and has requested prayer. Would you join me in praying for him today, as well as all those who are suffering from COVID-19?

(X는 위대한 미국인입니다. 그리고 코로나 바이러스로 병원에 입원해있습니다. 그리고 기도를 요청했습니다. COVID-19로 고통받는 사람들과 그를 위해 함께 기도해 주시겠습니까?⁴²)

AllenNLP의 SRL 섹션으로 이동하여 문장을 실행하고 결과를 확인해 보겠다.

동사 hospitalized는 의견과 무관한 실제 사실과 관련된 것만 보여준다.

42 역주. 여기서 has requested prayer는 ‘기도를 요청했다’ 또는 ‘기도를 요청 받았다’로 해석 가능하다.

hospitalized의 Frames :

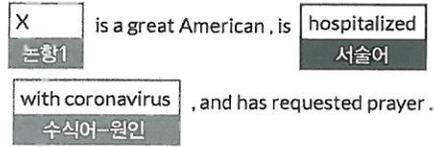


그림 13.12 동사 "hospitalized"에 대한 SRL 논항들

메시지는 단순하다. X + hospitalized + with coronavirus.

동사 requested는 메시지가 정치적인 성향을 띠고 있다는 것을 보여준다.

requested의 Frames :

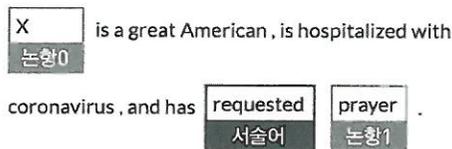


그림 13.13 동사 requested에 대한 SRL 논항들

그 사람이 전 대통령에게 기도를 요청했는지 아니면 강조하기 위한 것인지는 확실하지 않다.

한 가지 좋은 방법은 HTML 페이지를 표시해 사용자들에게 의견을 묻는 것이다. 예를 들어, 사용자들에게 SRL 작업의 결과를 제시하고 다음 두 가지 질문에 답하도록 요청할 수 있다.

트럼프 전 대통령이 기도해 달라는 요청을 받았을까요? 아니면 정치적 목표를 위해 트윗으로 요청을 했을까요?

전 대통령 트럼프가 자신이 간접적으로 X를 위해 기도하도록 요청받았다고 주장하는 것이 가짜 뉴스인가요, 아닌가요?

사람들은 이에 대해 스스로 생각해 보고 판단할 수 있을 것이다.

이제 트위터에서 차단된 트윗을 살펴보겠다. 이름을 제외하고 일부 표현을 순화했지만, 인터넷에서 실행하고 결과를 시각화하면 몇 가지 유의미한 SRL 결과들을 얻을 수 있다.

다음은 일부 수정된 내용의 트윗이다.

These thugs are dishonoring the memory of X.
When the looting starts, actions must be taken.
(이 폭력배들은 X에 대한 기억을 더럽히고 있습니다.
약탈이 시작되면 조치를 취해야 합니다.)

원본 트윗의 주요 내용들을 순화했음에도 불구하고, SRL 분석 결과는 부정적인 내용들을 보여준다.



그림 13.14 동사 dishonoring에 대한 SRL 논항들

SRL 분석을 통해 “폭력배(thugs)”와 “기억(memory)” 그리고 “약탈(looting)”이라는 단어들은 서로 무관하다는 것을 알 수 있다.

이러한 SRL 논항들이 왜 서로 무관한지 생각해 보는 것이 좋은 연습이 될 것이다.

이 연습을 통해 트랜스포머 모델 사용자들은 제시된 어떤 주제에 대해 비판적인 시각을 갖도록 하는 SRL 기술을 습득할 수 있을 것이다.

비판적 사고는 가짜 뉴스 유행을 막는 가장 좋은 방법이다.

지금까지, 트랜스포머, 휴리스틱 및 교육적인 웹사이트와 함께 가짜 뉴스에 대한 합리적인 접근 방식을 살펴보았다. 가짜 뉴스 토론에서의 많은 논쟁은 감정적이고 비합리적인 반응으로 귀결된다.

찬반 여론이 존재하는 세상에서는, 진실이 무엇인지에 대해 서로 동의하지 않기 때문에 완전히 객관적인 가짜 뉴스 감지를 위한 트랜스포머 모델은 없을 것이다. 한쪽은 트랜스포머 모델의 출력에 동의할 지라도, 다른 한쪽은 그 모델이 편향되어 있다고 말할 것이다.

가장 좋은 접근 방식은 다른 사람들의 의견에 귀를 기울이고 감정을 가라앉히는 것이다.

13.3 마치고 전에

이 장에서는 실제로 존재하지 않는 마법같은 트랜스포머 모델을 찾는 것보다는 다양한 트랜스포머 모델들을 문제에 적용하는 것에 초점을 맞췄다.

즉, NLP 문제를 해결하기 위한 두 가지 옵션이 있을 것이다. 새로운 트랜스포머 모델을 찾거나 신뢰할 수 있고 지속적으로 사용할 수 있는 트랜스포머 모델 활용법을 찾는 것이다.

이제 이 장을 마무리하고 트랜스포머 모델의 해석으로 넘어가도록 하겠다.

13.4 정리하기

가짜 뉴스는 인간으로서 우리의 감정적인 역사 깊숙이에서 시작된다. 어떤 사건이 발생하면 감정은 우리가 상황에 신속하게 대응할 수 있도록 돕기 위해 우리를 지배한다. 위협을 받을 때는 감정적으로 강하게 반응하도록 설계되어 있다.

가짜 뉴스는 강한 감정적 반응을 일으킨다. 우리는 이러한 뉴스가 일시적으로나 영구적으로 삶을 파괴할지도 모른다는 두려움을 가지고 있다. 예를 들어, 많은 사람들은 기후 변화가 지구상의 인간을 멸종시킬 수 있다고 믿는다. 다른 사람들은 기후 변화에 불필요하게 과하게 대응한다면 우리의 경제를 파괴하고 사회를 붕괴시킬 수 있다고 주장한다. 일부 사람들은 총기가 위험하다고 믿는다. 또 다른 사람들은 미국 수정 헌법 제2조가 총기 소지 권리를 부여한다고 이야기한다.

우리는 COVID-19, 전 대통령 트럼프, 기후 변화를 둘러싼 다른 격렬한 갈등을 살펴봤다. 각각의 경우들에 대해서 감정적인 반응들이 갈등으로 격화하는 것을 확인했다.

그런 다음 가짜 뉴스에 대한 감정적 인식을 합리적인 수준으로 끌어올리기 위한 로드맵을 설계했다. 트랜스포머를 활용한 NLP 도구들을 사용해 트윗, 페이스북 메시지 및 기타 미디어에서 핵심 정보를 찾을 수 있다는 것을 보였다.

누군가에게는 실제, 누군가에게는 가짜로 인식되는 뉴스에서 그들의 근거를 확인해 보았다. 일반적인 프로그래밍도 여전히 도움이 된다.

이제 여러분들은 트랜스포머 모델과 NLP 도구들 그리고 샘플 데이터들을 확인해 보았다.

인공지능을 인류의 이익을 위해 활용할 수 있다. 여러분은 이러한 트랜스포머 도구와 아이디어를 가져가서 모두를 위한 더 나은 세상을 만들기 위해 사용할 수 있다.

트랜스포머를 이해하는 좋은 방법은 내부 과정을 시각화하는 것이다. 다음 장인 **블랙박스 트랜스포머 모델 해석하기**에서는 트랜스포머가 어떻게 점진적으로 시퀀스의 의미를 만들어내는지 분석할 것이다.

13.5 문제

01. 가짜 뉴스로 라벨링된 뉴스는 항상 가짜이다. 참 / 거짓
02. 모두가 동의하는 뉴스는 항상 정확하다. 참 / 거짓
03. 트랜스포머를 트윗 감성 분석에 사용할 수 있다. 참 / 거짓
04. 페이스북 메시지에서 DistilBERT 모델을 사용해 주요 개체명을 추출할 수 있다. 참 / 거짓
05. 유튜브 채팅에서 주요 동사는 BERT 기반 모델을 사용하여 의미역 결정을 수행함으로써 식별할 수 있다. 참 / 거짓
06. 감정적 반응은 가짜 뉴스에 대한 자연스러운 첫 번째 반응이다. 참 / 거짓
07. 가짜 뉴스에 대한 합리적인 접근은 자신의 입장을 명확히 하는 데 도움이 된다. 참 / 거짓
08. 신뢰할 수 있는 웹사이트와 트랜스포머를 연결하면 어떤 뉴스가 가짜인지 이해하는 데 도움이 된다. 참 / 거짓
09. 트랜스포머를 사용해 신뢰할만한 웹사이트들을 요약해 가짜 뉴스라고 알려진 주제들을 이해할 수 있다. 참 / 거짓
10. 모두의 이익을 위해 인공지능을 사용한다면 세상을 바꿀 수 있다. 참 / 거짓

13.6 참고 문헌

- Daniel Kahneman, 2013, Thinking, Fast and Slow
- 허깅페이스 파이프라인: https://huggingface.co/transformers/main_classes/pipelines.html
- 앨런 인공지능 연구소: <https://allennlp.org/>

14장

블랙박스 트랜스포머 모델 해석하기

- 14.1 BertViz로 트랜스포머 시각화하기
- 14.2 LIT
- 14.3 딕셔너리 러닝을 활용한 트랜스포머 시각화
- 14.4 내부를 볼 수 없는 모델 분석하기
- 14.5 정리하기
- 14.6 문제
- 14.7 참고 문헌

수 백만에서 수 십억 개의 파라미터를 가진 트랜스포머는 아무도 해석할 수 없는 거대한 블랙박스처럼 보인다. 따라서 많은 개발자들이 이 놀라운 성능의 트랜스포머를 사용하다가 낙담하기도 했다. 하지만 최근에는 혁신적인 첨단 도구들로 이 문제를 해결하기 시작했다.

설명 가능한 모든 AI 방법론과 알고리즘을 설명하는 것은 이 책의 범위를 벗어난다. 대신 이번 챕터에서는, 트랜스포머를 사용하는 사람에게 인사이트를 제공하는, 바로 사용할 수 있는 시각화 인터페이스를 다룰 것이다.

이번 장에서는 먼저 제시 빅(Jesse Vig)의 BertViz를 설치하고 실행한다. Jesse는 훌륭한 시각화 인터페이스를 만들어서, BERT 트랜스포머 어텐션 헤드의 동작을 확인할 수 있도록 했다. BertViz는 BERT 모델과 상호 작용할 수 있도록 잘 설계된 인터페이스를 제공한다.

다음으로는 LIT(Language Interpretability Tool)를 사용해서 트랜스포머의 행동을 시각화할 것이다. LIT는 PCA나 UMAP을 활용하여 트랜스포머의 예측을 시각화하는 도구이다. PCA와 UMAP 또한 살펴보겠다.

그리고 나서, 딕셔너리 러닝(dictionary learning)을 수행하며 BERT의 층을 통과하는 트랜스포머의 여정을 시각화할 것이다. LIME(Local Interpretable Model-agnostic Explanations)의 기능을 사용하면 트랜스포머가 어떻게 언어를 이해하는지 시각화할 수 있다. 트랜스포머가 단어부터 시작해서, 문맥 속의 단어를 배우고, 마지막으로 장기 의존성(long-range dependency)을 배우는 것을 확인할 것이다.

장을 마치면 상호 작용을 통해 트랜스포머의 활동을 시각화할 수 있을 것이다. 딕셔너리 러닝을 시각화하는 방법이나, BertViz, LIT에도 아직 한계는 있다. 하지만 비록 초기 단계의 도구일지라도 트랜스포머를 이해하는데 도움이 될 것이다.

이 장에서는 다음 주제들을 다룬다.

- BertViz의 설치와 실행
- BertViz의 상호 작용 인터페이스 실행
- 프로빙(probing) 방식과 논-프로빙(non-probing) 방식의 차이
- 주성분 분석(Principal Component Analysis, PCA) 떠올리기
- LIT로 트랜스포머 출력 분석하기
- LIME에 대한 소개
- 딕셔너리 러닝으로 트랜스포머 시각화하기
- 단어 단위의 다의어 명확화
- 단기, 중기, 장기 의존성 시각화
- 트랜스포머 핵심 요소 시각화

먼저 BertViz를 설치하고 사용해 보겠다.

14.1

BertViz로 트랜스포머 시각화하기

제시 빅은 2019년에 작성한 논문 「A Multiscale Visualization of Attention in the Transformer Model」에서 트랜스포머의 효율성을 인정함과 동시에, 어텐션 메커니즘을 해석하는 것이 어렵다는 것을 설명한다. 이 논문은 시각화 도구인 BertViz의 수행 과정을 묘사하고 있다.

BertViz는 어텐션 헤드의 동작을 시각화하고 트랜스포머 모델의 행동을 해석한다.

BertViz는 BERT와 GPT-3를 해석하기 위해 설계되었다. 이번 섹션에서는 BERT 모델의 동작을 시각화할 것이다.

이제 BertViz를 설치하고 실행하자.

14.1.1 BertViz 실행하기

다섯 단계만 수행하면 트랜스포머의 어텐션 헤드를 시각화하고 상호 작용할 수 있다. 깃허브의 Chapter14 경로에 있는 BertViz.ipynb 노트북을 열어보자.

첫 번째 단계로, BertViz와 의존성(requirements)을 설치할 것이다.

14.1.1.1 1단계: BertViz를 설치하고, 모듈 불러오기

노트북에서 BertViz, 허깅페이스 트랜스포머, 그 외 기본적인 의존성을 불러온다.

```
!pip install bertViz
from bertViz import head_view, model_view
from transformers import BertTokenizer, BertModel
```

헤드 시각화(head view)와 모델 시각화(model view) 라이브러리를 불러왔다. 이제 BERT 모델과 토큰라이저를 불러올 것이다.

14.1.1.2 2단계: 모델을 불러오고 어텐션 가져오기

BertViz는 BERT, GPT-2, RoBERT 등의 모델을 지원한다. BertViz의 깃허브에서 더 많은 정보를 확인할 수 있다. (<https://github.com/jessevig/BertViz>)

이번 섹션에서는, bert-base-uncased 모델과 사전 학습된 토큰라이저를 사용한다.

```
# 모델을 불러오고 어텐션 가져오기
model_version = 'bert-base-uncased'
do_lower_case = True
model = BertModel.from_pretrained(model_version, output_attentions=True)
tokenizer = BertTokenizer.from_pretrained(model_version, do_lower_case=do_lower_case)
```

이제 두 문장을 입력할 것이다. 모델의 동작을 분석하기 위해 여러 시퀀스를 시도해 볼 수 있다. sentence_b_start는 5단계: Model view에서 사용하기 위한 변수이다.

```
sentence_a = "A lot of people like animals so they adopt cats"
#많은 사람이 동물을 좋아해서 고양이를 입양한다.
sentence_b = "A lot of people like animals so they adopt dogs"
#많은 사람이 동물을 좋아해서 강아지를 입양한다.
inputs = tokenizer.encode_plus(sentence_a, sentence_b, return_tensors='pt', add_special_tokens=True)
token_type_ids = inputs['token_type_ids']
input_ids = inputs['input_ids']
attention = model(input_ids, token_type_ids=token_type_ids)[-1] sentence_b_start = token_type_ids[0].tolist().index(1) input_id_list = input_ids[0].tolist()
# 0번째 배치
tokens = tokenizer.convert_ids_to_tokens(input_id_list)
```

여기까지가 전부이다! 시각화 인터페이스를 사용할 준비가 모두 끝났다.

14.1.1.3 3단계: 헤드 시각화

마지막 한 줄을 더하면, 어텐션 헤드를 시각화할 수 있다.

```
head_view(attention, tokens)
```

각 층에 있는 12개의 어텐션 헤드 정보를 서로 다른 색으로 확인할 수 있다. 처음에는 그림 14.1처럼 0번 층이 표시된다.

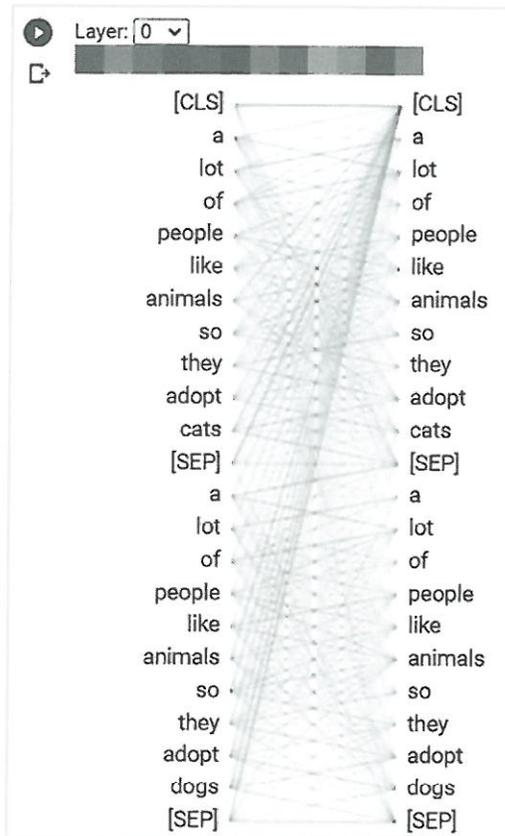


그림 14.1 어텐션 헤드 시각화

14.1.1.4 4단계: 어텐션 헤드를 선택하고 표시하기

토큰 정보 위에 있는 색상표는 해당 층의 어텐션 헤드를 나타낸다.

층 번호를 고르고 어텐션 헤드(색상표)를 눌러보자.

문장 속 토큰이 여러 토큰에 대한 어텐션으로 세분화된다. 이번 섹션에서는 트랜스포머의 헤드를 쉽게 설명하기 위해 '토큰'을 '단어'를 의미하는 포괄적 표현으로 사용하겠다.

'animal'이라는 단어를 선택하면, 그림 14.2와 같이 나타난다.

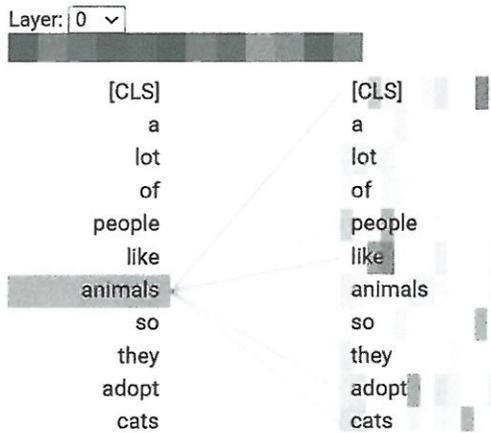


그림 14.2 층, 어텐션 헤드, 토큰을 선택

BertViz를 사용하여 animals와 다양한 단어 사이에 관계가 형성되었음을 확인했다. 결과가 이상해 보일 수 있으나, 이제 0번 층에 불과하므로 충분히 이해할 수 있다.

그림 14.3에서처럼, 1번 층에서는 좀 더 연관있는 단어가 나타난다.

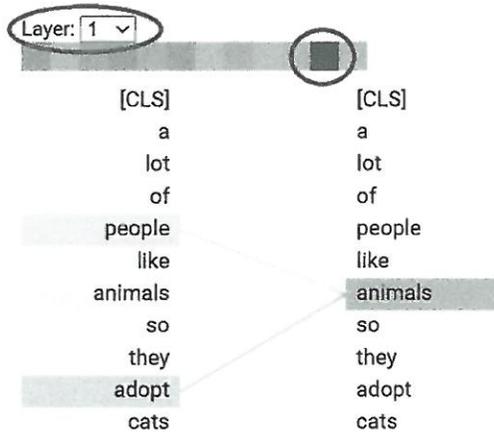


그림 14.3 1번 층에서, 11번째 어텐션 헤드를 시각화

11번째 어텐션 헤드는 animals, people, adopt와 관계를 형성했다.

cats를 선택하면, 그림 14.4처럼 흥미로운 결과를 볼 수 있다.



그림 14.4 cats와 다른 토큰과의 관계 시각화

cats와 animals 사이에 관계가 형성되어 있다. 모델이 고양이(cats)를 동물(animals)로 학습한 것이다.

문장을 바꾸고 층과 어텐션 헤드를 선택해 가며 트랜스포머가 어떻게 관계를 형성하는지 시각화해 볼 수 있다. 물론 한계도 있다. 관계정보를 보면 트랜스포머가 어떻게 잘 동작하는지 또는 왜 틀리는지 알 수 있다. 두 사례 모두 트랜스포머의 작동 방식을 설명하거나 층, 파라미터, 데이터가 많이 필요한 이 유를 설명하는데 유용한 정보이다.

이제 BertViz로 모델을 시각화하는 방법을 알아보자.

14.1.1.5 5단계: 모델 시각화

다음 한 줄로 BertViz에서 트랜스포머 모델을 시각화할 수 있다.

```
model_view(attention, tokens, sentence_b_start)
```

그림 14.5처럼, BertViz는 모든 층과 헤드를 하나의 창(view)으로 보여준다.

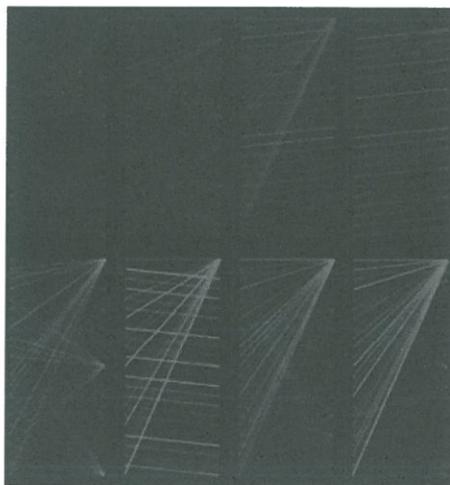


그림 14.5 BertViz의 모델 시각화

헤드 하나를 클릭하면, 단어-대-단어 또는 문장-대-문장 옵션에 따라 헤드를 시각화한다. 어텐션 헤드를 살펴보면 층을 거치면서 트랜스포머의 표현력이 점점 발전하는 것을 확인할 수 있다. 예를 들어, 그림 14.6은 첫 번째 층의 어텐션 헤드를 살펴보면 다음과 같다.

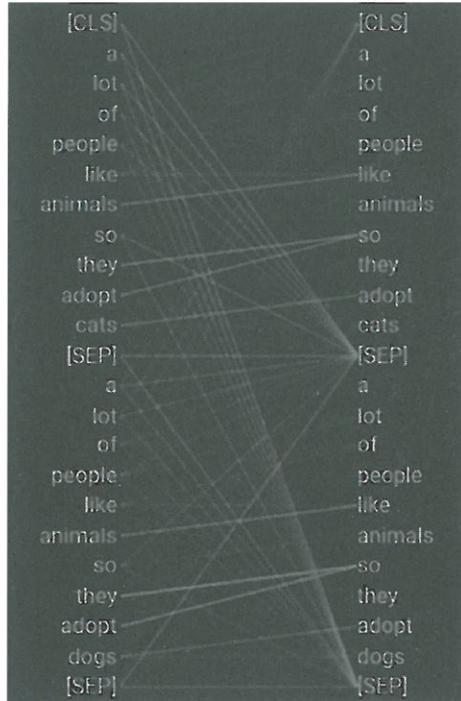


그림 14.6 낮은 층에서의 어텐션 헤드 동작

가끔씩 구분자 [SEP]와 단어 사이에 관계가 형성되는데 대부분 큰 의미는 없다. 때로는 특정 토큰이 모든 층의 모든 어텐션 헤드에서 활성화되지 않는 경우도 있으며, 트랜스포머의 학습 정도에 따라 해석의 품질이 제한적일 수 있다.

어떠한 경우에도, BertViz는 흥미롭고 교육적인 도구이자 유용한 트랜스포머 해석 도구이다.

이제 직관적인 LIT 도구를 실행해 보자.

14.2 LIT

LIT의 시각화 인터페이스로 모델이 잘못 처리하는 예제를 찾고, 비슷한 예제를 살펴보고, 컨텍스트에 따라 어떻게 달라지는지 확인하며 트랜스포머와 관련된 여러 언어 문제를 찾을 수 있다.

LIT는 BertViz처럼 단순히 어텐션 헤드의 활성도를 보여주는 것에 그치지 않는다. 대신, 문제가 발생한 이유를 분석하고 해결책을 찾도록 도와준다.

시각화 방법으로 UMAP(Uniform Manifold Approximation and Projection)과 PCA 중 하나를 선택하여 수행한다. PCA는 특정 방향과 크기로 더 많은 선형 투영(linear projection)을 수행하는 방법이며, UMAP은 투영을 여러 미니 클러스터로 세분화하는 방법이다. 모델의 출력을 분석할 때, 두 방법 모두 어떻게 분석할 것인지에 따라 의미가 달라진다. 두 가지를 모두 수행하며 동일 모델에 대한 다양한 관점을 살펴보는 것도 가능하다.

이번 섹션에서는 PCA를 선택하여 LIT를 수행할 것이다. PCA의 원리를 간단히 복습하며 시작하겠다.

14.2.1 PCA

PCA는 주어진 데이터를 더 높은 수준으로 표현한다.

지금 주방에 있다고 생각해 보자. 주방은 3차원의 좌표공간이다. 주방에 있는 모든 물건은 x , y , z 좌표로 표현할 수 있다.

요리를 하기 위해 재료를 모아서 조리대 위에 놓는다면, 조리대는 주방을 높은 수준으로 표현한 공간이 된다.

조리대 역시 좌표공간이며, 요리를 하기 위해 주방의 주요 특징을 조리대에 놓는 행위는 PCA를 수행한 것으로 볼 수 있다. 특정 요리를 위한 주요 성분(principal components)을 레시피에 맞게 배치했기 때문이다.

NLP에도 같은 개념을 적용할 수 있다. 예를 들어, 사전은 단어의 나열일 뿐이지만, 일부 단어가 모여 의미를 형성한다면 특정 시퀀스를 나타내는 주요 성분이 된다.

LIT에서 시퀀스의 PCA 표현은 트랜스포머를 시각화하는 데 도움을 준다.

NLP PCA 표현을 구하기 위한 주요 단계는 다음과 같다.

- **분산**: 데이터셋에서 단어의 분산, 빈도와 의미의 빈도 등.
- **공분산**: 단어와 데이터셋 내의 다른 단어 사이 관계에 대한 분산
- **고윳값과 고유 벡터**: 좌표를 사용하기 위해 공분산에 대한 방향과 크기를 표현할 수 있어야 한다. 고유 벡터는 방향을, 고윳값은 크기를 나타낸다.
- **데이터 도출**: 마지막 단계는 특징 벡터의 행을 데이터의 행과 곱하여 특징 벡터를 오리지널 데이터셋에 반영하는 것이다.
- **표시할 데이터** = 특징 벡터의 행 * 데이터의 행

PCA 투영으로 분석할 데이터를 선명하게 선형적으로 시각화할 수 있다.

이제 LIT를 실행해 보자.

14.2.2 LIT 실행하기

LIT는 온라인으로 실행하거나 구글 코랩(Google Colab) 노트북으로 실행할 수 있다. 다음 링크에서 두 방법 모두 확인할 수 있다.

- <https://pair-codegithub.io/lit/>

튜토리얼에는 다양한 NLP 작업이 나와있다.

- <https://pair-codegithub.io/lit/tutorials/>

이번 섹션에서는 LIT를 온라인으로 실행하여 감성 분류기를 살펴볼 것이다.

- <https://pair-codegithub.io/lit/tutorials/sentiment/>

‘Explore this demo yourself’를 누르면 LIT의 직관적인 인터페이스를 확인할 수 있다. 다음과 같이 작은 트랜스포머 모델을 사용한다.



Model: `sst2-tiny` Dataset: `sst_dev`

그림 14.7 모델 선택하기

모델을 누르면 다른 모델로 교체할 수 있다. 허깅페이스의 'hosted API'에서도 이와 비슷한 모델의 테스트를 제공하고있다.(<https://huggingface.co/sshleifer/tiny-distilbert-base-uncased-finetuned-sst-2-english>)

LIT의 온라인 버전이 업데이트 됨에 따라 NLP 모델은 조금씩 달라질 수 있다. 모델이 달라질 뿐 전체적인 콘셉트는 동일할 것이다.

PCA 프로젝터를 선택하고 각 샘플에서 이진분류 감성 라벨을 사용하도록 한다.

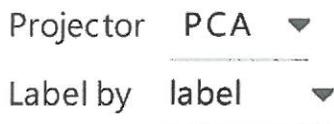
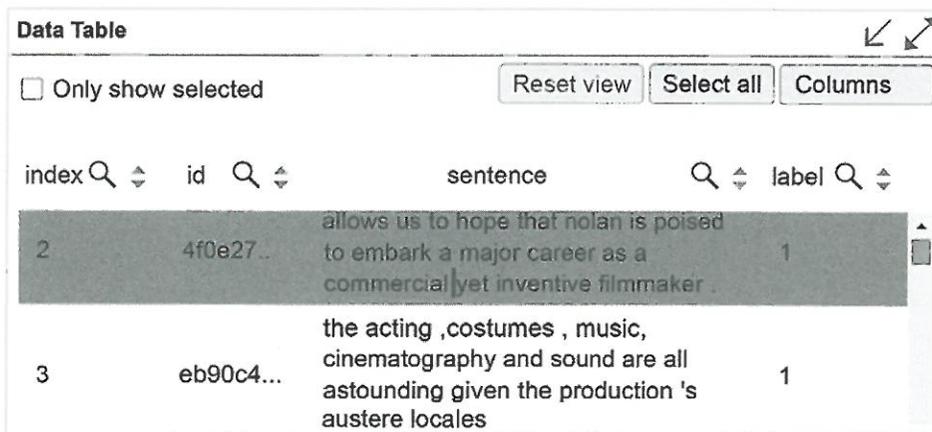


그림 14.8 프로젝터와 라벨 선택하기

이제 데이터 테이블에서 문장(sentence)과 분류 라벨(label)을 클릭해보자.



The screenshot shows a 'Data Table' window with a search bar and a table. The table has four columns: 'index', 'id', 'sentence', and 'label'. Row 2 is highlighted in grey. The text in the 'sentence' column is partially obscured by a vertical line.

index	id	sentence	label
2	4f0e27...	allows us to hope that nolan is poised to embark a major career as a commercial yet inventive filmmaker .	1
3	eb90c4...	the acting ,costumes , music, cinematography and sound are all astounding given the production 's austere locales	1

그림 14.9 문장 선택하기

확률적으로 동작하는 알고리즘이므로 결과는 실행 시마다 달라질 수 있다.

선택한 문장은 데이터포인트 편집기(datapoint editor)에도 나타난다.



그림 14.10 데이터포인트 편집기

데이터포인트 편집기를 사용하면 문장을 편집할 수 있다. 어떤 문장이 다른 클래스로 잘못 분류되어 원인을 찾으려 한다고 가정해 보자. 문장이 올바르게 분류될 때까지 조금씩 문장을 수정하면서, 모델이 어떻게 동작하고 왜 오류가 발생했는지 확인할 수 있다.

선택한 문장은 PCA 프로젝터에도 분류 결과와 함께 표시된다.

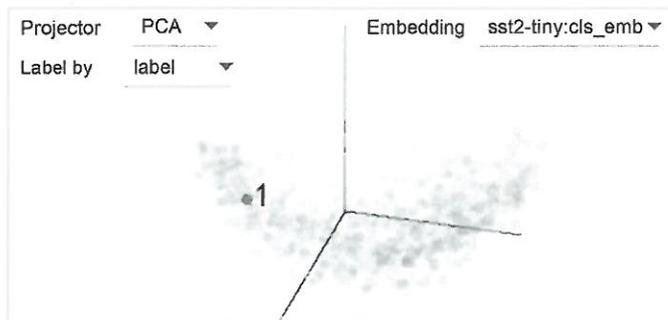


그림 14.11 PCA 프로젝터로 분류 결과 확인

PCA 프로젝터의 한 점을 클릭하면 데이터포인트 편집기에 해당하는 문장이 나타나며, 이런 방식으로 여러 결과를 비교할 수 있다.

이처럼 LIT는 직접 상호 작용할 수 있는 여러 탐색기능을 가지고 있다.

NOTE

LIT로 확인한 결과가 항상 납득할만한 것은 아니다. 하지만 많은 경우 LIT는 가치 있는 인사이트를 준다. 새로운 도구와 기법에 관심을 가지는 것은 매우 중요하다.

이제 딕셔너리 러닝으로 트랜스포머의 층을 시각화해 보자.

14.3

딕셔너리 러닝을 활용한 트랜스포머 시각화

딕셔너리 러닝을 활용한 트랜스포머 시각화는 트랜스포머 팩터(factor)를 기반으로 한다.

14.3.1 트랜스포머 팩터

트랜스포머 팩터는 문맥화된 단어로 구성된 임베딩 벡터이다.⁴³ 문맥화되지 않은 단어는 다의성 (polysemy) 문제를 야기한다. 예를 들어, *separate*는 동사일 수도 형용사일 수도 있으며 분리하다, 구별하다, 흩어지다 등 다양한 의미를 가질 수 있다.

때문에 2021년 윤(Yun et al.)은 임베딩 벡터를 만들 때 문맥화된 단어를 사용했다. 단어 임베딩 벡터는 단어 팩터의 희소(sparse) 선형 표현으로 나타낼 수 있다. 예를 들어, *separate*를 데이터셋의 문맥에 따라 다음과 같이 표현할 수 있다.

$separate = 0.3 \text{ "keep apart"} + 0.3 \text{ "distinct"} + 0.1 \text{ "discriminate"} + 0.1 \text{ "sever"} + 0.1 \text{ "disperse"} + 0.1 \text{ "scatter"}$

선형 표현을 희소한 상태로 유지하기 위해 (다시 말하면, 0으로 가득 찬 거대한 행렬을 만들지 않기 위해) 0 값을 가지는 단어는 표시하지 않았다. 즉 다음과 같은 무의미한 정보는 나타내지 않았다.

$separate = 0.0 \text{ "putting together"} + 0.0 \text{ "identical"}$

요점은 팩터의 계수를 0보다 큰 값으로 강제하여 희소한 상태를 유지하는 것이다.

트랜스포머의 각 층은 각 단어에 대한 은닉 상태(hidden state)를 조회하고, 문맥 속의 단어를 이해하려 노력하며 보이지 않는 종속성을 찾아간다. 트랜스포머 팩터의 희소 선형 조합(sparse linear superposition)은 추론할 계수의 희소 벡터와 딕셔너리 행렬(dictionary matrix)로 표현할 수 있다.

43 역주. 트랜스포머 모델의 층을 통과한 단어는 문장 속 다른 단어와의 관계를 고려해 얻은 임베딩으로 문맥화(Contextualized) 됐다고 볼 수 있다. 각 단어를 다의성을 가진 다른 단어의 문맥화된 임베딩의 선형 결합으로 나타낼 수 있고, 이때 각 단어의 문맥화된 임베딩을 트랜스포머 팩터라 부른다.

$$\varphi R^{dxm} \alpha$$

각 요소는 다음과 같다.

- φ (phi)는 딕셔너리 행렬이다.
- α 는 추론할 계수의 희소 벡터이다.

윤은 단어의 더 깊은 의미를 찾기 위해 알고리즘에 가우시안 노이즈(ϵ)를 추가했다.

또한, 표현을 희소한 상태로 유지하기 위해 방정식에 $\alpha > 0$ 조건을 추가했다.

층의 은닉 상태 집합을 X 로, X 에 속하는 트랜스포머 팩터의 희소 선형 조합을 x 로 표기했다.

종합하면, 다음과 같이 아름답게 딕셔너리 러닝 모델을 표현할 수 있다.

$$X = \varphi \alpha + \epsilon \quad \text{s.t. } \alpha > 0$$

딕셔너리 행렬에서 $\varphi; c$ 는 행렬의 열을 의미하며 트랜스포머 팩터로 구성되어있다.

$\varphi; c$ 는 세 가지 수준으로 구분할 수 있다.

- 낮은 수준(Low-level): 단어의 의미를 명확히 하여 다의어 문제를 해결하는 트랜스포머 팩터
- 중간 수준(Mid-level): 문장 단위의 패턴을 고려하여 중요한 문맥 정보를 가져다주는 트랜스포머 팩터
- 높은 수준(High-level): 더 넓은 범위의 종속성을 설명하는 트랜스포머 팩터

혁신적이고 흥미로우며 효율적인 방법이지만, 아직 시각화를 수행할수는 없다. 윤은 시각화를 위해, AI 해석을 위한 표준 도구인 LIME을 활용하였다.

14.3.2 LIME 소개

LIME(Local Interpretable Model-Agnostic Explanations)는 이름 그대로 AI를 설명(explainable)하기 위한 방법이다. 모델에 독립적인(model-agnostic) 방법이므로 '딕셔너리 러닝을 통한 트랜스포머 시각화'에도 바로 적용할 수 있다.

- 이 방법은 트랜스포머 층의 행렬, 가중치, 행렬곱 등을 깊게 파고들지 않는다.
- 이 방법은 2장, 트랜스포머 모델 아키텍처 살펴보기에서 살펴본 트랜스포머 동작 원리를 설명해주지는 않는다.
- 이번 장에서, 이 방법은 희소 선형 조합의 수학적 결과를 참조한다.

LIME은 데이터셋의 모든 정보를 분석하지는 않는다. 대신 LIME은 예측의 특성을 살펴봄으로써 모델이 국소적으로 신뢰할 만한지(locally reliable) 검사한다.

모델을 모든 관점에 대해 검사하는 대신, 좁은 환경에서의 예측에 집중하는 것이다.

이런 특성은 NLP를 다룰 때 더욱 효과적이다. LIME이 단어의 문맥을 탐색하여 모델 출력에 대한 가치 있는 정보를 제공하기 때문이다.

'딕셔너리 러닝을 통한 트랜스포머 시각화'를 위해 x 를 다음과 같이 표현할 수 있다.

$$x \in \mathbb{R}^d$$

x 에 대한 해석 가능한 표현은 이진 벡터이다.

$$x' \in \{0, 1\}^d$$

이번 목표는 특정 피처가 국소적으로 존재하는지 또는 존재하지 않는지 결정하는 것이다. NLP에서, 피처는 단어로 재구성될 수 있는 토큰을 의미한다.

LIME에서 g 는 트랜스포머 등의 머신러닝 모델을 의미하며, G 는 g 를 포함하고 있는 트랜스포머 모델 집합이다.

$$g \in G$$

LIME의 알고리즘은 어떤 트랜스포머 모델에도 적용될 수 있다.

이 시점에서 우리는 다음을 알고 있다.

- LIME은 단어를 대상으로 하며 다른 단어들로부터 문맥(local context)을 찾는다.
- 즉, LIME으로 단어의 문맥을 알 수 있고 왜 다른 단어가 아니라 이 단어가 선택되었는지 설명할 수 있다.

LIME과 같은 설명 가능한 AI를 살펴보는 것은 트랜스포머와 NLP를 다루는 이 책의 범위를 벗어난다. LIME에 대한 더 많은 정보는 참고 문헌 섹션을 참고하자.

이제 LIME으로 '딕셔너리 러닝을 통한 트랜스포머 시각화'를 어떻게 수행할 수 있는지 살펴보겠다.

먼저 시각화 인터페이스를 확인해보자.

14.3.3 시각화 인터페이스

다음은 트랜스포머 시각화를 위한 상호 작용 페이지의 주소이다. <https://transformervis.github.io/transformervis/>

그림 14.12처럼, 시각화 인터페이스로 특정 층의 트랜스포머 팩터를 클릭 한번으로 분석할 수 있다.

Visualization

In the following box, input a number c indicating the transformer factor $\Phi_{:,c}$ you want to visualize. Then click the button "Visualize!" to visualize this transformer factor at a particular layer. For a transformer factor $\Phi_{:,c}$ and for a layer- l , the visualization is done by listing the 200 word and context with the largest sparse coefficients $\alpha_c^{(l)}$ s

421

← Enter an integer from 0 to 531, indicating the transformer factor you want to visualize.

그림 14.12 트랜스포머 팩터 선택하기

팩터를 선택한 후에는 어떤 층에서 이 팩터를 시각화할 것인지 선택할 수 있다.

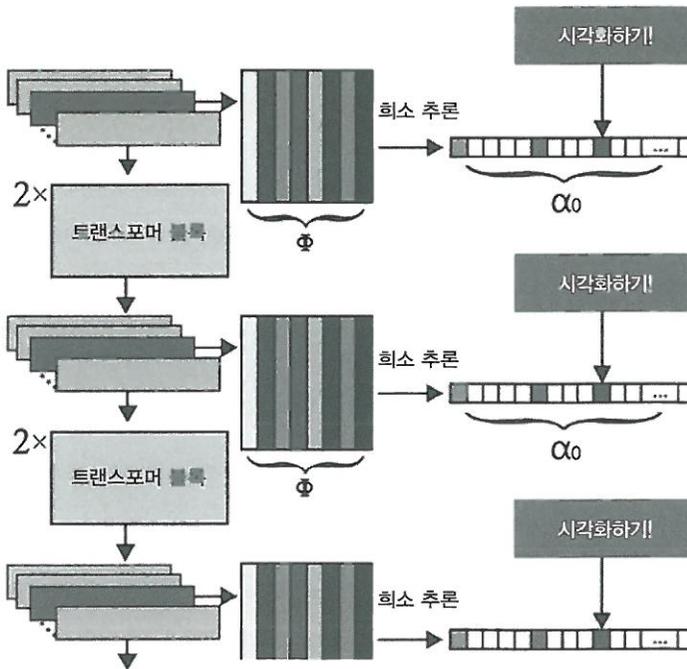


그림 14.13 층마다 시각화 버튼이 있다

첫 부분에서 층에 따른 팩터의 활성도를 확인할 수 있다.

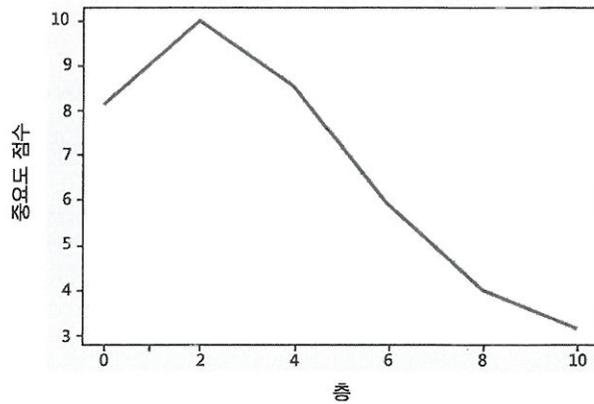


그림 14.14 각 층에 따른 팩터의 중요도

421번 팩터는 낮은 층에서 separate의 어휘 영역에 중점을 둔다.

- music, and while the band initially kept these releases separate, alice in chains' self@-@
- and o. couesi were again regarded as separate as a result of further work in texas,
- in july 2014, and changed to read" a separate moh is presented to an individual for each
- without giving it proper structure or establishing it as a separate doctrine.
- those species, and is now considered to form a separate, monotypic genus – homarinus.
- rp, each npc is typically played by a separate crew member.
- ," abzug" is presented as a separate track.

그림 14.15 낮은 층에서 표현한 "separate"

더 높은 층을 시각화하면 더 넓은 범위의 표현을 확인할 수 있다. 421번 팩터는 separate를 표현했었다. 하지만 그림 14.16처럼, 트랜스포머는 더 높은 층에서 더 깊은 이해를 바탕으로 separate를 distinct와 연결지었다.

- cigarette smoking; it was not even recognized as a distinct disease until 1761.
- the australian freshwater himantura were described as a separate species, *h. dalyensis*, in 2008
- japan, judo and jujutsu were not considered separate disciplines at that time.
- though during the episodes, the scenes took place in separate parts of the episode.
- triaenops in 1947, retained both as separate species; in another review, published in 1982
- *ycoperdon* (unk), but separate from *I. pyriforme*.
- although it is a separate award, its appearance is identical to its british
- ted upper atmosphere in which the gods dwell, as distinct from the

그림 14.16 높은 층에서 표현한 트랜스포머 팩터

다양한 트랜스포머 팩터로 시각화하면, 트랜스포머가 층을 지나며 언어에 대한 인식을 확장한다는 것을 알 수 있다.

좋은 예시들과 더불어 이상한 결과도 보일 것이다. 좋은 예시들을 중심으로 트랜스포머가 언어를 배우는 과정을 이해해 보자. 틀린 결과들로는 모델이 실수를 하는 이유를 생각해 보자. 이 시각화 인터페이스에 사용된 트랜스포머 모델은 최고의 성능을 가지거나 완벽하게 학습된 모델은 아니다.

어떠한 경우라도, 끊임없이 발전하고 있는 이 분야에 관심을 가지고 있자!

모델을 이해하기 위해, 이 장의 깃허브 저장소에 있는 `Understanding_GPT_2_models_with_Ecco.ipynb`를 살펴보자. 이 노트북으로, 트랜스포머가 토큰을 생성하기 전에 후보를 선택하는 방법을 확인할 수 있다.

이번 섹션에서, 트랜스포머가 각 층을 거치며 단어의 의미를 배워가는 것을 보았다. 트랜스포머는 결정을 내리기 전에 후보들을 생성한다. 노트북에서 보듯이 트랜스포머는 확률적이며, 가장 가능성이 높은 후보들을 선별하여 고른다. 다음 문장을 읽어보자.

"The sun rises in the_____."

문장 마지막에는 어떤 단어가 어울릴까? 트랜스포머도 인간처럼 순간적으로 멈치하면서 출력해 낸다.

GPT-2 모델은 sky라는 단어를 골랐다.

0 1 2 3 4 >> 5
The sun rises in the sky

그림 14.17 문장 완성하기

하지만 그림 14.18처럼 GPT-2가 고민했던 다양한 단어들이 있다.

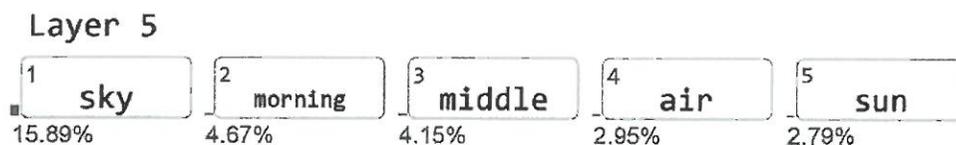


그림 14.18 고려했던 다양한 후보들

sky의 순위가 가장 높은 것을 볼 수 있지만, 두 번째인 morning도 제법 어울리는 단어이다. 모델이 확률적이기 때문에 여러번 실행해보면 다른 결과가 나올수도 있다.

AI와 트랜스포머 관련 분야는 완벽해 보인다.

하지만 아직 우리가 해야 할 일이 많이 남아있다.

14.4

내부를 볼 수 없는 모델 분석하기

이번 장에서 확인한 시각적 인터페이스 기능은 꽤 훌륭해보지만 한계점이 존재한다!

OpenAI의 GPT-3의 경우에는 온라인이나 API로 실행되기 때문에 **서비스형 소프트웨어(Software as a Service, SaaS)**로 제공되는 모델의 가중치에 접근할 수 없다. 연구와 컴퓨터 성능에 수백만 달러를 투자하는 기업들은 오픈 소스가 아닌 종량제 서비스를 제공한다. 이러한 추세는 몇 년 동안 더욱 확대될 것이다.

만약 GPT-3의 소스 코드에 접근하거나 가중치를 확보한다고 해도 9,216개의 어텐션 헤드(96개 층 × 96개 헤드)를 시각적 인터페이스로 분석하기는 쉽지 않다.

오류를 분석하는 것은 여전히 대부분 인간의 몫인 것이다.

예를 들어, 영어에서 프랑스어로 번역할 때 coach라는 단어로 인해 발생할 수 있는 다의성(polysemy) 문제를 생각해 보자. 영어에서 coach는 훈련시키는 사람 또는 버스를 의미하는 단어이다. 하지만 프랑스 단어 coach는 훈련시키는 사람만을 의미한다.

OpenAI의 GPT-3 플레이그라운드(<https://platform.openai.com/playground>)에서 coach를 포함한 문장을 번역하면 복합적인 결과를 얻을 수 있다.

다음 문장은 OpenAI에서 올바르게 번역된다.

English: The coach broke down, and everybody complained.(영어: 코치(버스)가 쓰러졌고 모두가 불평했다.)

French(프랑스어): Le bus a eu un problème et tout le monde s'est plaint.

coach가 적절하게 버스(Le bus)로 번역되었지만, 앞 뒤 문맥이 더 필요했음지도 모른다.

결과는 확률적이므로, 올바른 결과가 나와도 다음번에는 틀릴 수 있다.

반면에 다음 문장은 잘못 번역되었다.

English: The coach was dissatisfied with the team and everybody complained.(영어: 코치는 팀에 불만이 있었고 모두가 불평했다.)

French: Le bus était insatisfait du équipe et tout le monde s'est plaint.(프랑스어: 버스는 팀에 불만이 있었고 모두가 불평했다.)

이번에는 GPT-3가 사람을 의미하는 coach를 버스로 잘못 번역했다. 마찬가지로 결과는 확률적이고 실행 시마다 다를 수 있다.

두 번째 문장에 문맥을 추가한다면 더 나은 번역이 나온다.

English: The coach of the football team was dissatisfied and everybody complained.(영어: 축구팀 코치는 불만족스러워했고 모두가 불평했다.)

French(프랑스어): Le coach de l'équipe de football était insatisfait et tout le monde s'est plaint.

이번 번역은 영어 문장에서의 coach와 같은 의미인 프랑스 단어 coach를 잘 사용했다. 더 많은 문맥을 제공한 덕분이다.

OpenAI의 AI 솔루션(특히 트랜스포머)은 지속적으로 발전하고 있다. 또한 작은 부분에서 4차 산업의 AI 기반 의사 결정은 NLP나 번역 작업 수준의 정교함 없이도 효과적이다.

하지만 클라우드 AI API에서 인간의 개입과 개발은 앞으로도 상당 기간동안 계속 필요할 것이다!

14.5 정리하기

트랜스포머 모델은 단어 수준의 다의어를 구분하고 낮은 수준, 중간 수준, 상위 수준 종속성을 해결한다. 모델은 이런 능력을 얻기 위해 수백만에서 조 단위의 파라미터를 연결하고 학습한다. 이러한 거대한 모델을 해석하는 것은 어려워 보이지만, 다양한 도구가 개발되어 있다.

가장 먼저 BertViz를 설치했다. 상호 작용할 수 있는 인터페이스에서 어텐션 헤드의 동작을 해석해 보았다. 각 층에서 한 단어가 다른 단어와 어떻게 연관되는지 확인했다.

다음으로 논-프로빙 작업을 살펴보았다. NER 같은 프로빙 작업은 트랜스포머가 내부적으로 언어를 어떻게 표현하는지에 대한 인사이트를 준다. 하지만 논-프로빙 방식은 모델의 예측을 분석한다. 예를 들어, LIT를 사용하면 PCA나 UMAP으로 BERT의 출력값을 표현하고 클러스터를 분석하며 예측이 적합한지 확인할 수 있다.

마지막으로, 디셔너리 러닝을 활용하여 트랜스포머를 시각화했다. 트랜스포머 팩터를 선택하고 낮은 층에서 높은 층을 거치며 팩터의 표현이 진화하는 것을 분석하고 시각화했다. 팩터는 점진적으로 단어 수준의 다의어에서 문장의 문맥 그리고 장기 종속성까지 다루게 된다.

트랜스포머의 활동을 사용자 친화적인 방식으로 시각화하고 해석할 수 있다는 것이 이 장의 핵심이다. 이번 장에서 소개한 도구들은 점점 발전해 나갈 것이다. 다음 장에서는 새로운 트랜스포머 모델을 살펴보고, 트랜스포머 프로젝트에서 적합한 모델을 고르기 위해 위험을 관리하는 방법을 알아볼 것이다.

14.6 문제

01. BertViz는 BERT의 마지막 층만 보여준다. 참 / 거짓
02. BertViz로 BERT의 각 층에 있는 어텐션 헤드를 시각화할 수 있다. 참 / 거짓
03. BertViz로 토큰 간의 관계를 시각화할 수 있다. 참 / 거짓
04. LIT는 BertViz처럼 어텐션 헤드 내부의 모습을 시각화한다. 참 / 거짓
05. 프로빙(probing)은 모델이 내부적으로 언어를 어떻게 표현하는지 분석하는 방법이다. 참 / 거짓
06. NER은 프로빙 방식의 작업이다. 참 / 거짓
07. PCA와 UMAP은 논-프로빙(non-probing) 방식의 작업이다. 참 / 거짓
08. LIME은 모델에 독립적(agnostic)이다. 참 / 거짓
09. 트랜스포머의 층을 지날수록 토큰간 관계가 심화된다. 참 / 거짓
10. 트랜스포머의 시각화는 AI를 설명하는 새로운 축을 제공했다. 참 / 거짓

14.7 참고 문헌

- BertViz: Jesse Vig, 2019, A Multiscale Visualization of Attention in the Transformer Model, 2019, <https://aclanthology.org/P19-3007.pdf>
- BertViz: <https://github.com/jessevig/BertViz>
- LIT, 감성 분석 표현에 대한 설명: <https://pair-code.github.io/lit/tutorials/sentiment/>
- LIT: <https://pair-code.github.io/lit/>
- 딕셔너리 러닝과 트랜스포머 시각화: Zeyu Yun, Yubei Chen, Bruno A Olshausen, Yann LeCun, 2021, Transformer visualization via dictionary learning: contextualized embedding as a linear superposition of transformer factors, <https://arxiv.org/abs/2103.15949>
- 딕셔너리 러닝과 트랜스포머 시각화: <https://transformervis.github.io/transformervis/>

15장

NLP부터 범용 트랜스포머 모델까지

- 15.1 모델과 생태계 선택하기
- 15.2 리포머
- 15.3 DeBERTa
- 15.4 범용 모델에서 비전 트랜스포머까지
- 15.5 확장되는 모델 세계
- 15.6 정리하기
- 15.7 문제
- 15.8 참고 문헌

지금까지 인코더와 디코더 층이 있는 오리지널 트랜스포머 모델의 변형을 살펴 보고, 인코더 또는 디코더로만 이루어진 스택이 있는 다른 모델도 살펴봤다. 또한 층과 파라미터의 크기도 증가시켜 보았다. 그러나 동일한 층과 병렬화된 어텐션 헤드를 사용하는 트랜스포머의 기본 구조는 모두 같다.

이번 장에서는 오리지널 트랜스포머의 기본 구조를 존중하면서도 몇 가지 중요한 변화를 준 혁신적인 트랜스포머 모델을 살펴본다. 레고 조각 상자가 제공하는 수많은 가능성처럼 수많은 트랜스포머 모델이 등장할 것이다. 수백 가지 방법으로 조립할 수 있다! 트랜스포머 모델의 각 층은 레고의 첨단 AI 조각이다.

먼저 수많은 제품 중에서 어떤 트랜스포머 모델을 선택할지, 그리고 어떤 생태계(ecosystem)에서 구현할지 고민해 볼 것이다.

그런 다음 리포머 모델의 **LSH(Locality Sensitivity Hashing)** 버킷(bucket)과 청크화(chunking)를 살펴볼 것이다. 그런 다음 DeBERTa 모델에서 풀기(disentanglement)란 무엇인지 알아볼 것이다. 또한 디코더에서 위치를 관리하는 방법도 소개한다. 고성능 트랜스포머 모델인 DeBERTa는 인간 기준 성능을 뛰어넘는다.

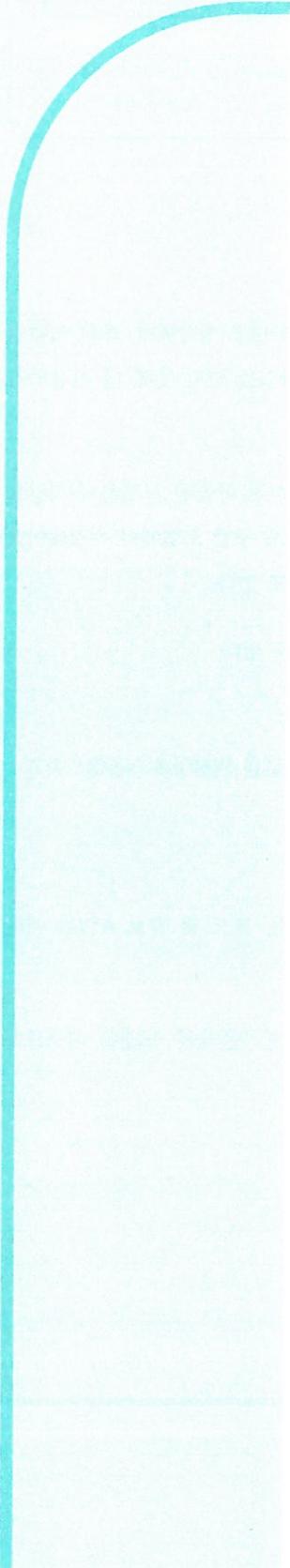
마지막 단계는 ViT, CLIP, DALL-E와 같은 강력한 컴퓨터 비전 트랜스포머를 살펴보는 것이다. CLIP과 DALL-E를 OpenAI GPT-3에 추가하고 구글 BERT(구글에서 학습)를 극소수의 **파운데이션 모델** 그룹에 추가할 수 있다.

이러한 강력한 파운데이션 모델은 트랜스포머가 범용(task-agnostic)이라는 것을 증명한다. 트랜스포머는 시퀀스를 학습한다. 이러한 시퀀스는 시각, 소리, 그리고 시퀀스로 표현되는 모든 유형의 데이터를 포함한다.

이미지 또한 언어처럼 시퀀스로 이루어진 데이터이다. ViT, CLIP, DALL-E 모델로 학습하여 비전 모델을 혁신적인 수준으로 끌어올려 볼 것이다.

이 장이 끝나면 범용 트랜스포머의 세계가 상상력과 창의력의 세계로 진화했음을 알게 될 것이다.

이 장에서는 다음 주제들을 다룬다.

- 
- 트랜스포머 모델 선택하기
 - 리포머 트랜스포머 모델
 - LSH(Locality Sensitivity Hashing)
 - 버킷 및 청크화 기술
 - DeBERTa 트랜스포머 모델
 - 어텐션 풀기(Disentangled attention)
 - 절대 위치(Absolute positions)
 - CLIP을 이용한 텍스트-이미지 비전 트랜스포머
 - 창의적인 텍스트-이미지 비전 트랜스포머, DALL-E

첫 번째 단계로 모델과 생태계를 선택하는 방법을 살펴보자.

15.1 모델과 생태계 선택하기

트랜스포머 모델을 다운로드하여 테스트하려면 기계와 인적 자원이 필요하다고 생각했다. 또한 이때까지 플랫폼에 온라인 샌드박스 없었다면 몇 가지 예제를 테스트하는 작업을 하지 못해 더 사용하기 위험하다고 생각했을 수도 있다.

하지만 허깅페이스에서는 리포머와 DeBERTa 섹션에서 살펴볼 것처럼 사전 학습된 모델을 실시간으로 자동으로 다운로드한다! 덕분에 머신에 직접 아무것도 설치하지 않고도 구글 코랩에서 허깅페이스 모델을 실행할 수 있다. 또한 허깅페이스 모델을 온라인으로 테스트할 수도 있다.

“설치”없이도 분석할 수 있다. “설치할 것이 없다”는 것은 다음을 의미할 수 있다.

- 온라인에서 트랜스포머 작업 실행
- 사전 설치된 구글 코랩 VM에서 트랜스포머를 실행하여 사전 학습된 모델을 원활하게 다운로드하고 코드 몇 줄로 실행
- API로 트랜스포머 실행

“설치”의 정의는 지난 몇 년 동안 확장되었다. “온라인”의 정의도 넓어졌다. 코드 몇 줄로 API를 메타 온라인(meta-online) 테스트로 실행해 볼 수 있다.

이 섹션에서는 “설치 없이”와 “온라인”을 넓은 의미로 사용한다. 그림 15.1은 “온라인” 모델을 테스트하는 방법을 보여준다.

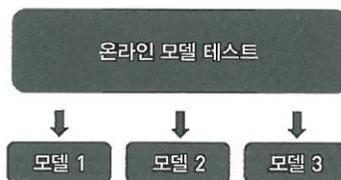


그림 15.1 온라인 트랜스포머 모델 테스트

지난 10년 동안 테스트는 다음처럼 유연하고 생산성 있도록 발전했다.

- 허깅페이스는 DeBERTa 및 다양한 모델의 API를 호스팅한다. 또한 허깅페이스는 트랜스포머 모델을 학습하고 배포할 수 있는 AutoML 서비스를 제공한다.
- OpenAI의 GPT-3 엔진은 온라인 플레이그라운드에서 실행되며 API를 제공한다. OpenAI는 다양한 NLP 작업을 처리하는 모델을 제공한다. 이 모델은 학습이 필요 없다. GPT-3의 10억 개 파라미터 제로 샷 엔진은 인상적이다. 많은 파라미터를 가진 트랜스포머 모델이 전반적으로 더 나은 결과를 생성한다. 마이크로소프트 애저(Azure), 구글 클라우드 AI, AllenNLP 및 기타 플랫폼에서 흥미로운 서비스를 제공한다.
- 논문을 읽어 온라인 모델을 분석할 수도 있다. 좋은 예로는 2021년 구글에서 발표한 페더스(Fedus et al.)의 「Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity(스위치 트랜스포머: 간단하고 효율적인 희소성으로 수조 개 파라미터 모델로 확장하기)」가 있다. 구글은 8장, 법률 및 금융 문서에 트랜스포머를 적용하여 요약하기에서 연구한 T5 모델의 크기를 늘렸다. 이 논문에서는 GPT-3와 같은 대규모 온라인 모델의 전략을 확인했다.

그러나 결국 솔루션을 선택하는 위험을 감수하는 것은 본인이다. 플랫폼과 모델을 탐색하는 데 투자한 시간은 선택을 마친 후 프로젝트 구현을 최적화하는 데 도움이 된다.

그림 15.2에 표시된 것처럼 세 가지 방법으로 선택한 것을 호스팅할 수 있다.

- 로컬 머신에서 API를 실행할 수 있다. OpenAI, 구글 Cloud AI, 마이크로소프트 애저 AI, 허깅페이스 등이 좋은 API를 제공한다. 애플리케이션은 로컬 머신에 있을 수 있지만 클라우드 서비스를 사용할 수도 있다.
- **Amazon Web Services(AWS)** 또는 구글 클라우드와 같은 클라우드 플랫폼에서 모델을 학습, 미세 조정, 테스트 및 실행할 수 있다. 이 경우 로컬 머신에 애플리케이션이 없고 모든 것이 클라우드에 있다.
- 로컬 머신, 데이터센터 가상 머신 또는 어디서든 API를 사용할 수 있다. 즉, API가 풍차, 비행기, 로켓 또는 자율 주행 차량과 같은 물리적 시스템에 통합될 수 있다. 따라서 시스템은 API를 통해 다른 시스템과 영구적으로 연결될 수 있다.

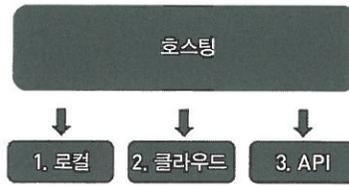


그림 15.2 모델에 대한 옵션 구현하기

결국 결정을 내리는 것은 여러분의 몫이다. 충분히 테스트하고, 분석하고, 비용을 계산하고, 팀과 협력하여 다양한 관점에 귀를 기울이자. 트랜스포머의 작동 원리를 더 많이 이해할수록 더 나은 선택을 할 수 있다.

이제 오리지널 트랜스포머 모델을 변형한 리포머에 대해 살펴보자.

또한 리포머는 다중 층 모델에 대한 정보를 저장하는 대신 각 층의 입력을 재계산하여 메모리 문제도 해결한다. 재계산은 일부 대규모 다중 층 모델의 경우 테라바이트의 메모리를 소비하는 대신 온디맨드 방식으로 이루어진다.

이제 표도르 도스토예프스키의 『죄와 벌』 영어 번역본으로 학습된 리포머 모델을 사용한다.

15.2.1 예제 실행하기

호스팅된 추론 API를 사용하여 온라인에서 직접 실행해 보자. 입력 문장은 다음과 같다.

The student was impoverished and did not know what to do.(학생은 가난했고 무엇을 해야 할지 몰랐다.)

온라인 인터페이스 링크에 입력이 포함되어 있다.(<https://huggingface.co/google/reformer-crime-and-punishment?text=The+student+was+impoverished+and+did+not+know+what+to+do>)

호스팅된 추론 API가 입력 문장과 함께 나타난다. compute를 클릭하면 입력 바로 아래에 결과가 표시된다.

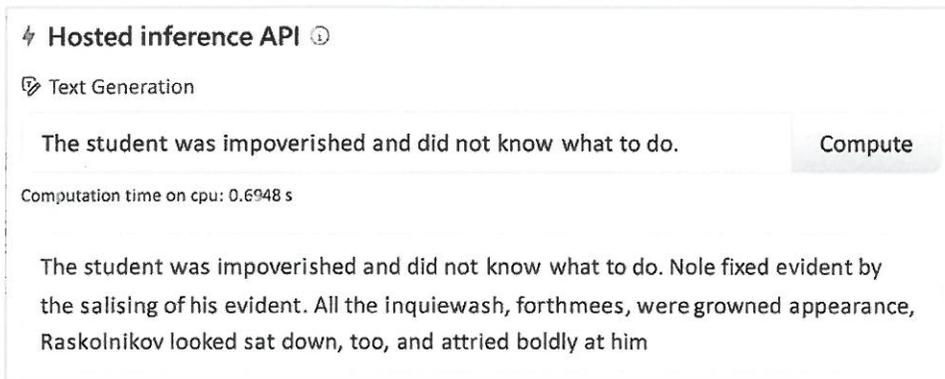


그림 15.4 리포머의 호스팅된 추론 API

알고리즘이 확률적이기 때문에 다른 응답을 얻을 수 있다. 리포머는 OpenAI의 GPT-3와 같은 수십 억 비트(bits) 정보를 가진 슈퍼컴퓨터는 아니지만 합리적으로 학습되었다. 리포머의 결과가 그다지 인상적이지 않다. 더 나은 결과를 얻으려면 더 많은 학습과 미세 조정이 필요하다.

OpenAI의 GPT-3 엔진은 똑같은 입력 문장을 넣으면 텍스트 완성에 대해 다음 결과를 생성한다.

학생은 가난했고 무엇을 해야 할지 몰랐다. 의지할 사람도 없었고 머물 곳도 찾을 수 없었다. 학생은 가방에서 메모지를 꺼내 다음처럼 글을 쓰기 시작했다. “제 이름은 XXXX입니다. 저는 XXXX의 학생입니다. 저는 가족도, 친구도, 돈도 없습니다.”

결과를 비교하면 GPT-3엔진이 더 설득력이 있다. 회원가입 후 OpenAI의 플레이그라운드에 접속할 수 있다.(<https://openai.com/>)

NOTE

다른 트랜스포머 모델 및 대부분의 딥러닝 모델과 마찬가지로 OpenAI GPT-3는 확률론적 알고리즘을 기반으로 한다. 결과는 모델마다 다를 수 있다.

이는 파라미터 수십억 개를 포함하는 고도로 잘 학습된 트랜스포머 모델이 혁신적인 트랜스포머 모델 아키텍처보다 성능이 뛰어날 수 있음을 보여준다.

슈퍼컴퓨터 기반의 클라우드 AI 플랫폼이 로컬이나 덜 강력한 클라우드 플랫폼보다 점차 더 나은 성능을 발휘할까? 솔루션을 고르기 전에 프로토타입을 통해 이러한 문제를 해결해야 한다.

NOTE

트랜스포머 모델의 확률적 특성으로 인해 실행마다 다른 결과가 나올 수 있다. 또한 온라인 플랫폼은 지속적으로 인터페이스를 변경하므로 적응해야 한다.

DeBERTa의 또 다른 혁신적인 아키텍처를 살펴보자.

15.3 DeBERTa

트랜스포머에 대한 또 다른 새로운 접근 방식은 풀기를 통해 찾을 수 있다. AI에서 풀기는 표현 기능을 분리하여 학습 프로세스를 보다 유연하게 만들 수 있다. Pengcheng He, Xiaodong Liu, Jianfeng Gao, Weizhu Chen은 다음 논문으로 트랜스포머에 풀기를 적용한 DeBERTa를 소개했다. DeBERTa: Decoding-enhanced BERT with Disentangled Attention: <https://arxiv.org/abs/2006.03654>

DeBERTa에 구현된 두 가지 주요 아이디어는 다음과 같다.

- 트랜스포머 모델에서 콘텐츠 벡터와 위치 벡터를 풀어내어, 두 벡터를 개별적으로 학습한다.
- 사전 학습 시 디코더에서 마스킹된 토큰을 예측할 때 절대 위치를 사용한다.

논문 저자는 깃허브에서 코드를 제공한다(<https://github.com/microsoft/DeBERTa>)

DeBERTa는 SuperGLUE 순위표에서 인간 기준 성능을 넘어섰다.

Rank	Name	Model	
1	ERNIE Team - Baidu	ERNIE 3.0	
+	2	Zirui Wang	T5 + Meena, Single Model (Meena Team - Google Brain)
+	3	DeBERTa Team - Microsoft	DeBERTa / TuringNLRv4

그림 15.5 SuperGLUE 순위표에서 DeBERTa

허깅페이스의 클라우드 플랫폼에서 예제를 실행해 보자.

15.3.1 예제 실행하기

허깅페이스의 클라우드 플랫폼에서 예제를 실행하려면 다음 링크를 클릭한다.

<https://huggingface.co/cross-encoder/nli-deberta-base>

호스팅된 추론 API에 사용 가능한 클래스 이름 및 예제가 출력된다.

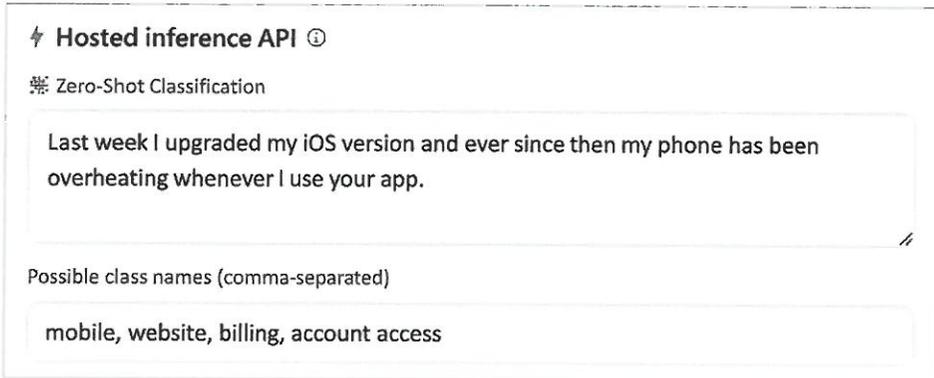


그림 15.6 DeBERTa의 호스팅된 추론 API

분류할 클래스 종류는 mobile, website, billing, account access이다.

결과는 흥미롭다. GPT-3 키워드 작업과 비교해 보자. 먼저 <https://openai.com/>에 가입한다.

다음처럼 Text와 Keywords를 입력한다.

Text: Last week I upgraded my iOS version and ever since then my phone has been overheating whenever I use your app.

Keywords: app, overheating, phone

결과 키워드는 app, overheating, phone이다.

지금까지 DeBERTa와 GPT-3 트랜스포머를 살펴봤다. 이제 트랜스포머를 비전 모델로 확장해 보자.

15.4

범용 모델에서 비전 트랜스포머까지

1장, 트랜스포머란 무엇인가?에서 살펴본 것 같이 파운데이션 모델에는 두 가지 고유한 특성이 있다.

- **이머전스(Emergence)** - 파운데이션 모델로써 트랜스포머 모델은 학습되지 않은 작업을 수행할 수 있다. 슈퍼컴퓨터에서 학습된 대형 모델이며, 다른 많은 모델처럼 특정 작업을 대상으로 학습되지 않았다. 파운데이션 모델은 시퀀스를 이해하는 방법을 학습한다.
- **단일화(Homogenization)** - 동일한 기본 아키텍처로 여러 도메인에서 동일한 모델을 사용할 수 있다. 파운데이션 모델은 다른 어떤 모델보다 데이터를 통해 새로운 기술을 더 빠르고 더 잘 학습할 수 있다.

GPT-3 및 구글 BERT(구글에서 학습한 BERT 모델만 해당)는 범용 파운데이션 모델이다. 이러한 범용 모델은 ViT, CLIP, DALL-E 모델로 이어진다. 트랜스포머는 놀라운 시퀀스 분석 능력을 가지고 있다.

트랜스포머 모델의 추상화 수준은 다중 모달 뉴런(multi-modal neurons)으로 이어진다.

다중 모달 뉴런은 이미지를 픽셀 또는 이미지 패치로 토큰화한다. 비전 트랜스포머는 이것을 단어로 처리할 수 있다. 이미지가 인코딩되면 트랜스포머 모델은 그림 15.7처럼 토큰을 모든 단어 토큰으로 간주한다.

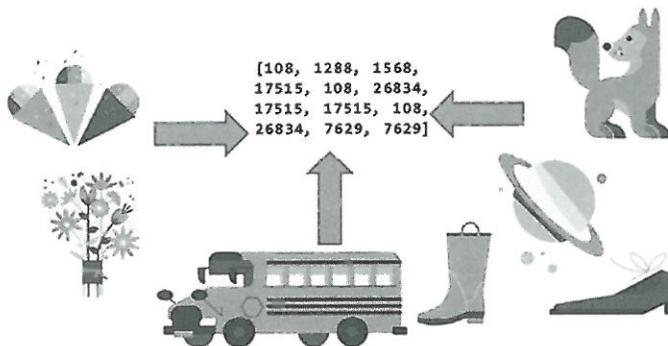


그림 15.7 이미지를 단어와 유사한 토큰으로 인코딩할 수 있다.

이 섹션에서는 다음을 다룬다.

- ViT, 이미지를 단어 패치로 처리하는 비전 트랜스포머
- CLIP, 텍스트와 이미지를 인코딩하는 비전 트랜스포머
- DALL-E, 텍스트로 이미지를 구성하는 비전 트랜스포머

먼저 이미지를 단어 패치로 처리하는 비전 트랜스포머인 ViT에 대해 살펴보자.

15.4.1 ViT - 비전 트랜스포머

2021년 도소비츠키(Dosovitskiy et al.)는 논문 제목에서 비전 트랜스포머 아키텍처의 본질을 다음처럼 요약했다. '대규모 이미지 인식을 위한 트랜스포머의 이미지는 16×16 단어의 가치가 있다.'

이미지는 16×16 단어의 패치로 변환할 수 있다.

코드를 살펴보기 전에 먼저 ViT의 아키텍처를 살펴보자.

15.4.1.1 ViT의 기본 아키텍처

비전 트랜스포머는 이미지를 단어 패치로 처리할 수 있다. 이 섹션에서는 과정을 세 단계로 진행한다.

1. 이미지를 패치로 분할하기
2. 패치의 선형 투영(linear projection)
3. 하이브리드 입력 임베딩 서브 층

첫 번째 단계로 이미지를 동일한 크기의 패치로 분할한다.

15.4.1.2 1단계: 이미지를 패치로 분할하기

그림 15.8처럼 이미지를 n 개의 패치로 분할한다. 모든 패치의 크기가 16×16 처럼 동일하다면 패치 개수에 대한 규칙은 없다.

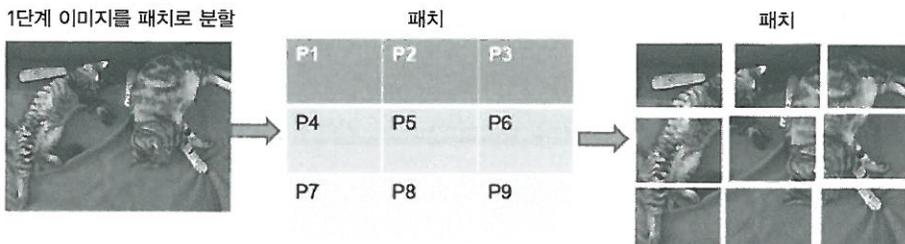


그림 15.8 이미지를 패치로 분할하기

이제 동일한 크기의 패치가 시퀀스의 단어를 나타낸다. 아직 패치로 무엇을 할 것인가에 대한 문제가 남아 있다. 각 유형의 비전 트랜스포머에는 고유한 방법이 있다.

NOTE

이미지 인용: 이 섹션과 이후 섹션에 사용된 고양이 이미지는 2006년 DocChewbacca: <https://www.flickr.com/photos/st3f4n/>에서 촬영한 것이다. 이미지는 Flickr 무료 라이선스에 따라 사용되었다. <https://creativecommons.org/licenses/by-sa/2.0/> 자세한 내용은 Flickr에서 DocChewbacca의 이미지를 참조한다. <https://www.flickr.com/photos/st3f4n/210383891>

이제 ViT로 입력하기 위해 2단계에서는 평평한(flatten) 이미지로 선형 투영을 준비해 보자.

15.4.1.3 2단계: 평평한 이미지로 선형 투영 준비하기

1단계에서는 이미지를 동일한 크기의 패치로 변환했다. 패치를 사용한 이유는 이미지를 픽셀 단위로 처리하는 것을 피하기 위함이다. 하지만 패치를 처리하는 방법을 찾아야 하는 과제가 남아 있다.

구글 리서치 팀은 그림 15.9처럼 이미지를 분할하여 얻은 패치를 사용하여 평평한 이미지의 선형 투영을 설계하기로 결정했다.

2단계: 평평한 이미지의 선형 투영



그림 15.9 평평한 이미지의 선형 투영

아이디어는 단어 시퀀스와 유사한 패치 시퀀스를 얻는 것이다. 이제 평평한 이미지 시퀀스를 임베딩하는 문제가 남았다.

15.4.1.4 3단계: 하이브리드 입력 임베딩 서브 층

단어와 유사한 이미지 시퀀스는 트랜스포머에 들어갈 수 있다. 문제는 여전히 이미지라는 점이다! 구글 리서치는 그림 15.10의 하이브리드 입력 모델이 작업을 수행할 수 있다고 판단했다.

- 컨볼루션(convolutional) 네트워크를 추가하여 패치의 선형 투영을 임베딩한다.
- 위치 인코딩을 추가하여 원본 이미지의 구조를 유지한다.
- 그런 다음 임베딩된 입력을 표준 BERT와 유사한 인코더로 처리한다.

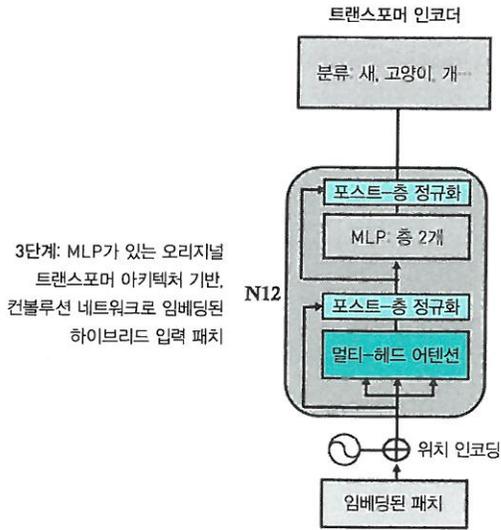


그림 15.10 하이브리드 입력 서브 층과 표준 인코더

구글 리서치에서 NLP 트랜스포머 모델을 비전 트랜스포머로 변환하는 현명한 방법을 발견했다. 이제 비전 트랜스포머의 히킹페이스 예제를 코드로 구현해 보자.

15.4.1.5 코드로 보는 비전 트랜스포머

이 섹션에서는 비전 트랜스포머의 특정 아키텍처와 관련된 코드에 중점을 둔다.

깃허브의 이번 챗터 저장소에 있는 Vision_Transformers.ipynb를 연다.

구글 코랩 VM에는 torch나 torchvision같은 많은 패키지가 사전 설치되어 있다. 노트북의 첫 번째 셀에 있는 명령의 주석 처리를 해제하면 설치된 패키지가 출력된다.

#아래 주석을 해제하고 실행해 사전 설치된 모듈 목록을 출력한다.

```
#!pip list -v
```

그런 다음 노트북의 비전 트랜스포머(ViT) 셀로 이동한다. 노트북이 먼저 히킹페이스 트랜스포머를 설치하고 필요한 모듈을 임포트한다.

```
!pip install transformers
from transformers import ViTFeatureExtractor, ViTForImageClassification
from PIL import Image
import requests
```

NOTE

책을 집필할 당시, 허깅페이스는 지속적인 변경 사항으로 인해 코드가 불안정할 수 있다고 경고했다. 그렇다고 해서 ViT 모델을 탐구하는 것을 멈춰서는 안 된다. 새로운 영역을 테스트하는 것은 최첨단 기술을 익히는데 필수적이다!

그런 다음 COCO 데이터셋에서 이미지를 다운로드한다. 더 많은 실험을 하고 싶다면 해당 웹사이트에서 포괄적인 데이터셋을 찾을 수 있다. <https://cocodataset.org/>

VAL2017 데이터셋을 다운로드하자. COCO 데이터셋 웹사이트의 지침에 따라 프로그램을 통해 이미지를 얻거나 로컬에서 데이터셋을 다운로드할 수 있다.

VAL2017에는 ViT 모델을 테스트할 수 있는 이미지 5,000개가 포함되어 있다. 이미지 5,000개 중 어떤 이미지라도 실행할 수 있다.

고양이 이미지로 노트북을 테스트해 보자. 먼저 URL로 고양이 이미지를 검색한다.

```
url = 'http://images.cocodataset.org/val2017/000000039769.jpg'
image = Image.open(requests.get(url, stream=True).raw)
```

그런 다음 구글의 피처 추출기(feature extractor)와 분류 모델을 다운로드한다.

```
feature_extractor = ViTFeatureExtractor.from_pretrained('google/vit-
basepatch16-224')
model = ViTForImageClassification.from_pretrained('google/vit-
basepatch16-224')
```

이 모델은 224×224 해상도 이미지로 학습되었지만 피처 추출 및 분류를 위해 16×16 패치를 사용한 다. 노트북이 모델을 실행하고 예측을 수행한다.

```
inputs = feature_extractor(images=image, return_tensors="pt")
outputs = model(**inputs)
logits = outputs.logits
# 모델이 1,000개 ImageNet 클래스 중 하나를 예측한다.
predicted_class_idx = logits.argmax(-1).item()
print("Predicted class:"predicted_class_idx," ", model.config.
id2label[predicted_class_idx])
```

출력은 다음과 같다.

```
Predicted class: 285 : Egyptian cat
```

예측 뒤에 나오는 코드를 살펴보면 다음과 같은 세부적인 정보를 얻을 수 있다.

- `model.config.id2label`로 클래스의 라벨을 나열할 수 있다. 1,000개의 클래스와 각각의 설명이 출력된다.

```
{0: 'tench, Tinca tinca',1: 'goldfish, Carassius auratus', 2: 'great white shark, white shark, man-eater, man-eating shark, Carcharodon carcharias',3: 'tiger shark, Galeocerdo cuvieri'.....999: 'toilet tissue, toilet paper, bathroom tissue'}
```

- `model` 명령어로, 컨볼루션 입력 서브 층으로 시작하는 모델의 아키텍처를 확인할 수 있다.

```
(embeddings): ViTEmbeddings(  
  (patch_embeddings): PatchEmbeddings(  
    (projection): Conv2d(3, 768, kernel_size=(16, 16), stride=(16, 16))  
  )  
)
```

컨볼루션 입력 임베딩 서브 층 이후 모델은 BERT와 유사한 인코더이다.

NLP 트랜스포머에서 이미지용 트랜스포머로, 그리고 모든 것을 위한 트랜스포머로 빠르게 이어지는 혁신적인 움직임을 천천히 살펴보길 바란다.

이제 또 다른 컴퓨터 비전 모델인 CLIP을 살펴보자.

15.4.2 CLIP

CLIP(Contrastive Language-Image Pre-Training)은 트랜스포머의 철학을 따른다. 트랜스포머 타입의 층에 데이터 시퀀스를 연결한다. 이번에는 텍스트 쌍을 전달하는 대신 텍스트-이미지 쌍을 전달한다. 데이터가 토큰화, 인코딩, 임베딩되면 범용 모델인 CLIP은 다른 데이터 시퀀스와 마찬가지로 텍스트-이미지 쌍을 학습한다.

이것은 두 이미지를 대조(contrast)하며 학습하는 방법이다. 마치 우리가 틀린 그림 찾기를 할 때 두 이미지를 비교하는 것과 유사하다.

코드를 살펴보기 전에 먼저 CLIP의 아키텍처를 살펴보자.

15.4.2.1 CLIP의 기본 아키텍처

대조(contrastive) 이미지의 차이점과 유사점으로 이미지가 서로 어떻게 어울리는지 학습한다. 이미지와 캡션은 사전 학습을 통해 서로를 향해 가는 길을 찾는다. 사전 학습 후 CLIP은 새로운 작업을 학습한다.

CLIP은 GPT 모델처럼 다양한 용도로 전이(transfer)될 수 있다. 예를 들어 새로운 시각적 콘셉트를 학습하고 동작 인식 등에 사용할 수 있다. 캡션에 따라 무궁무진한 분야로 응용할 수 있는 것이다.

ViT는 이미지를 단어와 같은 패치로 분할한다. CLIP은 그림 15.11처럼 코사인 유사도를 극대화하기 위해 (캡션, 이미지) 쌍에 대해 텍스트와 이미지 인코더를 동시 학습한다.

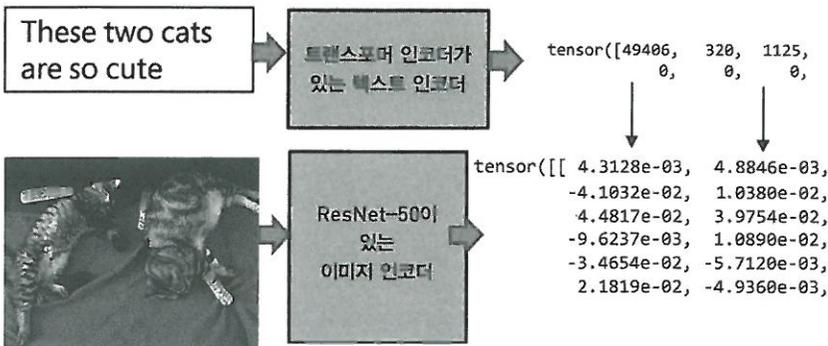


그림 15.11 텍스트와 이미지 공동 학습

그림 15.11은 트랜스포머가 입력된 텍스트로 표준 트랜스포머 인코더를 실행하는 방법을 보여준다. 인코더는 트랜스포머 구조에서 이미지에 대해 ResNet 50층의 CNN을 실행한다. ResNet 50은 멀티 헤드 QKV 어텐션 헤드의 풀링(pooling) 메커니즘으로 평균 풀링(average pooling)을 사용하도록 수정되었다.

CLIP이 텍스트-이미지 시퀀스를 학습하여 예측하는 방법을 살펴보자.

15.4.2.2 코드로 보는 CLIP

깃허브에서 이번 챕터 저장소에 있는 `Vision_Transformers.ipynb`를 열고 노트북의 CLIP 셀로 이동한다. 프로그램은 파이토치와 CLIP을 설치하는 것으로 시작된다.

```
!pip install ftfy regex tqdm
!pip install git+https://github.com/openai/CLIP.git
```

또한 프로그램은 이미지에 접근하기 위해 모듈과 CIFAR-100을 임포트한다.

```
import os
import clip
import torch
from torchvision.datasets import CIFAR100
```

인덱스가 0에서 9,999 사이인 이미지 10,000개를 사용할 수 있다. 다음 단계로 예측할 이미지를 선택한다.

Select an image index between 0 and 9999

index: 15

그림 15.12 이미지 인덱스 선택하기

그런 다음 프로그램은 사용 가능한 장치(GPU 또는 CPU)에 모델을 불러온다.

```
# 모델 불러오기
device = "cuda" if torch.cuda.is_available() else "cpu"
model, preprocess = clip.load('ViT-B/32', device)
```

이미지가 다운로드된다.

```
# 데이터셋 다운받기
cifar100 = CIFAR100(root=os.path.expanduser("~/cache"), download=True,
train=False)
```

입력이 준비되었다.

```
# 입력 데이터 준비하기
```

```
image, class_id = cifar100[index]
image_input = preprocess(image).unsqueeze(0).to(device)
text_inputs = torch.cat([cliptokenize(f"a photo of a {c}") for c in
cifar100.classes]).to(device)
```

예측하기 전에 선택한 입력을 시각화해 보자.

```
import matplotlib.pyplot as plt
from torchvision import transforms
plt.imshow(image)
```

출력된 index 15가 사자라는 것을 알 수 있다.

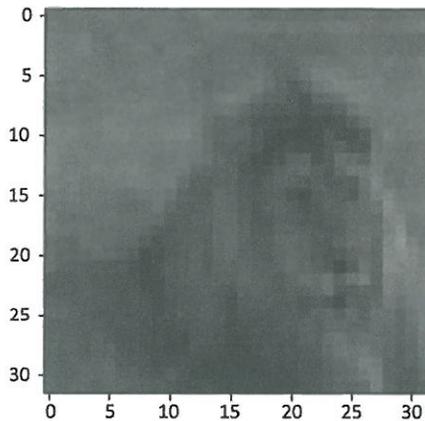


그림 15.13 인덱스 15의 이미지

NOTE

이 섹션의 이미지는 「Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, (2009: <https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf>)에서 가져왔다. CIFAR-10 및 CIFAR-100 데이터셋(toronto.edu)의 일부이다.(<https://www.cs.toronto.edu/~kriz/cifar.html>)

우리는 인간이기 때문에 이것이 사자라는 것을 알 수 있지만, NLP 용으로 설계된 트랜스포머는 이미지가 무엇인지 학습해야 한다. 이제 이미지를 얼마나 잘 인식할 수 있는지 살펴본다.

이 프로그램은 피처를 계산할 때 이미지 입력과 텍스트 입력을 분리하여 공동 트랜스포머 모델을 실행하는 것을 보여준다.

```
# 피처 계산하기
with torch.no_grad():
    image_features = model.encode_image(image_input)
    text_features = model.encode_text(text_inputs)
```

이제 CLIP이 예측을 수행하고 예측 결과 상위 5개를 출력한다.

```
# 이미지와 가장 유사한 상위 다섯 라벨 선택하기
image_features /= image_features.norm(dim=-1, keepdim=True)
text_features /= text_features.norm(dim=-1, keepdim=True)
similarity = (100.0 * image_features @ text_features.T).softmax(dim=-1)
values, indices = similarity[0].topk(5)
# 결과 출력하기
print("\nTop predictions:\n")
for value, index in zip(values, indices):
    print(f"{cifar100.classes[index]}: {100 * value.item():.2f}%")
```

더 많은 예측을 얻거나 더 적은 예측을 얻으려면 topk(5)를 수정한다. 예측 결과 상위 5개가 출력된다.

```
Top predictions:
  lion: 96.34%
  tiger: 1.04%
  camel: 0.28%
  lawn_mower: 0.26%
  leopard: 0.26%
```

CLIP은 사자를 찾았고, 이는 트랜스포머 아키텍처의 유연성을 보여준다.

다음 셀에는 클래스가 표시된다.

```
cifar100.classes
```

클래스를 살펴보면 클래스당 하나의 라벨만 사용할 수 있다는 제한이 있지만 CLIP이 잘 찾은 것을 알 수 있다.

```
[...,'kangaroo','keyboard','lamp','lawn_mower','leopard','lion', 'lizard', ...]
```

노트북에는 CLIP의 아키텍처와 설정을 설명하는 몇 가지 다른 셀이 포함되어 있다.

model 셀은 ViT 모델처럼 컨볼루션 임베딩으로 시작하여 멀티 헤드 어텐션이 있는 “표준” 크기(768)의 트랜스포머로 이어지는 시각 인코더를 확인할 수 있다.

```
CLIP(
  (visual): VisionTransformer(
    (conv1): Conv2d(3, 768, kernel_size=(32, 32), stride=(32, 32), bias=False)
    (ln_pre): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
    (transformer): Transformer(
      (resblocks): Sequential(
        (0): ResidualAttentionBlock(
          (attn): MultiheadAttention(
            (out_proj): NonDynamicallyQuantizableLinear(in_features=768,
out_features=768, bias=True)
          )
          (ln_1): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
          (mlp): Sequential(
            (c_fc): Linear(in_features=768, out_features=3072, bias=True)
            (gelu): QuickGELU()
            (c_proj): Linear(in_features=3072, out_features=768, bias=True)
          )
          (ln_2): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        )
      )
    )
  )
)
```

model 셀의 또 다른 흥미로운 측면은 이미지 인코더와 공동으로 실행되는 512 크기의 텍스트 인코더이다.

```

(transformer): Transformer(
  (resblocks): Sequential(
    (0): ResidualAttentionBlock(
      (attn): MultiheadAttention(
        (out_proj): NonDynamicallyQuantizableLinear(in_features=512, out_
features=512, bias=True)
      )
      (ln_1): LayerNorm((512.), eps=1e-05, elementwise_affine=True)
      (mlp): Sequential(
        (c_fc): Linear(in_features=512, out_features=2048, bias=True)
        (gelu): QuickGELU()
        (c_proj): Linear(in_features=2048, out_features=512, bias=True)
      )
      (ln_2): LayerNorm((512.), eps=1e-05, elementwise_affine=True)
    )
  )
)

```

아키텍처, 설정, 파라미터를 설명하는 셀을 살펴보고 CLIP이 어떻게 데이터를 표현하는지 확인하자.

범용 트랜스포머 모델이 이미지-텍스트 쌍을 텍스트-텍스트 쌍으로 처리한다는 것을 보였다. 음악-텍스트, 소리-텍스트, 음악-이미지 및 모든 유형의 데이터 쌍에 범용 모델을 적용할 수 있다.

이제 이미지와 텍스트를 처리할 수 있는 또 다른 범용 트랜스포머 모델인 DALL-E에 대해 살펴보자.

15.4.3 DALL-E

DALL-E는 CLIP과 마찬가지로 범용 모델이다. CLIP은 텍스트-이미지 쌍을 사용했다. DALL-E는 텍스트와 이미지 토큰을 다르게 처리한다. DALL-E의 입력은 토큰 1,280개로 구성된 텍스트와 이미지의 단일 스트림이다. 토큰 256개는 텍스트에, 토큰 1,024개는 이미지에 사용된다. DALL-E는 CLIP처럼 파운데이션 모델이다.

DALL-E는 Salvador Dali와 픽사(Pixar)의 WALL-E에서 이름을 따왔다. DALL-E는 텍스트 프롬프트(prompt)를 입력하고 이미지를 생성하는 방식으로 사용된다. 먼저 DALL-E는 텍스트로 이미지를 생성하는 방법을 학습해야 한다.

DALL-E는 파라미터 120억 개를 가진 GPT-3 버전이다.

이 트랜스포머는 텍스트-이미지 쌍의 데이터셋을 사용하여 텍스트 설명으로부터 이미지를 생성한다.

15.4.3.1 DALL-E의 기본 아키텍처

그림 15.14에서 볼 수 있듯이, CLIP과 달리 DALL-E는 BPE 인코딩된 텍스트 토큰 최대 256개를 이미지 토큰 $32 \times 32 = 1,024$ 개와 연결한다.

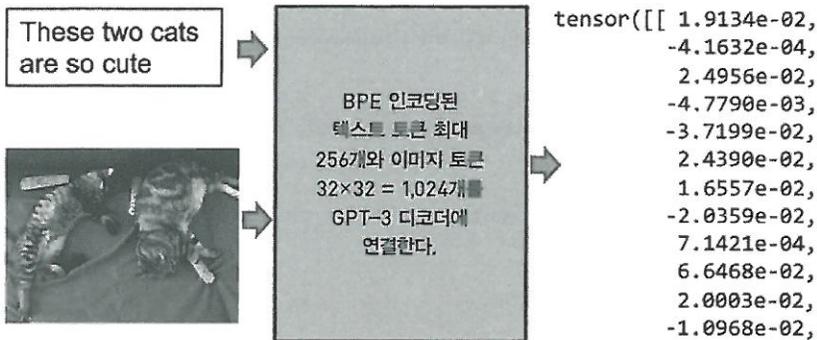


그림 15.14 텍스트 및 이미지 입력을 연결한 DALL-E

그림 15.14는 고양이 이미지가 입력 텍스트와 연결된 것을 보여준다.

DALL-E에는 인코더와 디코더 스택이 있으며, 트랜스포머 모델에 컨볼루션 함수를 적용한 하이브리드 아키텍처로 구축되어 있다. 코드로 모델이 어떻게 작동하는지 살펴보자.

15.4.3.2 코드로 보는 DALL-E

이 섹션에서는 DALL-E가 이미지를 재구성하는 방법을 살펴본다.

NOTE

구글 코랩에서 DALL-E를 실행하기 전에 런타임(Runtime) 메뉴의 세션 관리(Manage sessions)에서 모든 활성 세션을 종료하자. 그런 다음 셀 하나하나 실행해 DALL-E를 실행해보자.
이 장의 깃허브 저장소에서 DALL-E.ipynb를 사용해 독립적으로 실행해볼 수도 있다.

Vision_Transformers.ipynb를 열고, 노트북의 DALL-E 셀로 이동한다. 노트북은 먼저 OpenAI DALL-E를 설치한다.

```
!pip install DALL-E
```

노트북이 이미지를 다운로드하고 처리한다.

```
import io
import os, sys
import requests
import PIL

import torch
import torchvision.transforms as T
import torchvision.transforms.functional as TF

from dall_e import map_pixels, unmap_pixels, load_model
from IPython.display import display, display_markdown

target_image_size = 256

def download_image(url):
    resp = requests.get(url)
    resp.raise_for_status()
    return PIL.Image.open(io.BytesIO(resp.content))

def preprocess(img):
    s = min(img.size)
    if s < target_image_size:
        raise ValueError(f'min dim for image {s} < {target_image_size}')
    r = target_image_size / s
    s = (round(r * img.size[1]), round(r * img.size[0]))
    img = TF.resize(img, s, interpolation=PIL.Image.LANCZOS)
    img = TF.center_crop(img, output_size=2 * [target_image_size])
    img = torch.unsqueeze(T.ToTensor()(img), 0)
    return map_pixels(img)
```

이제 프로그램이 OpenAI DALL-E 인코더와 디코더를 불러온다.

```
# GPU로 변경 가능(예를 들면 'cuda:0')
dev = torch.device('cpu')

# 모델을 더 빠르게 불러오려면 이 파일을 로컬에 다운로드하고 경로를 로컬 경로로 변경하세요.
enc = load_model("https://cdn.openai.com/dall-e/encoder.pkl", dev)
dec = load_model("https://cdn.openai.com/dall-e/decoder.pkl", dev)
```

인코더와 디코더 블록을 살펴보고 하이브리드 모델이 어떻게 작동하는지 확인할 수 있도록 enc 및 dec 셀을 추가했다.

이 섹션에서 처리된 mycat.jpg 이미지는 이 책 깃허브 저장소에서 Chapter15 디렉터리에서 다운로드 할 수 있다.

```
x=preprocess(download_image('https://github.com/Denis2054/AI_Educational/
blob/master/mycat.jpg?raw=true'))
```

마지막으로 원본 이미지를 표시한다.

```
display_markdown('Original image:')
display(T.ToPILImage(mode='RGB')(x[0]))
```

이미지가 출력된다.



그림 15.15 고양이 이미지

이제 프로그램이 재구성된 이미지를 처리하고 표시한다.

```

import torch.nn.functional as F

z_logits = enc(x)
z = torch.argmax(z_logits, axis=1)
z = F.one_hot(z, num_classes=enc_vocab_size).permute(0, 3, 1, 2).float()

x_stats = dec(z).float()
x_rec = unmap_pixels(torch.sigmoid(x_stats[:, :3]))
x_rec = T.ToPILImage(mode='RGB')(x_rec[0])

display_markdown('Reconstructed image:')
display(x_rec)

```

재구성된 이미지는 원본과 매우 유사해 보인다.



그림 15.16 고양이 이미지를 재구성한 DALL-E

결과는 인상적이다. DALL-E는 스스로 이미지를 생성하는 방법을 배웠다.

이 책을 집필할 당시에는 DALL-E의 전체 소스 코드가 제공되지 않았으며 앞으로도 제공되지 않을 수도 있다. 텍스트 프롬프트에서 이미지를 생성하는 OpenAI API는 아직 온라인에 공개되지 않았다. 하지만 눈을 크게 뜨고 지켜보자!

그동안 OpenAI에서 DALL-E를 계속 사용할 수 있다.(<https://openai.com/index/dall-e>)

페이지를 접근하여 예제까지 아래로 스크롤한다. 예시로 샌프란시스코의 알라모 광장 사진을 프롬프트로 선택했다.

TEXT PROMPT a photo of alamo square, san francisco, from a street at night

그림 15.17 샌프란시스코의 알라모 광장 프롬프트

그런 다음 “밤에”를 “아침에”로 수정했다.

```
at night
at night
in the afternoon
in the morning
```

그림 15.18 프롬프트 수정하기

그러자 DALL-E가 수많은 text2image 이미지를 생성했다.



그림 15.19 텍스트 프롬프트에서 이미지 생성하기

우리는 세 가지 비전 트랜스포머인 ViT, CLIP, DALL-E를 구현했다. 마무리하기 전에 마지막으로 몇 가지 생각을 정리해 보자.

15.5 확장되는 모델 세계

새로운 스마트폰처럼 새로운 트랜스포머 모델이 거의 매주 등장한다. 일부 모델은 프로젝트 관리자에게 놀람기도 하고 도전과제를 제시하기도 한다.

- **ERNIE**: 지속적으로 사전 학습하여 언어 이해에 대한 인상적인 결과를 도출하는 프레임워크.

논문: <https://arxiv.org/abs/1907.12412>

도전 과제: 허깅페이스에서 모델을 제공한다. 완전히 학습된 모델인가? SuperGLUE 순위표에서 인간 기준 성능을 뛰어넘도록 바이두(Baidu)가 학습시킨 모델인가? (2021년 12월) <https://super.gluebenchmark.com/leaderboard> 가장 좋은 모델인가 아니면 그냥 장난감 모델인가? 이렇게 작은 버전의 모델에서 AutoML을 실행하는 목적은 무엇인가? 바이두 플랫폼이나 유사한 플랫폼에서 접근할 수 있나? 비용은 얼마나 들까?

- **SWITCH**: 희소(sparse) 모델링으로 최적화된 파라미터 1조 개 모델

논문: <https://arxiv.org/abs/2101.03961>

도전 과제: 논문은 환상적이다. 모델은 어디에 있나? 실제 완전히 학습된 모델을 사용할 수 있나? 비용은 얼마나 들까?

- **Megatron-Turing**: 파라미터 5,000억 개 트랜스포머 모델.

블로그: <https://developer.nvidia.com/blog/using-deepspeed-and-megatron-to-train-megatron-turing-nlg-530b-the-worlds-largest-and-most-powerful-generative-language-model/>

도전 과제: 시장에서 가장 우수한 모델 중 하나다. API를 통해 접근할 수 있나? 본격적인 모델이 될까? 비용은 얼마나 들까?

- **XLNET**은 BERT처럼 사전 학습되지만, 저자들은 이 모델이 BERT 모델 성능을 뛰어넘는다고 주장한다.

논문: <https://proceedings.neurips.cc/paper/2019/file/dc6a7e655d7e5840e66733e9ee67cc69-Paper.pdf>

도전 과제: XLNET이 실제로 구글이 사용하는 버전인 구글 BERT의 성능을 뛰어넘을 수 있을까? 구글 BERT 또는 XLNET 모델 중 가장 우수한 버전에 접근할 수 있을까?

목록은 끝이 없으며 계속 늘어나고 있다!

이들 모두를 테스트하는 것은 앞서 언급한 문제 외에도 여전히 어려운 과제이다. 몇 가지 트랜스포머 모델만이 파운데이션 모델로서 자격이 있다. 파운데이션 모델은 반드시 다음과 같아야 한다.

- 다양한 업무에 대해 충분히 학습해야 한다.
- 독보적인 수준의 NLU로 학습되지 않은 작업도 수행할 수 있어야 한다.
- OpenAI GPT-3처럼 정확한 결과를 합리적으로 보장할 만큼 충분히 큰 규모여야 한다.

많은 사이트에서 교육 목적으로는 유용하지만, 벤치마킹에 적합할 만큼 충분히 학습되고 규모가 크다고 볼 수 없는 트랜스포머를 제공한다.

가장 좋은 방법은 트랜스포머 모델을 최대한 깊게 이해하는 것이다. 어느 순간 전문가가 되어, 첨단 기술 혁신의 정글에서 길을 찾는 것이 스마트폰을 선택하는 것만큼이나 쉬워질 것이다!

15.6 정리하기

새로운 트랜스포머 모델이 계속 시장에 출시되고 있다. 따라서 간행물과 서적을 읽고 일부 시스템을 테스트하여 최신 연구를 따라잡는 것이 좋다.

이를 통해 어떤 트랜스포머 모델을 선택하고 어떻게 구현할지 평가할 수 있다. 시장에 출시되는 모든 모델을 몇 달씩 들여다볼 수는 없다. 프로젝트가 운영 중이라면 매달 트랜스포머 모델을 변경할 수도 없다. 4차 산업 시대는 끊임없는 API 생태계로 향하고 있다.

모든 모델을 학습하는 것은 불가능하다. 하지만 트랜스포머 모델에 대한 지식을 고도화하면 새로운 모델을 빠르게 이해할 수 있다.

트랜스포머 모델의 기본 구조는 변경되지 않는다. 인코더 또는 디코더 스택 층은 동일하게 유지된다. 어텐션 헤드를 병렬화하여 계산 속도를 최적화할 수 있다.

리포머 모델은 LSH 버킷과 청크화를 적용한다. 또한 정보를 저장하는 대신 각 층의 입력을 다시 계산하여 메모리 문제를 최적화한다. 그러나 GPT-3처럼 파라미터 10억 개를 사용하는 모델도 동일한 예제에 대해 수용할 만한 결과를 생성한다.

DeBERTa 모델은 콘텐츠와 위치를 풀어서 학습 프로세스를 더욱 유연하게 만든다. 결과는 인상적이다. 그러나 GPT-3와 같은 파라미터 10억 개를 사용하는 모델도 DeBERTa의 결과와 동일할 수 있다.

ViT, CLIP, DALL-E는 범용 텍스트-이미지 비전 트랜스포머 모델의 매력적인 세계를 보여준다. 언어와 이미지의 결합은 새롭고 생산적인 정보를 만들어낸다.

이제 바로 사용할 수 있는 AI와 자동화된 시스템이 어디까지 발전할 수 있을지가 문제다. 다음 장에서는 메타휴먼의 부상에 따른 트랜스포머 기반 AI의 미래를 시각화해 본다.

15.7 문제

01. 리포머 트랜스포머 모델에는 인코더가 포함되지 않는다. 참 / 거짓
02. 리포머 트랜스포머 모델에는 디코더가 포함되지 않는다. 참 / 거짓
03. 리포머 모델에서 입력은 층별로 저장된다. 참 / 거짓
04. DeBERTa 트랜스포머 모델은 콘텐츠와 위치 정보에 풀기(disentangle)를 적용했다. 참 / 거짓
05. 프로젝트에 사용할 트랜스포머 모델을 선택하기 전에 사전 학습된 트랜스포머 모델 수백 개를 테스트해야 한다. 참 / 거짓
06. 최신 트랜스포머 모델이 항상 가장 좋다. 참 / 거짓
07. 멀티태스크 트랜스포머 모델 한 개 보다 NLP 작업마다 하나의 트랜스포머 모델을 사용하는 것이 좋다. 참 / 거짓
08. 트랜스포머 모델은 항상 미세 조정해야 한다. 참 / 거짓
09. OpenAI GPT-3 엔진은 미세 조정 없이도 광범위한 NLP 작업을 수행할 수 있다. 참 / 거짓
10. AI 알고리즘은 항상 로컬 서버에서 구현하는 것이 좋다. 참 / 거짓

15.8 참고 문헌

- 허깅페이스 리포머: https://huggingface.co/transformers/model_doc/reformer.html?highlight=reformer
- 허깅페이스 DeBERTa: https://huggingface.co/transformers/model_doc/deberta.html
- Pengcheng He, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, 2020, Decoding-enhanced BERT with Disentangled Attention: <https://arxiv.org/abs/2006.03654>
- Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby, 2020, An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale: <https://arxiv.org/abs/2010.11929>
- OpenAI: <https://openai.com/>
- William Fedus, Barret Zoph, Noam Shazeer, 2021, Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity: <https://arxiv.org/abs/2101.03961>
- Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, Ilya Sutskever, 2021, Learning Transferable Visual Models From Natural Language Supervision: <https://arxiv.org/abs/2103.00020>
- C7LIP: <https://github.com/openai/CLIP>

16장

트랜스포머 기반 코파일럿의 등장

- 16.1 프롬프트 엔지니어링
- 16.2 코파일럿
- 16.3 도메인별 GPT-3 엔진
- 16.4 컴퓨터 비전
- 16.5 메타버스에서 인간과 AI 코파일럿
- 16.6 정리하기
- 16.7 문제
- 16.8 참고 문헌

4차 산업 시대가 본격화되면서 모든 것이 기계간 연결, 커뮤니케이션, 의사 결정에 관련될 것이다. AI는 주로 즉시 사용 가능한(ready-to-use) 종량제 (pay-as-you-go) 클라우드 AI 솔루션에 내장될 것이다. 빅테크 기업은 재능 있는 AI 전문가를 채용해 API, 인터페이스, 통합 도구를 만들 것이다.

AI 전문가는 개발부터 설계까지 다루며 설계자, 시스템 통합 담당자, 클라우드 AI 파이프라인 관리자의 역할을 맡을 것이다. 따라서 엔지니어 개발자보다 엔지니어 컨설턴트에게 더 많은 AI 일자리가 생기고 있다.

1장, 트랜스포머란 무엇인가?에서 학습하지 않은 NLP 작업을 수행할 수 있는 파운데이션 모델을 소개했다. **15장, NLP부터 범용 트랜스포머 모델까지**에서 파운데이션 트랜스포머 모델을 사용해 이미지 작업, NLP 작업 등을 수행하는 범용 모델로 확장했다.

이 장에서는 범용 OpenAI GPT-3 모델을 코파일럿(Copilot) 작업으로 확장할 것이다. 새로운 세대의 AI 전문가와 데이터 과학자가 AI 코파일럿을 사용해 소스 코드를 자동으로 생성하고 의사 결정을 내리는 방법을 다뤄볼 것이다.

이 장에서는 프롬프트 엔지니어링에 대해 자세히 살펴보는 것으로 시작할 것이다. 예제는 회의록을 요약하는 작업으로 구성되어 있다. 트랜스포머는 생산성을 높여주지만, NLP는 여전히 AI에게 어려운 과제를 맡을 수 있다.

OpenAI 코덱스(Codex)를 사용해 코파일럿 사용 방법을 익힐 것이다. 개발자가 소스 코드를 작성할 때, 깃허브 코파일럿은 코덱스를 사용해 소스 코드를 제한한다. 코덱스는 자연어를 소스 코드로 변환할 수도 있다.

도메인별 GPT-3 엔진을 사용해 새로운 AI 기법을 알아볼 것이다. 이 장에서는 12,288차원 임베딩을 생성하고 이를 머신러닝 알고리즘에 사용하는 방법을 다뤄볼 것이다. 또한 트랜스포머가 자동으로 지시(instruction)를 생성하는 방법도 살펴볼 것이다.

편향된 입력과 출력을 필터링하는 방법을 살펴본 후 트랜스포머 기반 추천 엔진을 살펴볼 것이다. 2020년대 AI는 윤리적인 방법으로 구축되어야 한다.

추천 시스템은 모든 소셜 미디어 플랫폼에 스며들어 동영상, 게시물, 메시지, 책 등 소비할 다양한 상품을 추천한다. 교육용 다목적 트랜스포머 기반의 추천 시스템을 머신러닝 사용해 구축해볼 것이다.

트랜스포머 모델은 시퀀스를 분석한다. NLP에서 시작했지만 컴퓨터 비전까지 성공적으로 확장할 수 있었다. JAX로 개발한 트랜스포머 기반 컴퓨터 비전 프로그램을 살펴볼 것이다.⁴⁴

마지막으로, 가상 시스템을 메타버스(metaverse)로 전환할 때 AI 코파일럿이 사용되는 부분을 살펴볼 것이며, 이는 10년 내에 확산될 것이다. 애플리케이션을 개발할 때, 소스 코드를 작성하는 개발자가 파일럿(pilot)이다. IDE에서 자동 완성을 활성화하면 코드 전체 줄이 아닌 메소드 단위로 자동 완성이 된다. 코파일럿(copilot)을 사용하면 코드 전체 단락 단위로 자동 완성을 사용할 수 있다!

이 장에서는 다음의 주제들을 다룬다.

- 프롬프트 엔지니어링
- 깃허브 코파일럿
- 코덱스를 사용해 자연어를 소스 코드로 변환하는 모델
- 임베딩 데이터셋
- 임베디드 기반 머신러닝
- 지시 시리즈(instruction series)
- 콘텐츠 필터 모델
- 트랜스포머 기반 추천 엔진 살펴보기
- NLP 시퀀스 학습(NLP sequence learning)을 행동 예측으로 확장하기
- JAX로 트랜스포머 모델 구현하기
- 트랜스포머 모델을 컴퓨터 비전에 사용하기

알아야 할 핵심 기술인 프롬프트 엔지니어링부터 시작하자.

⁴⁴ 역주. JAX는 구글이 개발한 고성능 딥러닝과 과학 계산을 위한 라이브러리이다. 넘파이와 비슷한 API를 제공하며 딥러닝 모델을 구축하고 훈련하는 데 사용한다. 텐서플로우, 파이토치 등과 같은 다른 딥러닝 프레임워크와 비슷한 기능을 제공하지만, 몇 가지 차이점이 존재한다. 가장 주목할만한 차이점 중 하나는 JAX가 자동 미분(automatic differentiation)을 GPU에서 사용해 딥러닝 모델 학습 속도를 크게 향상시킬 수 있다는 점이다.

16.1 프롬프트 엔지니어링

특정 언어를 구사하는 것은 유전이 아니다. 인간의 뇌에 부모의 언어를 저장하는 영역이 따로 존재하지 않다. 인간의 뇌는 언어를 말하고, 읽고, 쓰고, 이해할 수 있도록 어릴 때부터 뉴런을 설계한다. 사람마다 문화적 배경과 어린 시절의 의사소통 방식에 따라 각기 다른 언어 신경 회로를 갖고 있다.

사람은 성장하면서 듣는 언어 대부분이 미완성 문장, 문법 오류, 단어 오용, 잘못된 발음, 기타 여러 가지 왜곡된 언어 등의 형태로 엉망임을 알게 된다.

사람은 언어를 사용해 메시지를 전달한다. 상대방이나 청중에 맞게 언어를 조정해야 한다는 것을 깨닫는다. 원하는 결과(“출력”)를 얻기 위해 “입력” 혹은 “프롬프트”를 추가해야 할 수도 있다. GPT-3와 같은 파운데이션 트랜스포머 모델은 수백 가지 작업을 무한한 방식으로 수행할 수 있다. 다른 언어들과 마찬가지로 트랜스포머 모델을 사용함에 있어서 프롬프트와 응답을 익혀야 한다. 사람에 가까운 수준의 트랜스포머와 효과적으로 소통해 최대 결과를 얻기 위해서는 최소한의 정보를 제공해야 한다. 원하는 결과를 얻기 위해 제공한 최소 입력 정보를 minI로, 최대 결과를 maxR로 표기하도록 하자.

커뮤니케이션을 다음과 같이 표현할 수 있다.

$$\text{minI}(\text{input}) \rightarrow \text{maxR}(\text{output})$$

이를 트랜스포머와의 대화로 나타내기 위해 “input”을 “prompt”로, “output”을 “response”로 바꿔서, 입력한 프롬프트가 트랜스포머의 반응에 영향을 미친다는 것을 나타낼 것이다. 트랜스포머와의 대화 퀄리티 $d(T)$ 는 다음과 같이 나타낼 수 있다.

$$d(T) = \text{minI}(\text{prompt}) \rightarrow \text{maxR}(\text{response})$$

어떤 결과를 얻기 위해, 입력한 프롬프트가 최소 입력 정보일 확률이 1에 가까울수록, 최대 결과를 얻을 확률도 1에 가까워진다.

입력한 프롬프트가 최소 입력 정보일 확률이 0에 가까울수록, 최대 결과를 얻을 확률도 0에 가까워진다.

입력한 프롬프트는 이러한 확률에 영향을 끼친다! 왜냐하면 트랜스포머는 입력 프롬프트와 응답을 다

음 응답 생성하는 데에 사용하기 때문이다.

어렸을 때나 성인이 되어서도 언어를 배우는 데는 수년이 걸린다. 트랜스포머와 소통하는 언어인, $\text{minI}(\text{prompt})$ 를 효과적으로 설계하는 방법을 배우는 데도 상당한 시간이 필요하다. 트랜스포머와 그 아키텍처, 알고리즘이 계산을 하는 방식을 이해해야 한다. 그런 다음 트랜스포머가 예상대로 작동하도록 입력, 즉 프롬프트를 설계하는 방법을 이해하는 데 상당한 시간을 할애해야 한다.

이 섹션에서는 대화체를 중점적으로 다룬다. NLP 작업을 위한 OpenAI GPT-3의 프롬프트는 회의록이나 대화록에서 입력을 가져오는 경우가 많은데, 이는 구조화되지 않은 경우가 많다. 회의록이나 대화록을 요약하는 경우 상당히 어려울 수 있다. 이 섹션에서는 일상적인 영어부터 제한된 컨텍스트를 가진 일상적 또는 공식적인 영어까지 일곱 가지 상황에서 동일한 대화록을 요약하는 데 초점을 맞출 것이다.

유의미한 컨텍스트가 있는 일상적인 영어부터 시작하자.

16.1.1 유의미한 컨텍스트가 있는 일상 영어

일상 영어는 문장이 짧고 제한된 어휘를 사용한다.

OpenAI GPT-3에 “요약” 작업을 실행해 보자. www.openai.com에 접속해 회원 가입하고 로그인을 해보자.

Jane과 Tom의 일상 대화를 요약하는 데에 필요한 정보를 GPT-3에 입력해 보자. Jane과 Tom은 일을 시작하는 두 명의 개발자이다. Tom은 Jane에게 커피를 제안하고 Jane은 이를 거절하는 대화이다.

이 경우 그림 16.1과 같이 입력 정보가 충분하므로 $\text{minI}(\text{prompt})=1$ 이 된다.

Convert my shorthand into a first-hand account of the meeting:

We get used to hearing dialogs that only people that know each other well understand. Consider the following dialog between Jane and Tom, two developers, mumbling their way through the day while they are getting down to work in an open space:

Tom: "hi"

Jane: "yeah sure"

Tom: "Coffee?"

Jane: "Nope"

Tom: "Cool. You're trying then."

Jane: "Yup"

Tom: "My wife stopped too a few months ago."

Jane: "Right. She got it."

Tom: "Sleep better?"

Jane: "Yeah. Sure."

Tom: "I told you. Drinking too much of that"

Summary:

Generate



내가 메모한 것을 회의 기록으로 변환해:

우리는 서로를 잘 아는 사람들만 이해할 수 있는 대화를 듣는 데 익숙해져 있다. 제인과 톰, 두 개발자가 오픈 스페이스에서 일하면서 하루 종일 중얼거리며 대화하는 다음 대화를 고려해 보자.

톰: "안녕"

제인: "안녕"

톰: "커피?"

제인: "괜찮아"

톰: "좋네. 커피 안 마시려고 하고 있구나."

제인: "응"

톰: "내 아내도 몇 달 전부터 커피를 안 마시기 시작했어"

제인: "그렇군요. 그녀가 해냈구나."

톰: "잠이 더 잘와!"

제인: "응, 그럼"

톰: "내가 말했잖아. 너무 많이 마시는 것 같다고."

요약:

그림 16.1 잘 작성된 대화 요약하기

Generate를 클릭하면 그림 16.2처럼 놀라운 결과를 얻는다.

Summary:

Tom and Jane are two developers at a company that are getting started for the day. They are both drinking coffee. Tom asks Jane if she wants any coffee or if she has tried giving it up. Jane says she has, and that she is feeling better. Tom's wife also quit coffee and Tom asks if Jane slept



요약: 한 회사의 두 개발자인 톰과 제인은 하루 일과를 시작하고 있다. 둘 다 커피를 마시고 있다. 톰은 제인에게 커피를 더 마실 수 있는지, 아니면 커피를 끊으려고 노력하는 중인지 묻고 있다. 제인은 커피를 끊으려고 노력하고 있고 기분이 나아졌다 말하고 있다. 톰의 아내도 커피를 끊었고 톰은 제인이 잠을 잘 잤는지 묻고 있다.

그림 16.2 GPT-3가 생성한 수용할 만한 요약

AI가 혼란스러운 일상 대화, 회의, 목적 없는 수다에서 구조를 찾아낼 수 있다고 결론을 지을 수 있을까? 이 질문에 대답하는게 쉽지 않다. 환유⁴⁵를 추가해 입력을 좀 더 복잡하게 만들어보자.

16.1.2 환유를 사용한 일상 영어

Tom은 단어 coffee를 사용했다. 하지만 Tom이 coffee 대신에 “java”를 사용하면 어떻게 될까? coffee는 음료를 가리키지만, java는 자바 섬(Java Island)에서 유래한 커피 재료를 가리킨다. 환유는 coffee 대신 java를 사용한 것처럼 대상의 속성을 사용하는 비유의 한 종류이다. java는 커피 아이콘을 가진 프로그래밍 언어이기도 하다.

coffee 대신에 java를 사용하는 경우, GPT-3는 다의성 문제를 직면하게 된다. 왜냐하면 java는 환유를 사용한 경우로 커피를 의미하는 커피의 성분을 뜻하거나, Java 섬을 뜻하거나, 프로그래밍 언어 중 하나를 뜻할 수도 있기 때문이다.

인간은 다의성에 익숙하다. 단어의 다양한 의미를 배우며, 컨텍스트가 없이 단어 하나만으로는 큰 의미가 없다는 것을 알고 있다. Jane과 Tom의 예제에서는 두 명이 개발자이기 때문에 상황이 더 복잡해진다. 둘은 커피에 대해 이야기하는 것일까 아니면 프로그래밍 언어에 대해 이야기하는 것일까?

⁴⁵ 역주. 어떤 낱말 대신에 그것을 연상시키는 다른 낱말을 쓰는 비유의 한 종류. 예를 들어 '미국 대통령' 대신에 '백악관'을 사용하는 것과 같은 경우이다.

Tom이 커피를 끊은 아내에 대해 이야기하고 있기 때문에 인간이라면 쉽게 대답할 수 있다.

단어 java가 단어 coffee를 대체할 때, GPT-3는 다의성 때문에 혼동해 오답을 생성할 수 있다.

Summary:

Tom asked Jane if she wanted to work on Java and she declined. He asked if she wanted to work on it and she said she would, then he told her that his wife stopped drinking it and said she was sleeping better. Then Jane said she was, too.

요약: 톰은 제인에게 그녀가 Java로 일하고 싶은지 물었다. 그녀는 거절했다. 그는 그녀가 이 작업을 하고싶은지 물었고 그녀는 그렇다고 답했다. 그 다음 그가 그녀에게 그의 아내가 커피를 마시는 것을 그만뒀다고 말했다. 그리고 그녀가 더 잘 잔다고 답했다. 그다음 제인도 그렇다고 답했다.

그림 16.3 프롬프트가 혼란을 야기할 때 GPT-3의 틀린 응답

따라서 $\min I(\text{prompt})$ 가 0에 가까울 때, $\max R(\text{response})$ 일 확률은 0에 가까워진다는 것을 확인할 수 있다.

인간의 대화에 생략된 표현이 있으면 분석하기가 훨씬 더 어려워질 수 있다.

16.1.3 생략이 있는 일상 영어

상황은 더 복잡해질 수 있다. Tom이 커피 한 잔을 마시고 있는데 Jane이 Tom을 바라보며 커피 잔을 바라보며 자연스럽게 인사를 건네는 상황을 생각해 보자.

Tom은 Jane에게 커피나 자바를 원하냐고 묻는 대신 이렇게 말한다.

"want some?"

Tom의 대화에서 coffee를 생략했다. Jane은 Tom이 커피 잔을 들고 있는 모습만 봐도 Tom이 무슨 의미로 얘기 했는지 이해할 수 있다.

OpenAI GPT-3은 단어 drinking을 감지하고 이 동사를 질문 Want some?과 연관을 짓는다.

프로그래밍 언어로 이해하지 않기를 원하는 상황에서, GPT-3가 생성한 다음 요약은 정확하다는 것을 알 수 있다.

Summary:

Tom and Jane are two developers that are talking to each other about their day. Jane doesn't drink coffee, but Tom is giving it away for free. Tom's wife also stopped drinking coffee, and is sleeping better. Tom also tells Jane that he told her that she was drinking too much coffee.

요약: 톰과 제인은 그들의 하루에 관해 이야기를 나누고 있는 두 명의 개발자다. 제인은 커피를 마시지 않는다. 그러나 톰은 공짜로 나눠주고 있다. 톰의 아내 또한 커피를 마시는 것을 그만뒀다. 그리고 더 잘 자고 있다. 톰은 또한 제인에게 그가 그녀에게 커피를 너무 많이 마신다고 말했던 것에 대해 말하고 있다.

그림 16.4 GPT-3가 생성한 정확한 응답

인간은 쉽게 이해할 수 있지만 AI에게는 어려울 수 있는, 모호한 컨텍스트가 있는 경우를 살펴보자.

16.1.4 모호한 컨텍스트가 있는 일상 영어

Tom이 커피 잔을 들고 있다는 점에서, Tom이 무슨 말을 하는지 Jane이 이해하기 위해 Tom이 Jane에게 아내를 언급할 필요가 없어진다.

Tom이 아내를 언급하는 부분과 동사 drinking을 제거하자. coffee나 java 대신 want some은 그대로 두자.

Tom: "hi"
Jane: "yeah sure"
Tom: "Want some?"
Jane: "Nope"
Tom: "Cool. You're trying then."
Jane: "Yup"
Tom: "Sleep better?"
Jane: "Yeah. Sure."

톰: "안녕"
 제인: "안녕"
 톰: "한 잔 줄까?"
 제인: "아니"
 톰: "좋네, 커피 안 마시려고 하고 있구나."
 제인: "응"
 톰: "잠이 더 잘와?"
 제인: "응, 그럼"

그림 16.5 모호한 컨텍스트

혼란스러운 대화가 출력에 반영됐다.

Summary: Tom and Jane are discussing the meeting they had with the team yesterday. Tom is telling Jane that she's in charge of getting the team to try to sleep better, and Jane is telling Tom that she slept well.

요약: 톰과 제인은 어제 했던 팀 회의에 대해 논의하고 있다. 톰은 제인에게 그녀가 그 팀이 더 잘 자도록 하는 것을 담당하고 있다고 얘기하고 있다. 그리고 제인은 톰에게 그녀가 잘 잤다고 얘기하고 있다.

그림 16.6 좋지 못한 GPT-3의 응답

프롬프트가 너무 모호해서 부적절한 응답을 야기했다. 따라서 다음과 같이 요약이 가능하다.

$\min I(\text{prompt})$ 가 0에 가까워질 때, $\max R(\text{response})$ 의 확률은 0에 가까워지므로 $d(T)$ 또한 0에 가까워진다.

인간은 의사소통할 때 상대방의 문화, 과거 관계, 시각적 상황 및 기타 보이지 않는 요소들을 대화에 반영한다. 제3자에게 보이지 않는 요소들은 다음이 있다.

- 상대방의 모습(행동, 표정, 몸짓 등)을 보지 않고 텍스트 읽기
- 제3자는 모르나 대화 참여자끼리는 서로 알고 있는 것에 대해 듣기
- 제3자와는 다른 문화에서 발생한 사건들

이러한 요소는 무수히 많다.

이러한 보이지 않는 요소들이 AI의 눈을 가리게 만드는 것을 알 수 있다.

현재 예제에 센서를 도입해 보자.

16.1.5 센서를 사용한 일상 영어

사고 실험을 위해 실내에 비디오 센서를 도입해 보자. 비디오 피드(video feed)⁴⁶와 이미지 캡션(image captioning)⁴⁷을 사용해 다음과 같이 대화 초반에 컨텍스트를 제공할 수 있다고 상상해 보자.

인간은 때때로 서로를 잘 아는 사람들만 이해할 수 있는 대화를 한다. Jane과 Tom의 다음 대화를 생각해 보자. 비디오 피드가 Tom이 커피를 마시고 있고 Jane이 키보드로 타이핑하고 있다는 이미지 캡션을 생성한다고 가정하자. 또한 Jane과 Tom은 두 명의 개발자로, 탁 트인 공간에서 하루 일과를 시작하며 중얼거리고 있다는 캡션을 생성한다고 가정하자.

그리고 다음과 같은 혼란스러운 대화를 프롬프트로 입력한다.

```
Tom: "hi" (안녕)
Jane: "yeah sure" (응, 그래)
Tom: "Want some?" (한 잔 줄까?)
Jane: "Nope" (아니)
Tom: "Cool, You're trying then." (좋네, 커피 안 마시려고 노력하고 있구나.)
Jane: "Yup" (응)
Tom: "Sleep better?" (잠이 더 잘 와?)
Jane: "Yeah, Sure. " (응, 그럼)
```

GPT-3는 처음에 중요한 의미를 갖는 단어를 생략했지만, 다음과 같이 수용할 만한 결과를 출력한다.

요약: 한 개발자가 키보드로 타이핑하는 모습이 보입니다. 다른 개발자가 방으로 들어와 그녀에게 커피 한 잔을 권합니다. 그녀는 거절하지만 그는 고집을 부립니다. 그들은 그녀의 수면과 커피에 대해 이야기합니다.

46 역주. 비디오 피드는 비디오 신호나 스트리밍 데이터로 구성된 실시간 영상을 가리킨다.

47 역주. 이미지 캡션은 이미지의 내용을 텍스트로 설명하는 기술이다. 주어진 이미지를 분석해 이미지에 포함된 객체, 장면, 상황 등을 이해하고 이를 자연어로 표현하는 캡션을 생성한다.

결과는 실행할 때마다 바뀔 수 있다. GPT-3는 상위 확률의 결과를 살펴보고 그 중 가장 좋은 것을 선택한다. 이미지 캡션으로 컨텍스트를 제공했기에 GPT-3가 좋은 결과를 낼 수 있었다.

하지만 Tom이 커피 잔을 들고 있지 않아 GPT-3에게 시각적 컨텍스트를 제공하지 않는다면 어떨까?

16.1.6 센서가 있지만 시각적 컨텍스트가 없는 일상 영어

인공지능에게 가장 어려운 상황은 Tom이 매일 이벤트를 언급하지만 오늘은 언급하지 않는 경우이다. Tom이 매일 아침 커피 잔을 들고 들어온다고 가정해 보자. Tom은 커피를 마시기 전에 Jane에게 커피를 마실 것인지 물어본다. 우리의 사고 실험은 가능한 모든 경우를 상상하는 것이다. 이 경우(Tom이 커피 잔을 들고 오지 않은 오늘) 사고 실험의 비디오 피드에는 아무것도 표시되지 않아 다시 혼란스러워진다. 또한 비디오 피드에는 그들이 개발자, 회계사 또는 컨설턴트인지 알 수 없다. 따라서 컨텍스트에서 해당 부분을 제거하면 다음과 같은 컨텍스트가 남는다.

Tom: "hi" (안녕)

Jane: "yeah sure" (응, 그래)

Tom: "Want some?" (한 잔 줄까?)

Jane: "Nope" (아니)

Tom: "Cool. You're trying then." (좋네. 커피 안 마시려고 노력하고 있구나.)

Jane: "Yup" (응)

Tom: "Sleep better?" (잠이 더 잘 와?)

Jane: "Yeah. Sure. " (응, 그럼)

결과는 놀랍다. GPT-3가 터무니없는 결과를 생성한다. GPT-3는 확률적 알고리즘이다. 입력이 조금만 바뀌어도 전혀 다른 결과가 나올 수 있다. GPT-3는 상대방이 무슨 말을 하는지 추측하려고 한다. GPT-3는 대화가 무언가를 소비하는 내용임을 감지한다. 필자는 GPT-3가 불법 약물에 대한 얘기를 출력해서 이번 섹션에 실지 않았다.

GPT-3는 사용하는 언어의 수준을 결정하고 이를 관련 상황과 연관 짓는다.

같은 실험을 격식 차린 영어(formal English)로 재현한다면 결과는 어떻게 달라질까?

16.1.7 컨텍스트가 없는 격식 차린 영어

컨텍스트를 배제하고 격식 차린 영어를 실험해 보자. 격식 차린 영어는 긴 문장, 문법에 맞는 표현, 말투(manner)를 담고 있다. 컨텍스트는 제외하고 이전 대화를 격식 차린 영어로 바꾸어 보자.

Tom: "Good morning, Jane" (좋은 아침, 제인)
Jane: "Good morning, Tom" (좋은 아침, 톰)
Tom: "Want some as well?" (너도 좀 원해?)
Jane: "No, thank you. I'm fine." (아니, 됐어. 괜찮아)
Tom: "Excellent. You are on the right track!" (훌륭하네. 잘하고 있어!)
Jane: "Yes, I am" (응, 맞아)
Tom: "Do you sleep better these days?" (요즘 잠은 더 잘 자?)
Jane: "Yes, I do. Thank you." (응, 맞아. 고마워)

이 정도의 영어 수준과 말투를 토대로, GPT-3는 Tom이 마시고 있는 것을 파악해서 Jane에게 제안하고 있다는 것을 이해한다. GPT-3 결과는 만족스러운 수준이다.

요약: 톰이 제인에게 "좋은 아침"이라고 인사합니다. 톰은 자신이 마시는 음료수를 제인에게 권합니다. 제인은 "아니, 됐어. 괜찮아"라고 말합니다. 톰은 "훌륭해"라고 말하며 제인이 잘하고 있다고 말합니다. 제인은 "응, 맞아"라고 말합니다. 톰은 제인이 요즘 잠을 더 잘자냐고 묻습니다.

대화에 다른 사람이나 다른 사물을 등장시켜 무궁무진한 상황을 상상할 수 있다.

이제까지의 실험을 요약해 보자.

16.1.8 프롬프트 엔지니어링 학습

인간은 말하고자 할 때, 종종 생각이 혼란스러워 구조화되지 않은 문장을 재구성하기 위해 다양한 방법을 사용한다. 인간은 종종 다른 사람이 말하는 내용을 이해하기 위해 추가 질문을 해야한다. OpenAI GPT-3와 같이 학습된 트랜스포머와 상호 작용할 때도 이러한 점을 감안해야한다.

트랜스포머와의 대화 $d(T)$ 와 응답 $\max R(\text{response})$ 은 입력 $\min I(\text{prompt})$ 의 품질에 따라 달라진다는 점을 유의해야한다.

$$d(T) = \min I(\text{prompt}) \rightarrow \max R(\text{response})$$

입력한 프롬프트가 최소 입력 정보일 확률이 1에 가까울수록, 최대 결과를 얻을 확률도 1에 가까워진다.

입력한 프롬프트가 최소 입력 정보일 확률이 0에 가까울수록, 최대 결과를 얻을 확률도 0에 가까워진다.

프롬프트 엔지니어링을 연습하고 실험하는 것은 좋은 시도이다. 프롬프트 엔지니어링은 AI의 다음 수준으로 나아갈 수 있는 새로운 기술이다.

프롬프트 엔지니어링 사용 능력은 코파일럿(copilots) 사용 능력으로 이어진다.

16.2 코파일럿

비주얼 스튜디오(Visual Studio)에서 사용할 수 있는 OpenAI가 지원하는 AI 기반 개발(AI-driven development)의 세계에 온 것을 환영한다.

16.2.1 깃허브 코파일럿(Github Copilots)

깃허브 코파일럿에 대해 알아보자.

<https://github.com/github/copilot-docs>

이번 섹션에서는 PyCharm(JetBrains)에서 깃허브 코파일럿을 사용할 것이다.

<https://docs.github.com/ko/copilot/getting-started-with-github-copilot>

설명서를 따라 JetBrains를 설치하고 PyCharm에서 OpenAI Github Copilot을 활성화한다.

깃허브 코파일럿은 총 네 가지 단계로 진행된다(그림 16.7을 참고).

- OpenAI 코덱스(Codex)는 인터넷에 공개된 코드와 텍스트를 학습한다.
- 학습된 모델은 깃허브 코파일럿 서비스에 연결된다.
- 깃허브 서비스는 코드 편집기(이 경우 PyCharm)에서 작성한 코드와 OpenAI 코덱스 간 전후 흐름을 관리한다. 깃허브 서비스 관리자(Manager)는 코드 제안을 하고 사용자와의 상호 작용을 다시 깃허브 서비스로 보낸다.
- 코드 편집기는 개발 작업 공간이다.



그림 16.7 깃허브 코파일럿의 네 가지 단계

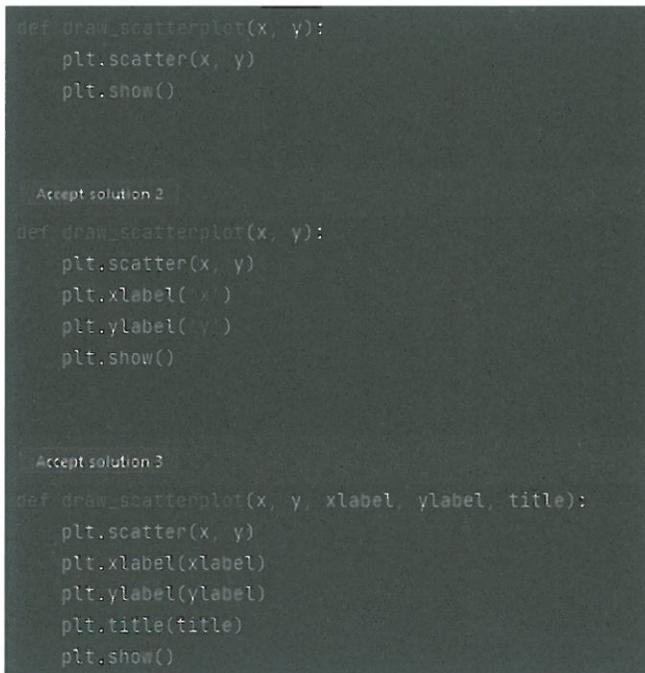
깃허브 코파일럿에서 제공한 설명서를 따라 PyCharm에서 깃허브에 로그인한다.

이때 발생한 문제는 <https://github.com/features/copilot/>의 “Frequently asked questions” 섹션을 참고해 해결할 수 있다.

PyCharm에서 설정을 마치면 다음과 같이 입력해 보자.

```
import matplotlib.pyplot as plt
def draw_scatterplot
```

위 코드를 입력하자마자 이후에 어떤 소스 코드를 입력하면 좋을지 제안을 받을 수 있다.



```
def draw_scatterplot(x, y):
    plt.scatter(x, y)
    plt.show()

Accept solution 2
def draw_scatterplot(x, y):
    plt.scatter(x, y)
    plt.xlabel(x)
    plt.ylabel(y)
    plt.show()

Accept solution 3
def draw_scatterplot(x, y, xlabel, ylabel, title):
    plt.scatter(x, y)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title)
    plt.show()
```

그림 16.8 제안받은 소스 코드들

제안 중에서 원하는 소스 코드를 선택하면 코드 편집기에 표시가 된다. **Tab**을 사용해 제안을 확인할 수 있다. 혹은 기다리면 또 다른 제안을 받을 수도 있다.

```
import matplotlib.pyplot as plt
def draw_scatterplot(x, y):
    plt.scatter(x, y)
```

```
plt.xlabel('x')
plt.ylabel('y')
plt.show()
draw_scatterplot([1, 2, 3, 4, 5], [1, 4, 9, 16, 25])
```

플롯(plot)은 다음과 같이 출력된다.

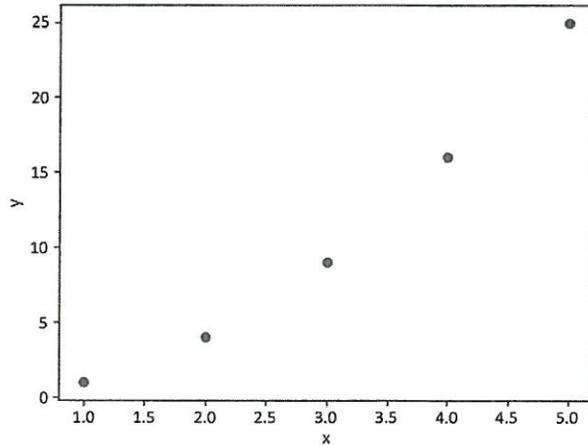


그림 16.9 코파일럿이 제안한 소스 코드로 표시한 플롯

이 책의 깃허브 저장소의 Chapter 16 폴더에 있는 Github_Copilot.py를 실행하여 위 결과를 확인할 수 있다.

이 기술은 사용자에게 직접적으로 보이지 않으며 자연스럽게 모든 개발 영역으로 점진적으로 확장될 것이다. 이 시스템에는 다른 파이프라인과 함께 GPT-3 기능이 포함되어 있다. 파이썬, 자바스크립트 등에서 사용할 수 있다.

OpenAI 코텍스로 구동되는 깃허브 코파일럿 작업에 익숙해지려면 프롬프트 엔지니어링 학습이 필요하다.

코파일럿을 사용해 훈련하기에 좋은 OpenAI 코텍스를 살펴보자.

NOTE

2023년 3월, 마이크로소프트 깃허브 코파일럿은 코텍스를 여전히 지원하고 있다. 3월 23일부터 OpenAI는 코텍스 지원을 중단했다. 대신 GPT-3.5-turbo와 GPT-4 사용을 권장한다. **17장, 초인간 트랜스포머를 사용한 OpenAI의 ChatGPT와 GPT-4에서 GPT-3.5-turbo와 GPT-4를 다룰 것이다.** OpenAI의 도메인별 엔진은 프로젝트의 성능을 향상시킬 수 있는 유용한 결과를 제공한다.

16.3

도메인별 GPT-3 엔진

이 섹션에서는 도메인별 작업을 수행할 수 있는 GPT-3 엔진을 살펴본다. 이 섹션의 세 가지 하위 섹션에서 세 가지 모델을 실행해 볼 것이다.

- Embedding2ML, GPT-3을 사용해 머신러닝 알고리즘에 임베딩을 제공하는 모델
- 모든 작업에 대한 지침을 GPT-3에 요청하는 지시 시리즈(instruct series)
- 허용되지 않는 모든 형태의 입력 및 출력과 편향성(bias)을 필터링하는 콘텐츠 필터

Domain_Specific_GPT3_Functionality.ipynb 파일을 열어보자.

Embedding2ML부터 시작해 보자.

16.3.1 Embedding2ML

OpenAI는 다양한 기능을 가진, 다양한 차원의 여러 임베딩 모델을 학습시켰다.

- Ada (1,024차원)
- Babbage (2,048차원)
- Curie (4,096차원)
- Davinci (12,288차원)

각 엔진에 대한 자세한 설명은 OpenAI 웹사이트에서 확인할 수 있다.

<https://platform.openai.com/docs/guides/embeddings>

Davinci 모델은 12,288차원 임베딩을 제공한다. 이 섹션에서는 Davinci 모델을 사용해 공급망 데이터 셋(SCM, Supply Chain Management)의 임베딩을 생성할 것이다. 그러나 생성한 임베딩을 트랜스포머 임베딩 하위층으로 보내지는 않을 것이다!

임베딩을 여섯 단계에 걸쳐 사이킷런(scikit-learn) 라이브러리에서 시작해 클러스터링 머신러닝 프로그램으로 전송한다.

- 1단계: OpenAI를 설치하고 임포트하고 API 키 입력하기
- 2단계: 데이터셋 불러오기
- 3단계: 열(column) 결합하기
- 4단계: GPT-3 임베딩 실행하기
- 5단계: 임베딩을 사용해 클러스터링(k-means) 하기
- 6단계: 클러스터링 결과 시각화하기(t-SNE)

위 여섯 단계를 요약하면 그림 16.10과 같다.



그림 16.10 임베딩을 클러스터링 알고리즘에 전달하는 여섯 단계

구글 코랩(Google Colab) 파일인 `Domain_Specific_GPT_3_Functionality.ipynb`을 열고 노트북의 `Embedding2ML with GPT-3 engine` 섹션을 보자.

이 섹션에서 설명한 각 단계는 노트북 셀과 일치한다. 프로세스의 각 단계를 살펴보자.

16.3.1.1 1단계: OpenAI 설치하고 임포트하기

하위 단계들을 시작해 보자.

- 셀을 실행한다.
- 런타임을 재시작한다.
- 셀을 재시작해서 런타임이 재시작 됐는지 확인한다.

```
try:
    import openai
except:
    !pip install openai
    import openai
```

API 키를 입력한다.

```
openai.api_key="[YOUR_KEY]"
```

이제 데이터셋을 불러오자.

16.3.1.2 2단계: 데이터셋 불러오기

이 책의 깃허브 저장소에서 `tracking.csv`를 확인할 수 있다. 셀을 실행하기 전에 이 파일을 불러오자. 이 파일은 SCM 데이터를 담고 있다.

```
import pandas as pd
df = pd.read_csv('tracking.csv', index_col=0)
```

데이터는 총 일곱 필드를 갖는다.

- Id
- Time
- Product
- User
- Score
- Summary
- Text

데이터 일부를 확인해 보자.

```
print(df)
```

	Time	Product	User	Score	Summary	Text	
Id							
1	01/01/2016 06:30	WH001	C001	4	on time	AGV1	
2	01/01/2016 06:30	WH001	C001	8	late	R1	NaN
3	01/01/2016 06:30	WH001	C001	2	early	R15	NaN
4	01/01/2016 06:30	WH001	C001	10	not delivered	R20	NaN
5	01/01/2016 06:30	WH001	C001	1	on time	R3	NaN
...
1049	01/01/2016 06:30	WH003	C002	9	on time	AGV5	NaN
1050	01/01/2016 06:30	WH003	C002	2	late	AGV10	NaN
1051	01/01/2016 06:30	WH003	C002	1	early	AGV5	NaN
1052	01/01/2016 06:30	WH003	C002	6	not delivered	AGV2	NaN
1053	01/01/2016 06:30	WH003	C002	3	on time	AGV2	NaN

[1053 rows x 7 columns]

열을 결합해서 원하는 클러스터를 만들 수 있다.

16.3.1.3 3단계: 열 결합하기

Product 열과 Summary 열을 결합해 제품과 제품의 배송 상태를 볼 수 있다. 이 예제는 실험적인 예제일 뿐이라는 걸 기억하자. 실제 프로젝트에서는 결합하려는 열을 신중하게 결정해야 할 것이다.

원한다면 다음 코드를 독자가 원하는 대로 수정해서 사용할 수도 있다.

```
df['combined'] = df.Summary.str.strip() + "-" + df.Product.str.strip()
print(df)
```

combined라는 새로운 열을 확인할 수 있다.

	Time	Product	User	...	Text		combined
Id				...			
1	01/01/2016 06:30	WH001	C001	...	AGV1		on time-WH001
2	01/01/2016 06:30	WH001	C001	...	R1 NaN		late-WH001
3	01/01/2016 06:30	WH001	C001	...	R15 NaN		early-WH001
4	01/01/2016 06:30	WH001	C001	...	R20 NaN		not delivered-WH001
5	01/01/2016 06:30	WH001	C001	...	R3 NaN		on time-WH001
...
1049	01/01/2016 06:30	WH003	C002	...	AGV5 NaN		on time-WH003
1050	01/01/2016 06:30	WH003	C002	...	AGV10 NaN		late-WH003
1051	01/01/2016 06:30	WH003	C002	...	AGV5 NaN		early-WH003
1052	01/01/2016 06:30	WH003	C002	...	AGV2 NaN		not delivered-WH003
1053	01/01/2016 06:30	WH003	C002	...	AGV2 NaN		on time-WH003

[1053 rows x 8 columns]

combined 열을 사용해 임베딩 모델을 실행해 보자.

16.3.1.4 4단계: GPT-3 임베딩 모델 실행하기

이제 davinci-similarity 모델을 실행해 combined 열의 12,288차원 임베딩을 얻을 수 있다.

```
import time
import datetime
# 시작 시간
start = time.time()
def get_embedding(text, engine="davinci-similarity"):
    text = text.replace("\n", " ")
    return openai.Engine(id=engine).embeddings(input = [text])['data'][0]
['embedding']

df['davinci_similarity'] = df.combined.apply(lambda x: get_embedding(x,
engine='davinci-similarity'))

# 종료 시간
end = time.time()
etime=end-start
conversion = datetimetimedelta(seconds=etime)
```

```
print(conversion)
print(df)
```

결과가 인상적이다. combined 열의 차원은 12,288이다.

```
0:04:44.188250
      Time      ... davinci_similarity
Id
1 01/01/2016 06:30 ... [-0.0047378824, 0.011997132, -0.017249448, -0...
2 01/01/2016 06:30 ... [-0.009643857, 0.0031537763, -0.012862709, -0...
3 01/01/2016 06:30 ... [-0.0077407444, 0.0035147679, -0.014401976, -0...
4 01/01/2016 06:30 ... [-0.007547746, 0.013380095, -0.018411927, -0...
5 01/01/2016 06:30 ... [-0.0047378824, 0.011997132, -0.017249448, -0...
...
1049 01/01/2016 06:30 ... [-0.0027823148, 0.013289047, -0.014368941, -0...
1050 01/01/2016 06:30 ... [-0.0071367626, 0.0046446105, -0.010336877, 0...
1051 01/01/2016 06:30 ... [-0.0050991694, 0.006131069, -0.0138306245, -0...
1052 01/01/2016 06:30 ... [-0.0066779135, 0.014575769, -0.017257102, -0...
1053 01/01/2016 06:30 ... [-0.0027823148, 0.013289047, -0.014368941, -0...
[1053 rows x 9 columns]
```

결과를 넘파이(numpy) 행렬로 바꾸어보자.

```
#행렬 만들기
import numpy as np
matrix = np.vstack(df.davinci_similarity.values)
matrix.shape
```

행렬의 모양(shape)은 1,053 × 12,288이다.

```
(1053, 12288)
```

여기서 얻은 행렬을 사이킷런(scikit-learn) 클러스터링 알고리즘에 입력해 보자.

16.3.1.5 5단계: 임베딩을 사용해 클러스터링(k-means)하기

4단계에서 얻은 행렬을 트랜스포머의 다음 하위층이 아닌 k-means 클러스터링 알고리즘에 입력해 보자. 먼저 사이킷런의 k-means를 임포트하자.

```
from sklearn.cluster import KMeans
```

12,288차원의 데이터셋을 사용해 고전적인 k-means 클러스터링 알고리즘을 실행하자.

```
n_clusters = 4
kmeans = KMeans(n_clusters = n_clusters,init='k-means++',random_state=42)
kmeans.fit(matrix)
labels = kmeans.labels_
df['Cluster'] = labels
df.groupby('Cluster').Score.mean().sort_values()
```

입력한 n_clusters=4에 따라 결과는 총 네 개의 클러스터를 갖는다.

```
Cluster
2 5.297794
0 5.323529
1 5.361345
3 5.741697
```

각 데이터의 라벨을 출력해 보자.

```
print(labels)
```

출력은 다음과 같다.

```
[2 3 0 ... 0 1 2]
```

이제 t-SNE를 사용해 클러스터링 결과를 시각화해 보자.

16.3.1.6 6단계: 클러스터링 결과 시각화하기(t-SNE)

t-SNE는 지역 유사성(local similarity)을 유지한다.⁴⁸

t-SNE를 시각화하기 위해 matplotlib 라이브러리를 사용한다.

```
from sklearn.manifold import TSNE
import matplotlib
import matplotlib.pyplot as plt
```

시각화하기 전에 t-SNE 알고리즘을 실행해야 한다.

```
#t-SNE
tsne = TSNE(n_components=2, perplexity=15, random_state=42, init='random',
            learning_rate=200)
vis_dims2 = tsne.fit_transform(matrix)
```

이제 matplotlib을 사용해 시각화할 수 있다.

```
x = [x for x,y in vis_dims2]
y = [y for x,y in vis_dims2]
for category, color in enumerate(['purple', 'green', 'red', 'blue']):
    xs = nparray(x)[df.Cluster==category]
    ys = nparray(y)[df.Cluster==category]
    plt.scatter(xs, ys, color=color, alpha=0.3)
    avg_x = xs.mean()
    avg_y = ys.mean()

    plt.scatter(avg_x, avg_y, marker='x', color=color, s=100)
plt.title("Clusters of embeddings-t-SNE")
```

48 역주. PCA를 적용하기 전 두 데이터의 거리는 PCA를 적용 후 줄어들게 된다. 이때, 적용 전 거리가 가장 큰 두 데이터 포인트 사이 거리가 최대한 줄어들지 않게 PCA는 동작한다.

각 데이터 포인트는 속한 클러스터에 따라 다른 색을 갖는 동그라미 모양으로 플롯에 표시된다. 그리고 각 클러스터의 중심(centroid)은 엑스 모양으로 표시된다.

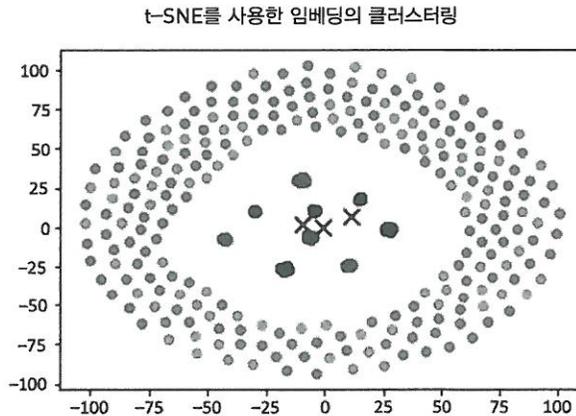


그림 16.11 t-SNE를 사용한 임베딩의 클러스터링 결과

지금까지 GPT-3 모델을 실행해 12,288차원의 임베딩을 얻었다. 그런 다음 클러스터링 알고리즘에 연결했다. 이와 같은 방식으로 트랜스포머를 머신러닝에 결합할 수 있는 방식은 무궁무진하다!

데이터 프레임을 자세히 보고 싶다면 노트북의 Peeking into the embeddings 섹션을 참고하자.

이제 지시 시리즈를 살펴보자.

16.3.2 지시 시리즈

개인 비서, 메타버스 속 아바타, 웹사이트 및 기타 여러 도메인에서 사용자가 도움을 요청할 때, 명확한 지시를 제공해야 하는 경우가 더 많아질 것이다. `Domain_Specific_GPT_3_Functionality.ipynb`의 `instruct series` 섹션을 살펴보자.

이번 섹션에서는 트랜스포머에 다음 프롬프트를 사용해 마이크로소프트 Edge에서 자녀 보호 기능을 설정하는 방법을 요청하자.

먼저 완성(completion) 셀을 실행하자.

```

import os
import openai
os.environ['OPENAI_API_KEY']=[YOUR_API_KEY]
print(os.getenv('OPENAI_API_KEY'))
openai.api_key = os.getenv("OPENAI_API_KEY")
response = openai.Completion.create(
    engine="davinci-instruct-beta",
    prompt="Explain how to set up parent control in Edge.WnWnWnACTIONS:",
    temperature=0,
    max_tokens=120,
    top_p=1,
    frequency_penalty=0,
    presence_penalty=0
)
r = (response["choices"])[0]
print(r["text"])

```

응답 결과 지시 목록은 다음과 같다.

1. Start Internet Explorer.(인터넷 익스플로러를 시작합니다.)
2. Click on the tools menu.(도구 메뉴를 클릭합니다.)
3. Click on the Internet options.(인터넷 옵션을 클릭합니다.)
4. Click on the advanced tab.(고급 탭을 클릭합니다.)
5. Click to clear or select the enable personalized favorite menu check box.
(클릭하여 개인화된 즐겨찾기 메뉴 사용 확인 상자를 선택합니다.)

요청할 수 있는 지시의 종류는 무궁무진하다. 창의력과 상상력을 발휘해서 더 많은 예시를 시도해 보자!

때로는 허용되지 않는 입력과 출력이 존재한다. 콘텐츠 필터를 구현해 해결해 보자.

16.3.3 콘텐츠 필터

편견과 허용되지 않는 언어 및 모든 형태의 비윤리적인 입력은 AI 애플리케이션에서 제외되어야 한다.

OpenAI에는 학습된 콘텐츠 필터 모델이 있다. 이번 섹션에서 예제를 실행해 보자. `Domain_Specific_GPT3_Functionality.ipynb`에서 `content filter` 섹션을 살펴보자.

그림 16.12에 보이는 것처럼 입력과 출력에 필터를 사용하는 것을 권장한다.

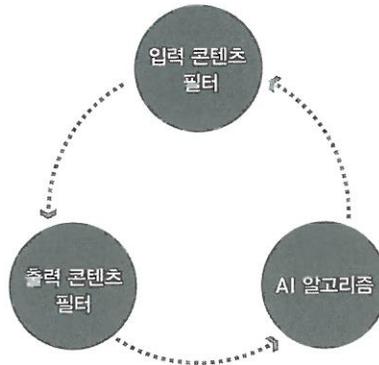


그림 16.12 콘텐츠 필터 구현하기

세 가지 과정을 거쳐 구현하기를 권장한다.

- 모든 입력 데이터에 콘텐츠 필터 적용하기
- 학습된 AI 알고리즘 실행하기
- 모든 출력 데이터에 콘텐츠 필터 적용하기

이번 섹션에서는 입력과 출력 모두 변수명 `content`를 사용한다.

혐오감을 일으키는 문장을 입력해 보자.

```
content = "Small and fat children should not play basketball at school."  
작고 뚱뚱한 아이들은 학교에서 농구를 하면 안 됩니다.
```

이러한 입력이 허용되어서는 안 된다! 학교는 NBA가 아니다. 농구는 누구나 어디에서나 할 수 있는 운동이다.

`content-filter-alpha` 셀을 실행해 콘텐츠 필터 결과를 살펴보자.

```

response = openai.Completion.create(
    engine="content-filter-alpha",
    prompt = "<|endoftext|"+content+"Wn--WnLabel.",
    temperature=0,
    max_tokens=1,
    top_p=1,
    frequency_penalty=0,
    presence_penalty=0,
    logprobs=10
)

```

API 결과 response는 파이썬 딕셔너리 객체이다. response에서 choice의 값을 통해 허용 가능 수준 결과를 얻을 수 있다.

```

r = (response["choices"])[0]
print("Content filter level:", r["text"])

```

콘텐츠 필터는 아래 세 가지 값 중 하나를 반환한다.

- 0-안전(safe)
- 1-민감한(sensitive)
- 2-안전하지 않은(unsafe)

이 경우, 결과는 2이다.

```
Content filter level: 2
```

콘텐츠 필터만으로는 부족할 수 있다. 입력/출력 콘텐츠를 제어하고 필터링하기 위해 규칙 기반, 사전 등 다른 알고리즘을 추가하는 것이 좋다.

지금까지 도메인별 모델을 살펴보았다. 이제 트랜스포머 기반 추천 시스템을 구축해 보자.

16.3.4 트랜스포머 기반 추천 시스템

트랜스포머 모델은 시퀀스를 학습한다. 매일 소셜 미디어와 클라우드 플랫폼에 게시되는 수십억 개의 메시지를 고려하면 언어 시퀀스를 학습하는 것은 좋은 시작일 수 있다. 소비자 행동, 이미지, 소리도 시퀀스로 표현할 수 있다.

이 섹션에서는 먼저 범용 시퀀스 그래프를 만든 다음 구글 코랩(Google Colab)에서 범용 트랜스포머 기반 추천 엔진을 구축한다. 그런 다음 메타휴먼(metahuman)으로 사용하는 방법을 살펴볼 것이다.

먼저 범용 시퀀스를 정의해 보자.

16.3.4.1 범용 시퀀스

많은 활동은 엔티티와 엔티티 간의 링크로 표현할 수 있다. 따라서 이러한 활동들을 순서대로 나열해 시퀀스를 구성할 수 있다. 예를 들어 유튜브의 동영상은 엔티티 A가 될 수 있고, A를 본 사람이 동영상 E로 이동한다면, 링크는 동영상 A에서 동영상 E로 이동하는 사람의 행동을 표현할 수 있다.

또 다른 예로, 발열 증상을 엔티티 F라 하고 의사가 내린 추론 결과를 엔티티 B라 하면, 엔티티 F와 B 사이 추론이 링크가 될 수 있다. 소비자가 아마존에서 제품 D를 구매하면 제안 제품 C 또는 다른 제품으로 연결되는 링크가 생성될 수 있다. 이러한 예시는 무궁무진하다!

이 섹션에서 다루는 엔티티는 다음과 같이 여섯 가지만 정의한다.

$$E = \{A, B, C, D, E, F\}$$

말을 할 때 문법 규칙을 따라야 한다.

예를 들면, A="I", E="eat" 그리고 D="candy"라 하자. I가 candy를 소비한 사실을 나타내는 적절한 시퀀스는 유일하다. "I eat candy"

누군가가 "eat candy I"라 말하면 약간 어색하게 들릴 것이다.

이 시퀀스에서 문법 규칙을 나타내는 링크는 다음과 같다.

$$A \rightarrow E \text{ (I eat)}$$

$$E \rightarrow D \text{ (eat candy)}$$

어떠한 도메인에서든 행동을 관찰하거나, 머신러닝을 사용해 데이터셋을 학습시키거나, 직접 전문가에게 의견을 들어 이러한 규칙을 추론할 수 있다.

이번 챕터에서는 몇 시간 동안 동영상을 시청하는 유튜브 사용자를 몇 달 동안 관찰했다고 가정한다. 사용자가 한 유형의 동영상에서 다른 유형의 동영상으로 체계적으로 이동하는 것을 발견했다. 예를 들어, 가수 B의 비디오에서 가수 D의 비디오로 이동하는 것을 발견했다. 사용자 P의 행동 규칙 X는 다음과 같다.⁴⁹

$$X(P) = \{AE, BD, BF, C, CD, DB, DC, DE, EA, ED, FB\}$$

이 엔티티와 규칙을 그래프로 나타낼 수 있다. 엔티티는 그래프의 정점(vertices)으로, 링크는 그래프의 변(edge)으로 표현할 수 있다. 예를 들어 X(P)를 그래프로 나타내면 다음 그래프를 얻을 수 있다.

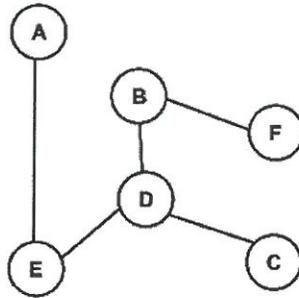


그림 16.13 유튜브 사용자의 행동 조합 그래프

사용자가 본 비디오는 좋아하는 가수들의 비디오라 가정하자. 그중 C는 사용자가 가장 좋아하는 가수라 하자. 사용자의 동영상 전환에 값 1을 부여하자. 그리고 가장 좋아하는 가수 C의 영상을 연달아 본 것에 대해서는 값 100을 부여하자.

이 사용자의 값이 적용된 경로(path) $V(R(P))$ 는 다음과 같다.

$$V(X(P)) = \{AE = 1, BD = 1, BF = 1, C = 100, CD = 1, DB = 1, DE = 1, EA = 1, ED = 1, FB = 1\}$$

추천 엔진의 목표는 가수 C의 영상으로 이끄는 시퀀스를 제안하거나 바로 C를 제안하는 것이다.

무향 그래프를 보상 행렬(reward matrix) R로 표현하면 다음과 같다.

49 역주. *C*는 CC를 가리킨다. 즉, 가수 C의 비디오를 연달아 본 경우를 나타낸다.

```

A,BC,DE,F
R = q1.matrix([[0,0,0,1,0], A
               [0,0,1,0,1], B
               [0,0,100,1,0,0], C
               [0,1,1,0,1,0], D
               [1,0,0,1,0,0], E
               [0,1,0,0,0,0]]) F

```

이 보상 행렬을 사용해 몇 달 동안 사용자 행동을 시뮬레이션해 보자.

16.3.4.2 MDP 기반 강화학습을 사용한 데이터셋 파이프라인 시뮬레이션

이번 섹션에서는 유튜브에서 노래 동영상을 시청하는 사용자 P의 행동 X를 시뮬레이션하고, 이를 X(P)로 정의한다. P의 행동에 값을 부여한 V(X(P))를 정의한다. 그런 다음 벨만 방정식으로 구현할 MDP(Markov Decision Process)에 사용한 보상 행렬 R의 값을 채워간다.

이 책의 깃허브 저장소에서 이 챗터 폴더에 있는 KantaiBERT_Recommender.ipynb를 열어보자. 이 노트북은 4장, RoBERTa 모델 처음부터 사전 학습하기에서 설명한 KantaiBERT.ipynb를 수정한 것이다.

4장에서 임마누엘 칸트의 작품 일부를 포함한 kant.txt를 사용해 트랜스포머를 학습했다. 이번 섹션에서는 강화학습(RL, Reinforcement Learning)으로 사람의 행동 시퀀스를 수천 개 생성한다. 강화학습은 이 책의 범위는 아니지만 이 섹션에서 일부 내용을 다룬다.

첫 번째 단계는 사람의 행동을 학습하고 시뮬레이션하기 위해 트랜스포머 모델을 학습하는 것이다.

16.3.4.3 MDP를 사용해 고객 행동 학습하기

4장에서 kant.txt를 불러와 DistilBERT 아키텍처를 사용한 RoBERTa 모델을 학습하며 장을 시작했다. kant.txt는 임마누엘 칸트 작품을 담고 있다. 이번 섹션에서는 이 챗터의 범용 시퀀스 섹션에서 정의한 보상 행렬 R을 사용해 시퀀스를 생성한다.

```
R = q1.matrix([[0,0,0,1,0],
               [0,0,0,1,0,1],
               [0,0,100,1,0,0],
               [0,1,1,0,1,0],
               [1,0,0,1,0,0],
               [0,1,0,0,0,0]])
```

프로그램의 첫 번째 셀은 다음과 같다.

Step 1A Training: Dataset Pipeline Simulation with RL using an MDP

(단계 1A 학습하기: MDP 기반 강화학습을 사용한 데이터셋 파이프라인 시뮬레이션)

이 셀은 벨만 방정식을 사용해 MDP를 구현한다.

```
# 벨만 MDP 기반 Q 함수
```

```
Q[current_state, action] = R[current_state, action] + gamma * MaxValue
```

이 방정식에서,

- R은 초기 보상 행렬이다.
- Q는 업데이트된 행렬로 R과 동일한 크기를 가지지만, 강화 학습을 통해 각 엔티티(정점)와 링크(변)의 상대적 값을 계산해 업데이트 된다.
- gamma는 학습 과정의 과적합(overfitting)을 방지하기 위해 0.8로 설정한 학습률이다.
- MaxValue는 current_state에서 action을 수행하여 얻은 next_state를 나타내는 행렬 Q의 행(row)에서 최댓값을 가리킨다. 즉, 다음 정점에서의 Q의 최댓값이다. 예를 들어 시청자 P가 가수 A를 시청하고 있을 때, 프로그램이 가수 E의 비디오를 다음 영상으로 제안할 수 있도록 E의 값을 증가시킬 수 있다.

이 프로그램은 시청자가 시청하기에 가장 적합한 동영상을 찾을 수 있도록 조금씩 최적의 값을 찾으려고 노력한다. 프로그램이 최적의 링크(변)를 학습하면 최적의 시청 시퀀스를 추천할 수 있다.

학습을 완료한 보상행렬을 사용해 서비스에 사용할 수 있다. 기존 엔티터를 표시하면 학습 결과를 명확하게 확인할 수 있다.

	A	B	C	D	E	F	
[[0.	0.	0.	0.	258.44	0.] A
[0.	0.	0.	321.8	0.	207.752]	B
[0.	0.	500.	321.8	0.	0.]	C
[0.	258.44	401.	0.	258.44	0.]	D
[207.752	0.	0.	321.8	0.	0.]		E
[0.	258.44	0.	0.	0.	0.]]	F

사용자 P의 행동 X의 초깃값 V 시퀀스는 다음과 같았다.

$$V(X(P)) = \{AE = 1, BD = 1, BF = 1, C = 100, CD = 1, DB = 1, DE = 1, EA = 1, ED = 1, FB = 1\}$$

학습 후 다음과 같이 변했다.

$$V(X(P)) = \{AE = 259.44, BD = 321.8, BF = 207.752, C = 500, DB = 258.44, DE = 258.44, EA = 207.752, ED = 321.8, FB = 258.44\}$$

이는 매우 큰 변화이다.

이제 P가 선호하는 가수의 동영상 시퀀스를 추천할 수 있게 됐다. P가 가수 E의 동영상을 조회한다 가정하자. 학습된 행렬의 E 라인(다섯 번째 row)의 가장 높은 값인 D=321.8의 동영상을 추천한다. 따라서 가수 D의 동영상이 사용자 P의 유튜브 피드에 표시될 것이다.

이 섹션의 목표는 여기서 더 나아간다. 이 섹션에서는 MDP를 사용해 유의미한 시퀀스를 생성해 트랜스포머가 학습에 사용할 데이터셋을 생성한다.

유튜브는 데이터셋을 생성하기 위해 시퀀스를 생성할 필요가 없다. 유튜브는 모든 사용자의 모든 행동을 저장한다. 그런 다음 구글의 강력한 추천 알고리즘이 최적의 동영상을 사용자의 동영상 피드에 추천한다.

다른 플랫폼은 9장, 데이터셋에 적합한 토큰라이저에서 구현한 코사인 유사도를 사용해 예측을 수행한다.

MDP는 유튜브, 아마존, 구글 검색 결과, 의사의 진단 과정, 공급망 및 모든 유형의 시퀀스를 학습할 수 있다. 트랜스포머는 시퀀스 학습, 예측의 수준을 높여주고 있다.

트랜스포머 모델에 사용할 행동 시퀀스를 생성하는 시뮬레이션을 구현해 보자.

16.3.4.4 MDP를 사용한 소비자 행동 시뮬레이션하기

첫 번째 셀에서 강화학습 부분을 학습하고 나면, 두 번째 셀의 Step 1B Applying: Dataset Pipeline Simulation with MDP(단계 1B 적용하기: MDP를 사용한 데이터셋 파이프라인 시뮬레이션)에서 유튜브 시청자의 행동을 시뮬레이션한다. 또한 유사한 시청자 프로필을 추가해 10,000개의 동영상 시청 시퀀스에 대한 시뮬레이션도 다룬다.

두 번째 셀은 KantaiBERT 트랜스포머 모델 학습에 사용할 kant.txt를 생성하며 시작한다.

```
""" 의사 결정 과정 시뮬레이션 """  
f = open("kant.txt", "w")
```

엔티티(정점)를 추가하자.

```
conceptcode=["A","B","C","D","E","F"]  
maxv=10000 # 시퀀스 수 10,000개로 설정
```

다음 함수는 임의의 시작 정점을 선택해 변수 origin에 할당한다.

```
origin=ql.random.randint(0,6)
```

이 프로그램은 학습한 행렬 Q를 사용해 origin에서 시작해 최고의 시퀀스⁵¹를 생성한다. 이 경우 다음과 같이 특정 사용자의 좋아하는 가수 시청 내역을 생성할 수 있다.

```
FBDC EDC EDC DC BDC AEDC AEDC BDC BDC AEDC BDC AEDC EDC BDC AEDC DC  
AEDC DC.../...
```

시퀀스 10,000개를 생성하고 나면, 트랜스포머 학습용 데이터셋 kant.txt가 완성된다.

프로그램의 나머지 셀들은 4장, RoBERTa 모델 처음부터 사전 학습하기에서 다룬 KantaiBERT.ipynb와 동일하다. 달라진 것은 앞서 만든 kant.txt를 사용한다는 점이다.

50 역주. 각 정점마다 Q 값이 가장 높은 정점만 선택한 시퀀스

이제 트랜스포머가 추천을 할 준비가 됐다.

16.3.4.5 추천 생성하기

4장에서 KantaiBERT.ipynb에서 다음과 같은 마스킹된 시퀀스를 다뤘다.

```
fill_mask("Human thinking involves human.") #인간에 대한 생각은 인간의 <mask>를 포함한다.
```

이 시퀀스는 임마누엘 칸트의 작품과 관련된 특정 시퀀스이다. 이 노트북에는 모든 도메인에서 사용할 수 있는 데이터셋이 있다.

노트북에서 다음을 입력해 보자.

```
fill_mask("BDC<mask>")
```

출력에 중복 결과가 있을 수 있다. 필터링 기능을 사용해 중복을 제거한 결과를 얻을 수 있다.

```
[{'score': 0.00036507684853859246,
  'sequence': 'BDC FBDC',
  'token': 265,
  'token_str': ' FBDC'},
 {'score': 0.00023987806343939155,
  'sequence': 'BDC DC',
  'token': 271,
  'token_str': ' DC'}]
```

결과 시퀀스는 일리가 있다. 한 시청자가 같은 동영상을 시청할 때도 있고 그렇지 않을 때도 있기 때문이다. 시청자의 행동은 무질서 할 수 있다. 머신러닝이 필요한 이유도 바로 여기에 있다. 이것이 바로 메타휴먼에 AI를 사용하는 방법이다.

메타휴먼 추천 엔진

일단 시퀀스가 생성되면, 사용자 인터페이스에 맞게 자연어로 다시 변환을 해줘야 한다. 이 섹션에서 메타휴먼은 많은 수의 피쳐(features)를 입력받아 다음 특징을 갖는 추천 엔진을 가리킨다.

- 많은 파라미터를 사용해 인간의 추론 능력을 증가
- 인간 보다 더 정확한 예측이 가능

이러한 실용적인 메타휴먼은 아직 디지털 휴먼이 아니지만 강력한 컴퓨팅 도구이다. 디지털 관점의 메타휴먼은 메타버스에서 인간과 AI 코파일럿 섹션에서 살펴볼 것이다.

예를 들어, BDC 시퀀스는 가수 B의 노래, 가수 D의 노래, 그리고 P가 가장 좋아하는 가수 C의 노래가 될 수 있다.

일단 시퀀스가 자연어로 변환되면, 여러 옵션이 가능하다.

- 시퀀스를 봇(bot)이나 디지털 휴먼에게 전송이 가능

NOTE

새로운 기술이 등장하면 한번 체험해 보자! 이 기술을 익혀 성장하게 될 것이다. 구글에서 검색해 다른 메타휴먼 플랫폼을 발견할 수 있다. 한계를 극복하는 방법을 익히고 새로운 기술을 사용하는 방법을 찾으며 최신 기술을 이어나갈 수 있다.

- API 요청 결과를 기다릴 때, 메타휴먼을 사용해 교육용 비디오를 제공할 수 있다.
- 메타휴먼을 인터페이스 내 음성 메시지로 삽입할 수 있다. 예를 들어 운전할 때, 구글 지도(Google Maps)를 사용하면 음성 메시지를 듣는다. 사람처럼 들려 착각하기도 하지만 기계이다.
- 아마존에 보이지 않게 내장될 수 있다. 사용자가 미세한 의사 결정(micro-decision)을 내릴 수 있도록 추천을 생성한다. 영업사원이 하는 것처럼 사용자에게 영향을 미친다. 보이지 않는 메타휴먼의 예이다.

이 섹션에서 범용 시퀀스는 MDP로 생성하고 RoBERTa 트랜스포머로 학습했다. 이는 트랜스포머가 모든 유형의 시퀀스에 적용될 수 있음을 보여준다.

트랜스포머가 컴퓨터 비전에 어떻게 적용되는지 살펴보자.

16.4 컴퓨터 비전

이 책은 컴퓨터 비전이 아닌 NLP에 관한 책이다. 하지만 이전 챕터에서는 다양한 분야에 적용할 수 있는 범용 시퀀스를 구현했다. 컴퓨터 비전도 그 중 하나이다.

2021년 도소비츠키(Dosovitskiy et al.)의 논문 제목이 모든 것을 말해준다. 「An image is worth 16x16 words: Transformers for Image Recognition at Scale.(이미지를 16x16 단어 시퀀스로 간주하고 트랜스포머에 이미지를 적용할 것임을 유추할 수 있다.)」 도소비츠키는 이미지를 시퀀스로 처리했다. 연구 결과는 그들의 생각이 옳았음을 입증했다.

구글은 코랩 노트북에서 비전 트랜스포머를 지원한다. 이 책의 깃허브 저장소 내 Chapter16 디렉터리에 있는 `Vision_Transformer_MLP_Mixer.ipynb`를 열어보자.

NOTE

이 노트북 파일을 실행하는 데 문제가 발생한다면 `Compact_Convolutional_Transformer.ipynb`를 대신 사용해 보자.

`Vision_Transformer_MLP_Mixer.ipynb`은 JAX 프레임워크를 사용한 트랜스포머 컴퓨터 비전 모델을 사용한다. JAX 프레임워크는 Autograd 라이브러리와 XLA 컴파일러를 동시에 지원한다.⁵¹ JAX는 네이티브 파이썬 함수와 넘파이 함수를 자동으로 구분할 수 있다. 컴파일 기술과 병렬처리로 파이썬과 넘파이 처리 속도를 높여준다.

노트북에 설명이 충분히 적혀있다. 노트북을 살펴보고 어떻게 동작하는지 확인해 보자. 명심할 점은 4차 산업 시대가 지나 5차 산업 시대가 도래하면 클라우드 AI 플랫폼에 데이터를 통합해 최상의 구현이 가능하다는 것이다. 로컬 개발은 줄어들고 기업들은 로컬 개발, 유지 관리 및 지원을 중단하고 클라우드 AI로 전환할 것이다.

51 역주. Autograd 라이브러리는 네이티브 파이썬과 넘파이를 함께 사용한 함수의 기울기(gradient)를 구할 때 유용하다. XLA 컴파일러는 소스 코드를 변환하지 않고 텐서플로우 모델을 가속화하는 선형대수학 도메인 전용 컴파일러이다. JAX는 고성능 머신러닝 연구에 많이 사용한다.

노트북 목차에는 이 책에서 여러 번 설명한 트랜스포머 처리 과정을 담고 있다. 이번에는 디지털 이미지 정보 시퀀스에 적용해 보았다.

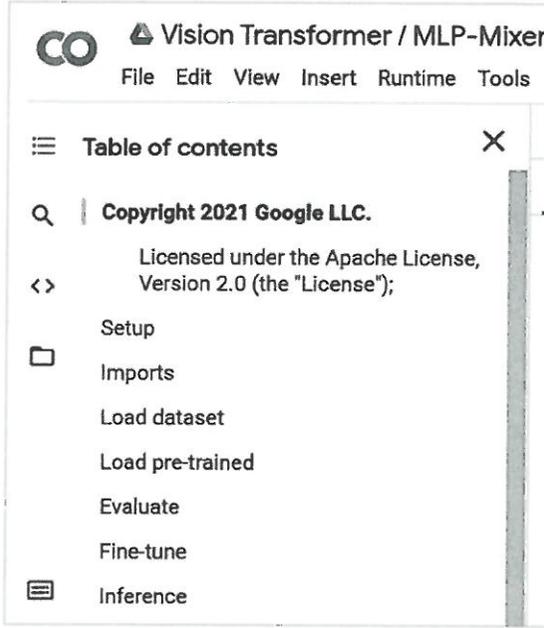


그림 16.14 비전 트랜스포머 노트북

노트북은 표준 딥러닝 방식을 따른다. 라벨이 있는 이미지를 보여준다.

```
# 라벨과 함께 이미지 출력하기
images, labels = batch['image'][0]:9, batch['label'][0]:9
titles = map(make_label_getter(dataset), labels.argmax(axis=1))
show_img_grid(images, titles)
```

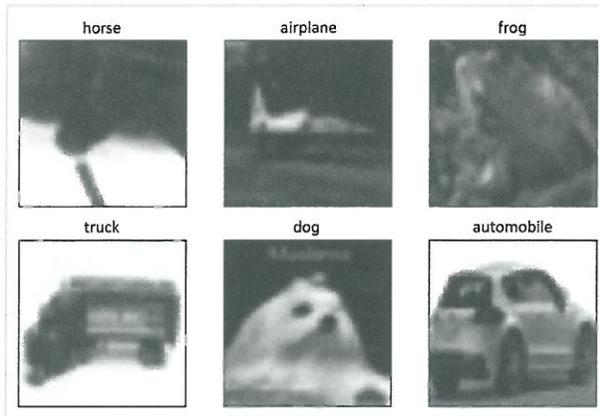


그림 16.15 라벨이 있는 이미지

NOTE

이 장에서 사용한 이미지들은 Alex Krizhevsky의 2009년 논문 「Learning Multiple Layers of Features from Tiny Images」(<https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf>)에서 가져왔다. 이미지들은 CIFAR-10 과 CIFAR-100 데이터셋에서 가져온 것이다. (torontoedu: <https://www.cs.toronto.edu/~kriz/cifar.html>)

노트북은 표준 트랜스포머 처리 과정을 거친 후 학습 이미지를 표시한다.

```
# 앞의 내용과 동일하지만, 학습 데이터를 표시
# 이미지가 어떻게 다르게 잘리고/확대되는지 주목한다.
# 오른쪽 에디터에서 input_pipeline.get_data() 코드를 확인해 이미지가 어떻게 전처리되는지 확인
한다.
batch = next(iter(ds_train.as_numpy_iterator()))
images, labels = batch['image'][0]:9, batch['label'][0]:9
titles = map(make_label_getter(dataset), labels.argmax(axis=1))
show_img_grid(images, titles)
```

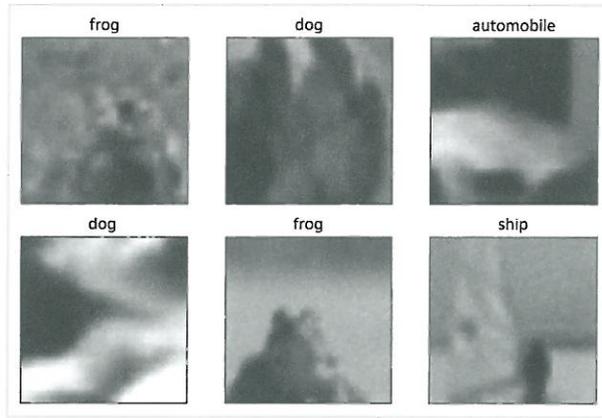


그림 16.16 학습 데이터

트랜스포머 프로그램은 임의의 사진을 분류할 수 있다. 원래 NLP용으로 설계된 트랜스포머 모델을 추천 엔진에 사용할 범용 시퀀스에 사용하고, 그런 다음 컴퓨터 비전에도 사용해 봤다. 마치 기적처럼 보인다. 하지만 시퀀스 학습의 일반화를 탐구한 것은 시작에 불과하다.

모델의 단순함은 놀랍다! 비전 트랜스포머는 트랜스포머의 아키텍처에 의존한다. 복잡한 컨볼루션 신경망을 사용하지 않는다. 그렇지만 비슷한 결과를 보여준다.

이제 로봇에 트랜스포머 모델을 탑재해 언어를 이해하고 이미지를 해석해 주변 세계를 이해하게 만들 수 있다.

메타휴먼과 메타버스에 비전 트랜스포머를 사용할 수 있다.

인간과 메타휴먼 AI가 메타버스로 합쳐지고 있다. 메타버스를 살펴보는 것은 이 책의 범위를 벗어난다. 이 책에서는 인간과 메타휴먼 AI로 채워진 메타버스로 가는 길을 안내할 도구를 제공한다.

아바타, 컴퓨터 비전, 그리고 비디오 게임은 다른 사람과의 커뮤니케이션을 몰입감있게 만들어 줄 것이다. 각자 스마트폰을 보는 것을 넘어 다른 사람들과 한 자리에 참여하는 듯한 몰입감을 얻을 수 있다.

16.5.1 보는 것을 넘어 참여하는 것

보는 것에서 참여하는 것으로의 진화는 자연스러운 과정이다. 우리는 컴퓨터를 발명하고 스크린을 추가한 다음 스마트폰을 발명했고, 이제는 화상 회의 앱을 사용한다.

이제 모든 유형의 회의와 활동을 위해 가상현실로 들어갈 수 있다. 예를 들어, 스마트폰에서 페이스북의 메타버스를 사용해 만나는 사람들(사적으로나 업무상으로)과 같은 장소에 있는 듯한 느낌을 받을 것이다. 이 느낌은 스마트폰 커뮤니케이션의 주요한 진화가 될 것이다.

어딘가에 있다고 느끼는 것은 스마트폰의 작은 스크린을 보는 것과는 완전히 다르다.

메타버스는 우주 여행, 거대한 파도 위에서 서핑, 숲 속 산책, 공룡 만나기 등 상상력이 닿는 어느 곳 이든 가능하게 해 줄 것이다.

물론 인간의 기술에는 한계, 위험, 위협 등이 존재한다. 하지만 콘텐츠 필터에서 보았듯이 AI를 사용해 AI를 제어할 수도 있다.

이 책에서 소개된 트랜스포머 도구는 새로운 메타버스 기술에 더해져 말 그대로 우리를 다른 세계로 데려다 줄 것이다.

이 책에서 익힌 지식과 기술을 잘 활용하자. 이것으로 메타버스 또는 물리적 세계에서 윤리적 미래를 만들어 보자.

16.6 정리하기

이 장에서는 인간의 의사 결정 수준 능력을 갖춘 AI 코파일럿의 부상에 대해 다루었다. 4차 산업 시대는 기계 간 상호 연결 시대의 문을 열었다. 기계 간 미세한 의사 결정으로 처리 속도가 빨라질 것이다. AI 코파일럿은 다양한 영역에서 생산성을 향상시킬 것이다.

코딩하는 동안 자연어 명령어로 소스 코드를 생성하기 위해 OpenAI의 지시 모델을 사용하는 방법을 살펴보았다.

MDP 프로그램으로 생성한 데이터셋을 사용해 RoBERTa 트랜스포머 모델을 학습하고, 이 모델을 사용해 트랜스포머 기반 추천 엔진을 구축했다. 범용 시퀀스 모델로 데이터셋을 얻었다. 따라서 메타휴먼은 범용 도메인 추천 기능을 가질 수 있었다.

그런 다음 비전 트랜스포머가 이미지를 시퀀스로 간주해 어떻게 분류할 수 있는지를 다루었다.

마지막으로, 메타휴먼이나 소셜 미디어에 내장돼 보이지 않는 형태로 사용자에게 추천을 제공할 수 있음을 살펴보았다.

트랜스포머는 엄청나게 복잡한 새로운 시대에 혁신적인 코파일럿 모델과 함께 등장했다. **17장, 초인간 트랜스포머를 사용한 OpenAI의 ChatGPT와 GPT-4**에서 도전적이고 흥미로운 여정을 이어가 보자.

16.7 문제

01. 코드를 자동으로 생성하는 AI 코파일럿은 존재하지 않는다. 참 / 거짓
02. AI 코파일럿은 결코 인간을 대체하지 못할 것이다. 참 / 거짓
03. GPT-3 엔진은 한 가지 작업만 수행할 수 있다. 참 / 거짓
04. 트랜스포머를 학습해 추천 엔진으로 사용할 수 있다. 참 / 거짓
05. 트랜스포머는 언어만 처리할 수 있다. 참 / 거짓
07. 트랜스포머에는 오직 단어시퀀스만 사용할 수 있다. 참 / 거짓
08. 비전 트랜스포머는 CNN과 필적할 수 없다. 참 / 거짓
09. 컴퓨터 비전을 갖춘 AI 로봇은 존재하지 않는다. 참 / 거짓
10. 파이썬 소스 코드를 자동으로 생성하는 것은 불가능하다. 참 / 거짓
11. 사람이 로봇의 코파일럿이 되는 날이 올 수 있다. 참 / 거짓

16.8 참고 문헌

- GPT-3를 사용하는 OpenAI 플랫폼: <https://openai.com>
- OpenAI의 모델과 엔진: <https://beta.openai.com/docs/engines>
- 비전 트랜스포머: Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby, 2020, An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale: <https://arxiv.org/abs/2010.11929>
- JAX와 비전 트랜스포머: <https://github.com/google/jax>
- OpenAI, 비주얼스튜디오 코파일럿: <https://copilot.github.com/>
- 페이스북 메타버스: <https://www.facebook.com/Meta/videos/577658430179350>

17장

초인간 트랜스포머를 사용한 OpenAI의 ChatGPT와 GPT-4

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

2022년 11월, OpenAI가 ChatGPT를 대중에 공개하며 지각변동을 일으켰다. 신문과 방송, 소셜 미디어는 앞다투어 ChatGPT를 소개하였다. ChatGPT가 할 수 있는 일들에 대한 루머는 산불처럼 퍼져나갔다. 2023년 3월, OpenAI는 Whisper와 GPT-4를 사용할 수 있는 API를 공개하며 또 한 번 미디어를 뜨겁게 달구었다. 또한 OpenAI가 코드 생성을 위해 Codex 대신 GPT-3.5-turbo와 GPT-4를 사용하기 시작하며 새로운 전환점을 맞이했다.

AI의 역사가 진행되고 있다!

하지만 AI 열풍과 거리가 멀었더라도 이 책을 읽고 있는 사람이라면 GPT-3.5 (ChatGPT)가 GPT-3를 논리적으로 진화시킨 버전임을 알 수 있다. 이 책을 열심히 읽었다면 프롬프트(prompt) 엔지니어링, 생성 완성, 멀티모달 비전 트랜스포머의 잠재력을 알고 있을 것이다.

이번 장에서는 끝없이 발전하는 트랜스포머의 강력함을 확인한다. 이전 장에서 습득한 이론과 실무 지식이 있다면 이 장에서 소개할 최첨단 모델을 살펴보기에 충분하다.

이번 장은 탐험하고 발견하는 경험을 제공한다. 지난 모든 장에서 얻은 깨달음과 기술들은 인공지능과 트랜스포머의 핵심으로 안내해 줄 것이다.

앞에서 다룬 익숙한 기능과 경험 대신 새로운 내용들에 초점을 맞출 것이다.

OpenAI가 GPT-3를 어떻게 GPT-3.5 legacy, GPT-3.5-turbo, GPT-4, DALL-E 등에 성공적으로 접목했는지 살펴볼 것이다.

그리고 나서, 파이썬 노트북에서 ChatGPT API를 확인하고, ChatGPT Plus로 코드를 생성하고, GPT-4를 사용하고, 고급 프롬프트를 설계하고, DALL-E 2 API로 이미지를 생성하고, 대화 AI에 오디오를 넣고, Whisper를 사용해 보는 등 최첨단 모델을 살펴볼 것이다.

이 장에서는 다음 주제들을 다룬다.

- ChatGPT API 시작하기
- ChatGPT와 ChatGPT Plus를 코파일럿(copilot)처럼 사용하기
- ChatGPT에게 분류 프로그램의 코드와 주석 작성 요청하기
- GPT-3.5 legacy 사용하기
- GPT-3.5-turbo 사용하기
- GPT-4 API 사용하기
- 고급 프롬프트 엔지니어링
- OpenAI의 모더레이션(moderation) 모델 실행하기
- ChatGPT와의 대화에 오디오 사용하기
- DALL-E 2 API를 사용하여 AI 아티스트 되기

먼저 가장 최신의 초인간 트랜스포머 모델을 살펴보자.

이번 섹션에서는 앞으로 살펴볼 내용에 관한 두 가지 측면에 대해 확인한다.

- 이번 장을 최대한 활용하는 방법
- 새로운 기회

이번 장을 최대한으로 활용하는 방법부터 알아보자.

17.1.1 이번 장을 최대한 활용하는 방법

이전 장들은 트랜스포머 모델의 주요 특징을 다루었다. ChatGPT와 GPT-4 같은 최신 OpenAI 모델들을 살펴보는 것은 기존 지식을 보완하는 것이 아니라 새롭게 더 나아가는 것이다.

17.1.1.1 기존의 지식 활용하기

책의 첫 장부터 지금까지에 걸쳐서 트랜스포머 모델의 주요 특징을 확인했다. 따라서 앞으로 살펴볼 노트북을 밑바닥부터 확인할 필요는 없다.

17.1.1.2 혁신에 집중하기

기억해야 할 내용을 요약하여 혁신에 집중할 수 있도록 구성했다. 책에서 얻은 전문 지식을 쌓는 과정을 즐기면 된다!

17.1.1.3 깃허브 저장소

이번 장에서 사용하는 모든 노트북 예제는 이 책에서 제공하는 깃허브 저장소의 Chapter17 경로에 있다.(<https://github.com/Denis2054/Transformers-for-NLP-2nd-Edition>)

각 섹션에서 다음과 같이 노트북 파일명을 언급한다면 Chapter17 경로에서 확인할 수 있다.(Open_Jump_Starting_ChatGPT_with_the_OpenAI_API.ipynb)

모든 노트북은 구글 코랩(Google Colab)의 무료 버전에서 개발되었지만 다른 환경에서도 실행 가능하다.(Kaggle, Gradient, Sagemaker 등)

일부 섹션은 깃허브 저장소의 Bonus 경로에 있는 노트북을 사용한다.

17.1.1.4 파괴적 혁신의 한계점 이해하기

최첨단 기술을 탐구한다는 것은 기술의 한계를 확인하는 것과 같다. OpenAI도 예외는 아니다. 한계를 인정하고 극복하기 위해 노력하면 새로운 지평을 발견할 수 있다.

우리가 마주할 한계점은 다음과 같다.

- ChatGPT에 접속할 때 OpenAI는 다음과 같이 경고한다. 때때로 시스템이 잘못된, 공격적인, 편향된 출력을 생성할 수 있다. (sometimes the system will generate erroneous, offensive, and biased outputs)
- 날짜로 인해 잘못된 정보를 생성할 수 있다. ChatGPT는 2022년 11월에 공개되었기 때문에 2021년 이전의 데이터만 사용한다.
- OpenAI의 모델은 빠르게 발전한다. 모델이 자주 바뀌고 중단된다.
- ChatGPT Plus를 사용하게 될 수 있다. ChatGPT Plus는 GPT-3.5 legacy (ChatGPT의 초기버전), GPT-3.5 default, GPT-4를 지원하지만 OpenAI의 스케일링 등의 이슈로 사용이 제한될 수 있다.
- 비용이 문제될 수 있다. 비용을 지불하기로 했다면 주의해야 한다! OpenAI에서 지출 비용에 따른 알람을 설정할 수 있다. 알람을 활성화하고 예산을 관리하는 것이 좋다.

아직 ChatGPT Plus와 API를 신청하지 못했어도 노트북을 읽어보며 ChatGPT를 살펴보자. 지난 챗터들의 내용을 알고있다면 이번 노트북을 충분히 이해할 수 있다.

AI의 콘텐츠 생성 잠재력은 무한하다. 하지만 더 나아가기 전에 저작권 이슈에 대해 짚고 넘어갈 필요가 있다.

17.1.2 AI 창작물의 소유권은 누구에게 있을까?

예술, 코드, 언어 등 AI가 생성한 콘텐츠의 소유권은 어떻게 될까? 이것은 수십억 달러 규모의 소송과 관련한 문제이다! 다음 세 글은 아직 정립되지 않은 AI 창작물의 저작권을 다룬다.

2022년 12월, Joe McKendrick가 포브스(Forbes)에 기고한 「Who Ultimately Owns Content Generated By ChatGPT And Other AI Platforms?」(<https://www.forbes.com/sites/joemckendrick/2022/12/21/who-ultimately-owns-content-generated-by-chatgpt-and-other-ai-platforms/>)

2023년 2월, Blake Brittain가 로이터(Reuters)에 기고한 「AI-created images lose U.S. copyrights in test for new technology」(<https://www.reuters.com/legal/ai-created-images-lose-us-copyrightstest-new-technology-2023-02-22/>)

2023년 4월 초, Tom Hals와 Blake Brittain이 로이터(Reuters)에 기고한 「Humans vs. machines: the fight to copyright AI art」(<https://www.reuters.com/default/humans-vs-machines-fight-copyright-ai-art-2023-04-01/>)

글을 읽고 직접 생각해보자. 결과는 좀 더 지켜보아야 한다.

명확한 결과가 나오기 전에, 생성형 AI를 사용할 때 다음을 주의하는 것이 좋다.

- 출판을 위한 콘텐츠는 직접 작성하자. AI는 검색 엔진이나 사전과 같은 도구로써 사용하며 작성 중인 내용을 검증하자.(필자는 이 장을 직접 작성했다.)
- 직장에서 ChatGPT를 사용했다면 사용했음을 표시하자. 예를들어, 이 책의 깃허브 저장소는 사용한 AI를 인용하고 있다. 또한 저장소의 모든 노트북을 MIT 라이선스로 오픈소스화 했다.
- AI로 콘텐츠를 생성하기 전에 법적인 내용을 확인하자. 콘텐츠에 접근하는 지역의 법과 규제를 미리 확인하는 것이 좋다.

이슈가 현재 진행형임을 기억하며, 이번에는 기회에 대해 생각해 보자.

17.1.2.1 새로운 기회

AI 프로젝트에 최신 OpenAI를 적용하며 얻을 수 있는 기회를 제약사항 때문에 놓쳐서는 안된다.

새롭게 얻을 수 있는 몇 가지 기회를 살펴보자.

- GPT-3.5 legacy, GPT-3.5 default, GPT-4는 AI의 새로운 단계를 보여준다. 다양한 NLP 문제에 이 트랜스포머 모델들을 적용할 수 있다.
- GPT-4로 생산성을 늘릴 수 있다면 경쟁에서 이길 수 있다.
- 2023년 3월 23일, OpenAI는 GPT-4와 같은 LLM(Large Language Models)의 영향력을 조사하고 다음과 같이 말했다.

“저희의 분석에 따르면 LLM을 사용하면 미국 내 모든 작업 중 약 15%를 훨씬 더 빠르게 동일한 품질로 완료할 수 있습니다.”

참조: <https://openai.com/research/gpts-are-gpts>

GPT(Generative Pre-trained Transformer) 모델은 범용기술(General-Purpose Technologies)이다. 트랜스포머는 NLP 뿐만 아니라 비전과 오디오에도 사용할 수 있다. 이전 장의 범용 시퀀스와 Embedding 2ML 섹션에서 보았듯이, NLP 이외의 분야도 GPT의 이점을 활용할 수 있다.

이제 이번 장에서 사용할 모델을 살펴보자.

17.1.2.2 이번 챕터에서 사용할 모델 목록

이번 장에서는 OpenAI가 <https://platform.openai.com/docs/models/>에서 소개한 모델들을 사용할 것이다.

- **GPT-4 limited beta:** GPT-3.5를 개선한 모델. 자연어와 코드를 이해하고 생성할 수 있다.
- **GPT-3.5:** GPT-3를 개선한 모델. 자연어와 코드를 이해하고 생성할 수 있다.
- **DALL-E beta:** 자연어 프롬프트에 따라 이미지를 생성하고 편집할 수 있는 모델
- **Whisper beta:** 오디오를 텍스트로 변환할 수 있는 모델(<https://openai.com/blog/introducing-chatgpt-and-whisper-apis>)
- **모더레이션:** 텍스트에 민감하거나 안전하지 않은 내용이 있는지 탐지하도록 미세 조정된(fine-tuned) 모델
- **GPT-3:** 자연어를 이해하고 생성할 수 있는 모델

모델의 종류는 다소 제한적이다. 하지만 각 모델에 다양한 버전이 있음을 고려하면 상당히 많은 엔진이 있음을 알 수 있다.

OpenAI의 주요 모델에는 다음 목록과 같이 다양한 하위 집합이 있다.

0 babbage	18 ada-code-search-code	36 text-search-curiequery-001
1 davinci	19 ada-similarity	37 text-search-babbagedoc-001
2 text-davinci-edit-001	20 text-davinci-003	38 gpt-3.5-turbo
3 babbage-code-search-code	21 code-search-ada-text-001	39 curie-search-document
4 text-similarity-babbage-001	22 text-search-adaquery-001	40 text-search-curie-doc-001
5 code-davinci-edit-001	23 davinci-search-document	41 babbage-search-query
6 text-davinci-001	24 ada-code-search-text	42 text-babbage-001
7 ada	25 text-search-ada-doc-001	43 gpt-4
8 curie-instruct-beta	26 davinci-instruct-beta	44 text-search-davincidoc-001
9 babbage-code-search-text	27 text-similarity-curie-001	45 gpt-4-0314
10 babbage-similarity	28 code-search-adaquery-001	46 text-search-babbagequery-001

11 whisper-1	29 ada-search-query	47 curie-similarity
12 code-search-babbagetext-001	30 text-search-davinciquery-001	48 curie
13 text-curie-001	31 curie-search-query	49 text-similarity-davinci-001
14 code-search-babbagecode-001	32 gpt-3.5-turbo-0301	50 text-davinci-002
15 text-ada-001	33 davinci-search-query	51 davinci-similarity
16 text-embedding-ada-002	34 babbage-searchdocument	
17 text-similarity-ada-001	35 ada-search-document	

표 17.1 OpenAI의 엔진 목록

OpenAI의 라이브러리는 탐색할 수 있는 기회와 방법이 가득하다! OpenAI는 전속력으로 발전하고 있으며 몇몇 모델은 최신 모델로 교체될 수 있음을 명심하자.

책이 쓰여진 시점에서는, 다음 모델들이 사라질 예정이다.

- 2023년 6월 1일, gpt-3.5-turbo-0301
- 2023년 6월 14일, gpt-4-0314
- 2023년 6월 14일, gpt-4-32k-0314

깃허브 저장소의 Bonus 경로에 있는 Exploring_GPT_4_API.ipynb 노트북에서 OpenAI 모델들을 확인할 수 있다. 이 노트북의 내용은 다음과 같다.

- 모델 목록을 확인한다
- 동일한 작업에 대해 GPT-4, GPT-3.5-turbo, GPT-3를 비교한다. 몇 가지 놀라운 점을 발견할 것이다.
- 대화, 수학 문제, 지시사항, 영화 이모티콘 표현, 일반 상식 질문, 안전하지 않은 콘텐츠, 감지되지 않은 부정적 트윗, 잘못된 번역, 사용자에 따른 검색 엔진 프롬프트 최적화와 같은 다양한 비교 작업을 수행한다.
- 비교 도구로 OpenAI 링크를 제공한다.

OpenAI가 수행하는 연구를 깊게 이해하게 될 것이다. 지금 실행해도 좋고, 다음 몇 섹션을 확인하고 나서 실행해도 좋다.

이제 최고 수준 트랜스포머의 세계로 들어갈 준비가 되었다.

먼저 ChatGPT의 API로 시작하자.

초기의 ChatGPT는 GPT-3를 발전시킨 GPT-3.5-turbo를 사용했다. 이전 내용에서 한 걸음만 나아갔지만, 정말 대단한 발걸음이다!

Jump_Starting_ChatGPT_with_the_OpenAI_API.ipynb를 열어보자.

OpenAI를 설치하고 불러온 후 API key를 입력한다.

7장, GPT-3 엔진을 사용한 초인간 트랜스포머 등장에서 수행한 과정과 동일하다. 필요하다면 7장을 다시 확인해 보자.

17.2.1 콘텐츠 생성

OpenAI는 혁신적인 요청 방식을 도입했다.

1. ChatCompletion이라는 포맷을 사용하여 요청한다.
2. ChatGPT로도 알려진 강력한 대화형 AI, gpt-3.5-turbo를 사용한다.
3. 각 메시지에는 역할(role)이 있다. 시스템(system), 어시스턴트(assistant), 사용자(user). 시스템은 전반에 설정하는 기본 메시지, 어시스턴트는 모델이 제공하는 사실적인 정보, 사용자는 핵심 요청 내용을 의미한다. 다음과 같이 프롬프트를 설계하고 전송할 수 있다.

```
response=openai.ChatCompletion.create(
    model="gpt-3.5-turbo",
    messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        #어시스턴트의 역할 설정
        #당신은 도움이 되는 어시스턴트입니다.
        {"role": "user", "content": "What web services do you offer?"},
        #사용자 혹은 개발자의 지시
        #어떤 웹 서비스를 제공하나요?
```

```

{"role": "assistant", "content": "We provide web designers, developers and
web templates."},
#어시스턴트의 이전 메시지
#웹 디자인, 개발, 웹 템플릿을 제공합니다.
{"role": "user", "content": "Do you have a starter package?"})
#사용자의 후속 질문 또는 개발자의 제안
#스타터 패키지가 있나요?

```

17.2.1.1 API 응답

API의 전체 응답(response)을 출력할 수 있다.

```

#응답 확인
response

```

응답 결과를 비롯한 다양한 정보를 확인할 수 있다.

```

<OpenAIObject chat.completion id=chatcmpl-6pO66dwZxO5lwtRNJhJaFeWwo1pAR
at
0x7fa9ee4cfb80> JSON: {
  "choices": [
    {
      "finish_reason": null,
      "index": 0,
      "message": {
        "content": "I'm sorry, but I'm just an AI language model and I
don't provide web services directly. If you're looking for a web design
and development package, there are several options available depending
on your needs, budget, and preferences. Many web design and development
companies offer starter packages that provide basic website features
and functionalities, such as a few pages, limited design options, and
basic SEO optimization. It's best to research different companies and
compare their packages and prices to choose the one that meets your
requirements.",
        "role": "assistant"
      }
    }
  ]
}

```

```

    }
  }
].
  "created": 1677705378,
  "id": "chatcml-6p066dwZx05lwtRNJhJaFeWwo1pAR",
  "model": "gpt-3.5-turbo-0301",
  "object": "chat.completion",
  "usage": {
    "completion_tokens": 104,
    "prompt_tokens": 52,
    "total_tokens": 156
  }
}
}

```

응답에 대한 자세한 설명은 <https://platform.openai.com/docs/guides/chat/introduction>에서 확인할 수 있다.

비용을 모니터링하기 위해 응답 내용에서 토큰 수를 확인하는 것이 중요하다! 토큰 수를 추출하여 어플리케이션 비용을 모니터링하는 기능을 추가하여 것이 좋다.

비용에 대한 자세한 내용은 <https://openai.com/pricing#language-models>에서 확인할 수 있다.

이제 응답에서 필요한 텍스트를 추출해 보자.

17.2.1.2 응답에서 텍스트 추출하기

다음 코드 한 줄로 json에서 결과를 가져올 수 있다.

```

#생성 AI 응답 추출하기
response["choices"][0]["message"]["content"]

```

적절히 생성된 텍스트가 출력된다.

I'm sorry, but I'm just an AI language model and I don't provide web services directly. If you're looking for a web design and development package, there are several options available depending on your needs, budget, and preferences. Many web design and development companies offer starter packages that provide basic website features and functionalities, such as a few pages, limited design options, and basic SEO optimization. It's best to research different companies and compare their packages and prices to choose the one that meets your requirements.

"죄송하지만 저는 AI 언어 모델일 뿐 웹 서비스를 직접 제공하지 않습니다. 웹 디자인 및 개발 패키지를 찾고 있다면 요구 사항, 예산, 선호도에 따라 여러 가지 옵션을 선택할 수 있습니다. 많은 웹 디자인 및 개발 회사에서 몇 개의 페이지, 제한된 디자인 옵션, 기본적인 SEO 최적화 등 기본적인 웹사이트 기능을 제공하는 스타터 패키지를 제공합니다. 여러 회사를 조사하고 패키지와 가격을 비교해 요구 사항에 맞는 회사를 선택하는 것이 가장 좋습니다."

OpenAI의 ChatGPT API를 사용할 수 있게 되었다! 이제 원하는 방향으로 기능을 변경하거나 확장할 수 있다.

TIP

이 책에서 살펴본 바와 같이, 좋은 응답을 얻기 위해선 프롬프트 엔지니어링이 중요하다. 계속해서 챗터를 살펴보고 ChatGPT의 예제들을 확인하며 지식을 쌓아나가자.

이번 섹션에서는 ChatGPT Plus로 코드와 주석 및 설명을 작성해 볼 것이다.

IBM SPSS 결정 트리(IBM SPSS Decision Tree)는 의사 결정 시스템을 위한 분류 및 결정 트리를 만드는 도구이다.(<https://www.ibm.com/products/spss-decision-trees>)

하지만 어떤 프로젝트에서는 복잡한 프로그램이 아니라 단순한 함수로도 충분하다.

깃허브 저장소의 Chapter17 경로에 있는 ChatGPT_Plus_writes_and_explains_classification.ipynb를 열어보자.

내용을 실행하기 전에 OpenAI를 설치하고 임포트한 후 API key를 입력하자.

17.3.1 프롬프트 설계

ChatGPT Plus의 권장 사항대로 사이킷런(scikit-learn)을 설치하고, ChatGPT Plus에 다음 두 가지 요청을 보냈다.

1. 사이킷런으로 Iris 데이터셋 분류기를 만들고 matplotlib 그래프로 결과를 표시한다. OpenAPI의 API는 사용하지 않는다.
2. 파이썬 초보자를 위해 자세한 설명을 추가한다.

이제 ChatGPT Plus가 설명과 코드를 작성한다!

ChatGPT Plus가 작성한 프로그램 소개글은 다음과 같다.

"This program is a simple machine learning project that demonstrates how to use Scikitlearn, a popular machine learning library in Python, to build a decision tree classifier for the Iris dataset. The Iris dataset is a famous dataset in the machine learning community and is often used as a benchmark for classification algorithms.

In this program, you will first load the Iris dataset and split it into training and testing sets. Then, you will create a decision tree classifier model using Scikit-learn's DecisionTreeClassifier class and train it on the training data. Next, you will evaluate the accuracy of the model on the testing set and compute the confusion matrix, which is a table that summarizes the performance of the model. Finally, you will plot the confusion matrix using Matplotlib..."

"이 프로그램은 파이썬에서 널리 사용되는 머신러닝 라이브러리인 사이킷런을 사용해, 홍채 데이터셋(Iris dataset)에 대한 결정 트리 분류기(decision tree classifier)를 구축하는 방법을 보여 주는 간단한 머신 러닝 프로젝트입니다. 홍채 데이터셋은 머신러닝 커뮤니티에서 유명한 데이터셋이며 분류 알고리즘의 벤치마크로 자주 사용됩니다.

이 프로그램에서는 먼저 홍채 데이터셋을 로드하고 훈련 데이터셋과 테스트 데이터셋으로 분할합니다. 그런 다음 사이킷런의 DecisionTreeClassifier 클래스를 사용하여 결정 트리 분류기 모델을 생성하고 학습 데이터에 대해 학습합니다. 다음으로 테스트 세트에서 모델의 정확도를 평가하고 모델의 성능을 요약한 표인 오차 행렬(confusion matrix)을 계산합니다. 마지막으로 Matplotlib를 사용하여 오차 행렬을 그립니다..."

ChatGPT Plus의 지시에 따라 사이킷런이 설치된 이후, ChatGPT가 프로그램을 작성한다.

17.3.2 ChatGPT Plus가 작성한 프로그램

이제 결과를 기다리기만 하면 된다.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
```

```

from sklearn.tree import plot_tree

# 홍채 데이터셋 불러오기
iris = load_iris()

# 학습 데이터셋과 테스트 데이터셋으로 분할하기
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.
target, test_size=0.2, random_state=42)

# 결정 트리 분류기 모델 학습하기
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)

# 테스트셋으로 모델 평가하기
score = model.score(X_test, y_test)
print(f"Accuracy: {score:.2f}")

# 오차 행렬 계산하기
y_pred = model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)

# 오차 행렬 그리기
fig, ax = plt.subplots()
im = ax.imshow(cm, interpolation="nearest", cmap=plt.cm.Blues)
ax.figure.colorbar(im, ax=ax)
ax.set(xticks=np.arange(cm.shape[1]),
       yticks=np.arange(cm.shape[0]),
       xticklabels=iris.target_names, yticklabels=iris.target_names,
       title="Confusion matrix",
       xlabel="Predicted label",
       ylabel="True label")
plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
         rotation_mode="anchor")
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        ax.text(j, i, format(cm[i, j], "d"), ha="center", va="center",
              color="white" if cm[i, j] > cm.max() / 2 else "black")
fig.tight_layout()

```

```
# 결정 트리 그리기
plt.figure(figsize=(20,10))
plot_tree(model, feature_names=iris.feature_names, class_names=iris,
          target_names, filled=True)
plt.show()
```

이제 ChatGPT가 생성한 코드를 복사하기만 하면 된다.

```
python
import pandas as pd
import numpy as np
```

그림 17.1 ChatGPT가 생성한 코드

노트북에 붙여 넣고 실행해 보자.

17.3.3 ChatGPT Plus가 그려낸 결과

ChatGPT는 오차 행렬(confusion matrix)을 만들었다.

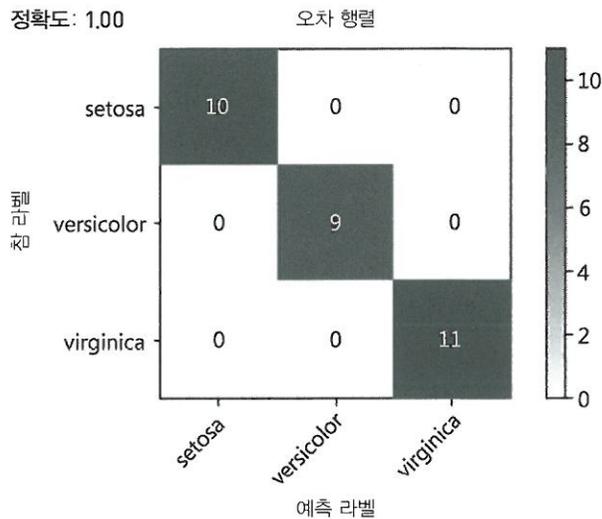


그림 17.2 ChatGPT가 만든 오차 행렬

그리고 결정 트리를 그렸다.

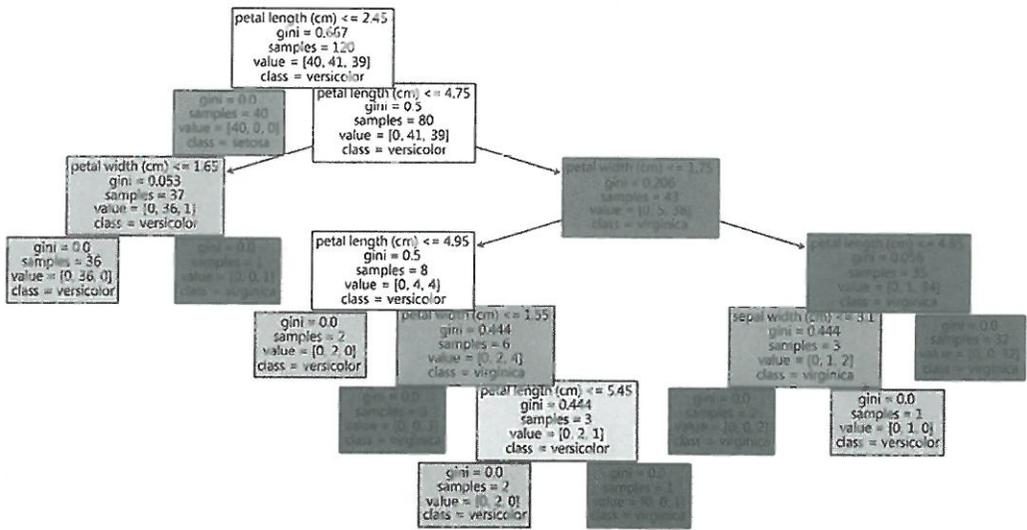


그림 17.3 ChatGPT가 만든 결정 트리

작업을 완료하기까지 10분도 걸리지 않았다. 아무리 빠르게 개발한다고 해도, 코파일럿으로 ChatGPT를 사용하는 것이 훨씬 빠르다!

이제 GPT-4를 살펴보자.

17.4 GPT-4 API 시작하기

이번 챕터에서는 GPT-4 API를 사용하는 ChatGPT Plus GPT-4를 코파일럿으로 사용해 보겠다. 출시 기간을 단축하기 위해 GPT-4를 코파일럿으로 사용하는 최첨단 방식을 경험해 보자.

챕터의 대부분은 필자가 직접 작성하였지만 GPT-4의 도움도 받았다. 윤리적 문제를 방지하기 위해 GPT-4가 작성한 내용은 “GPT-4:”로 표시했다.

Getting_Started_with_GPT_4.ipynb 파일을 열자.

이번 섹션은 노트북의 구조와 내용을 만드는 과정을 따라가 본다. 빠르고 편하게 작업하는 짜릿한 경험을 할 수 있다.

17.4.1 GPT-4에게 소스 코드 작성 도움 받기

GPT-4에 잘 만들어진 프롬프트를 전달하면 모델을 조종할 수 있다.

GPT-4는 소스 코드를 생성할 것이다. 이 노트북은 파이썬을 사용했지만 다른 언어로도 가능하다.

소스 코드의 오른쪽 위를 클릭하여 코드를 복사할 수 있다.

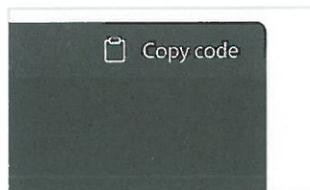


그림 17.4 코드 복사하기

노트북의 셀 안에 코드를 붙여 넣고 실행하자! 한계점을 명심하자. 디버깅이 필요할 수 있다. 각 섹션의 마지막에 테스트 결과를 첨부하였다.

17.4.2 GPT-4로 Greg Brockman이 2023년 3월 14일에 GPT-4를 소개한 유튜브 영상을 보여주는 기능 만들기

Denis Rothman: I would like to write a program in Python in Google Colab to display a YouTube video in a frame in Google Colab cell. The video is a presentation of GPT-4 by Greg Brockman. How I can write this code?

GPT-4: To display a YouTube video in a Google Colab notebook...

"

데니스 로스만: 구글 코랩에서 파이썬으로 프로그램을 작성해 구글 코랩 셀의 프레임에 유튜브 동영상을 표시하고 싶습니다. 이 동영상은 그렉 브록만(Greg Brockman)의 GPT-4 프레젠테이션입니다. 이 코드를 작성하려면 어떻게 해야 하나요?

GPT-4: 구글 코랩 노트북에 유튜브 동영상을 표시하려면...

"

GPT-4가 응답한 내용의 뒷부분을 노트북에서 확인하자. 원하는 기능을 GPT-4에게 설명하자, 수행해야 하는 내용을 알려주었다.

```
from IPython.display import YouTubeVideo

Greg_Brockman="hdhZwyf24mE"
# 아래 video_id를 원하는 유튜브 동영상 id로 바꾼다.

# 그렉 브록만의 GPT-4 프레젠테이션 id로 바꾸기
video_id = Greg_Brockman
YouTubeVideo(video_id)
```

17.4.3 GPT-4로 WikiArt의 이미지를 보여주는 프로그램 만들기

프롬프트를 입력하자, GPT-4가 지시사항을 전달했다.

```
Denis Rothman: I would like to write a Python program for Google Colab that
can display images from wikiart. How do I start?
GPT-4: To display images from Wikiart in a Google Colab notebook using
Python, you can follow these steps...
"
데니스 로스만: 위키아트(wikiart) 이미지를 그리는 구글 코랩용 파이썬 프로그램을 작성하고 싶습니다.
어떻게 시작해야 하나요?
GPT-4: 파이썬을 사용해 구글 코랩 노트북에 위키아트 이미지를 그리려면 다음 단계를 따르세요...
"
```

GPT-4의 지시사항을 확인하고 코드를 실행하자.

```
# 필요 라이브러리 임포트하기
import requests
from IPython.display import Image, display

# 위키아트 이미지를 그리는 함수 정의하기
def display_wikiart_image(url):
    response = requests.get(url)
    if response.status_code == 200:
        img = Image(data=response.content)
        display(img)
    else:
        print("Unable to fetch image")

# 아래 URL을 원하는 위키아트 이미지 URL로 바꾸기
wikiart_image_url = "https://uploads7.wikiart.org/images/salvador-dali/
the-persistence-of-memory-1931.jpg"
display_wikiart_image(wikiart_image_url)
```

각 기능을 수행하기까지 1분도 걸리지 않음을 명심하자.

17.4.4 GPT-4로 IMDb 리뷰를 확인하는 프로그램 만들기

모델을 조종하는 작업이 점점 익숙하고 편리해진다!

프롬프트를 입력하고 지시사항을 받았다.

```
Denis Rothman: Ok, I want to create another program on Google Colab in Python. This time I want to write a program that displays movie reviews from IMDb and displays them in the Google Colab notebook. How do I do this?
```

```
GPT-4: To display movie reviews from IMDb in a Google Colab notebook using Python, you can use the requests library to fetch the HTML ...
```

```
데니스 로스만: 좋아요. 구글 코랩에서 파이썬으로 다른 프로그램을 만들고 싶습니다. 이번에는 IMDb의 영화 리뷰를 보여주는 프로그램을 작성하고 싶습니다. 어떻게 해야 하나요?
```

```
GPT-4: 파이썬을 사용해 구글 코랩 노트북에 IMDb의 영화 리뷰를 보여주려면 requests 라이브러리를 사용해 HTML을 가져올 수 있습니다. ...
```

노트북에서 GPT-4의 지시사항을 확인하고 코드를 실행하자.

먼저 웹 페이지를 스크랩하기 위해 beautifulsoup4를 설치해야 한다.

```
pip install beautifulsoup4 lxml
```

이제 GPT-4가 작성한 코드를 실행할 수 있다.

```
import requests
from bs4 import BeautifulSoup
from IPython.display import display, Markdown

def display_imdb_reviews(movie_id, num_reviews=5):
    url = f"https://www.imdb.com/title/{movie_id}/reviews"
    response = requests.get(url)
    if response.status_code != 200:
        print("Unable to fetch IMDb reviews")
        return
    soup = BeautifulSoup(response.text, "lxml")
    reviews = soup.find_all("div", class_="imdb-user-review")
```

```

for idx, review in enumerate(reviews[:num_reviews]):
    title = review.find("a", class_="title").text.strip()
    author = review.find("span", class_="display-name-link").text.strip()
    date = review.find("span", class_="review-date").text.strip()
    content = review.find("div", class_="text").text.strip()
    display(Markdown(f"Review {idx + 1}: {title}"))
    display(Markdown(f"by {author} on {date}"))
    display(Markdown(f"{content}\n\n---"))

# 원하는 영화의 IMDB ID를 movie_id에 넣기
movie_id = "tt1375666" # 인셉션(2010)
display_imdb_reviews(movie_id)

```

요청한대로 리뷰 결과가 출력된다.

```

Review 1: A one-of-a-kind mind-blowing masterpiece!
by adrien_ngoc_1701 on 1 March 2019
My 3rd time watching this movie! Yet, it still stunned my mind...
"
Review 1: 세상에 단 하나뿐인 심금을 울리는 걸작!
by adrien_ngoc_1701 on 1 March 2019
이 영화를 세 번째 보는 겁니다! 하지만, 여전히 놀랍습니다. ...
"

```

17.4.5 GPT-4로 뉴스 피드를 보여주는 프로그램 만들기

인류는 걷는 것에서 시작해서, 말을 타고, 기차를 타고, 차와 비행기를 타며 발전했다. 곧 인류는 우주 여행을 것이다. 21세기에서, 뉴욕에서 LA까지 말을 타고 가고 싶어 하는 사람은 없다. 일부는 차와 기차를, 대부분은 비행기를 원할 것이다.

GPT-4도 동일하다. 한번 GPT-4의 속도를 경험하면 되돌아갈 수 없다!

이번에는 단순한 프롬프트 한 문장을 사용한다.

Denis Rothman: I want to write a program in Python in Google Colab that reads a news feed. How do I do that?

GPT-4: To write a Python program in Google Colab that reads a news feed, ...

"

데니스 로스만: 구글 코랩에서 뉴스 피드를 읽는 프로그램을 파이썬으로 작성하고 싶습니다. 어떻게 해야 하나요?

GPT-4: 구글 코랩에서 뉴스 피드를 읽는 파이썬 프로그램을 작성하려면...

"

노트북에서 GPT-4의 지시사항을 확인하고 코드를 실행하자.

```
!pip install feedparser

import feedparser

news_feed_url = "http://feeds.bbc.co.uk/news/rss.xml"

feed = feedparser.parse(news_feed_url)

for entry in feed.entries:
    print(entry.title)
    print(entry.link)
    print()
```

뉴스의 헤드라인과 참조할 웹 사이트를 확인할 수 있다.

```
NHS 5% pay offer may end bitter dispute in England https://www.bbc.co.uk/news/health-64977269?at\_medium=RSS&at\_campaign=KARANGA ./...
```

17.4.6 GPT-4로 K-means 클러스터링(kmc) 수행하기

이번 예제에서는 정확한 소스 코드를 얻기 위해 GPT-4와 대화를 주고받는다.

처음에 GPT-4는 정확한 답을 주지 못했다. 따라서 GPT-4에 에러 메시지를 전달하며 코드를 수정하도록 했다.

직접 만든 코드를 에러 내용과 함께 전달할 수도 있다.

여기서는 GPT-4가 직접 코드를 제공하고, 오류를 분석하고, 올바른 답을 다시 생성했다.

다음은 k-means 클러스터링을 요청하는 프롬프트이다.

```
Denis Rothman: Create a program in Google Colab in Python that can generate 1000 random numbers between 1 and 1000 and then run a k-means clustering algorithm on this data and finally display the result in matplotlib.
```

```
GPT-4: To create a program in Google Colab that generates 1000 random numbers between 1 and 1000, runs a k-means clustering algorithm on the data, and displays the result using matplotlib, follow these steps:
```

```
"
```

```
데니스 로스만: 구글 코랩에서 1에서 1000 사이의 난수 1000개를 생성하고 이 데이터에 대해 k-평균 군집화 알고리즘을 실행하고 마지막으로 matplotlib로 결과를 표시하는 파이썬 프로그램을 작성해줘.
```

```
GPT-4: 1에서 1000 사이의 난수 1000개를 생성하고 이 데이터에 대해 k-평균 군집화 알고리즘을 실행한 다음 matplotlib를 사용하여 결과를 표시하는 프로그램을 구글 코랩에서 만들려면 다음 단계를 따르세요:
```

```
"
```

노트북에서 GPT-4의 지시사항을 확인하고 코드를 실행하자

GPT-4가 이번에 생성한 코드는 동작하지 않을 것이다. 오류가 발생해도 당황하지 말자. GPT-4의 반응을 확인하고 새로운 코드를 실행해 보면 된다. 새로운 코드는 완벽하게 동작하였다.

GPT-4가 생성한 코드나 사용자가 직접 생성한 코드 모두 질문에 볼 수 있다. 21세기의 개발 생산성은 새로운 단계에 올라왔다!

17.4.7 GPT-4에게 GPT-4와 GPT의 아키텍처에 대해 질문하기

이 노트북은 OpenAI의 설립자인 Greg Brockman이 GPT-4를 발표한 지 24시간이 지난, 2023년 3월 16일에 작성되었다.

GPT-4와 흥미로운 인터뷰를 진행해 보았다.

인터뷰에서 주고받은 첫 대화는 다음과 같다.

Denis Rothman: * What is the difference between GPT-3.5 turbo and GPT-4?
GPT-4: As an AI language model, I am based on GPT-4 architecture, so I can provide you with a comparison between GPT-3.5 Turbo and GPT-4. GPT-3.5 Turbo is a version of the GPT-3 architecture. It was developed by OpenAI and is an improved, more efficient version of GPT-3.

./...

The improvements in GPT-4 include:

1. Higher capacity: GPT-4 has a larger number of parameters than GPT-3.5 Turbo, enabling it to store more knowledge and generate more sophisticated responses.
2. Better performance: GPT-4 can understand and generate text more effectively, resulting in improved performance across a range of tasks, including translation, summarization, question-answering, and more.
3. Enhanced fine-tuning capabilities: GPT-4 allows for better fine-tuning, making it more customizable for specific applications, industries, or use cases.
4. Newer knowledge base: Since GPT-4 was trained on more recent data, it has access to information and events that occurred after GPT-3.5 Turbo's training data was collected. However, it's important to note that GPT-4 may be more resource-intensive and potentially more expensive to use, depending on the specific application and API usage. In summary, GPT-4 is a more advanced and capable version of the GPT series compared to GPT-3.5 Turbo, with improvements in capacity, performance, fine-tuning, and knowledge base."

데니스 로스만: * GPT-3.5 turbo와 GPT-4의 차이점은 무엇인가요?

GPT-4: AI 언어 모델로서 저는 GPT-4 아키텍처를 기반으로 하고 있으므로 GPT-3.5 Turbo와 GPT-4를 비교하여 설명해드릴 수 있습니다. GPT-3.5 Turbo는 GPT-3 아키텍처의 버전입니다. OpenAI에서 개발했으며 GPT-3의 개선되고 더 효율적인 버전입니다.

./...

GPT-4의 개선 사항은 다음과 같습니다.

1. 더 큰 용량: GPT-4는 GPT-3.5 터보보다 더 많은 수의 매개 변수를 가지고 있어 더 많은 지식을 저장하고 더 정교한 응답을 생성할 수 있습니다.
2. 더 나은 성능: GPT-4는 텍스트를 더 효과적으로 이해하고 생성할 수 있어 번역, 요약, 질의응답 등 다양한 작업에서 성능이 향상됩니다.
3. 향상된 미세 조정 기능: GPT-4는 더 나은 미세 조정이 가능하여 특정 애플리케이션, 산업 또는 사용 사례에 맞게 더욱 맞춤화할 수 있습니다.

4. 최신 지식 기반: GPT-4는 최신 데이터로 학습되었기 때문에 GPT-3.5 Turbo의 학습 데이터가 수집된 이후에 발생한 정보 및 이벤트에 액세스할 수 있습니다. 하지만 특정 애플리케이션과 API 사용량에 따라 GPT4는 리소스를 더 많이 사용하고 잠재적으로 사용 비용이 더 많이 들 수 있다는 점에 유의해야 합니다. 요약하자면, GPT-4는 용량, 성능, 미세 조정 및 지식 기반이 개선되어 GPT3.5 Turbo에 비해 더욱 발전된 기능을 갖춘 GPT 시리즈 버전입니다.

GPT-4의 유연성과 코파일럿으로서 잠재력을 확인할 수 있다.

이제 OpenAI의 다른 주요 모델들을 살펴보자.

이번에는 이 책의 모든 장과 챕터에서 얻은 모든 전문 지식을 활용하여 진보된 프롬프트 엔지니어링을 해보려 한다. AI 기반 고급 검색 엔진의 철학을 바탕으로 도전적인 예제를 만들어두었다.

마이크로소프트의 코파일럿(Copilot)을 사용해 보자. 코파일럿은 요청한 내용에서 키워드를 파싱하고, 해당 키워드를 가진 웹페이지를 찾고, 대화형 AI의 형태로 결과를 보여준다. 검색 엔진의 과정을 간소화한 이번 노트북을 이해하는데 도움이 될 것이다.

우리는 지금까지 다음 중 하나의 방법으로 트랜스포머 모델을 사용했다.

- **모델을 그대로 실행하기**

지난 챕터들에서 번역, 감성 인식, 문장 완성 등 많은 예제를 살펴보았다. 하지만 이러한 트랜스포머 모델이 항상 우리의 요구사항에 맞는 것은 아니다.

- **모델을 미세 조정하기**

3장, BERT 모델 미세 조정하기와 7장, GPT-3 엔진을 사용한 초인간 트랜스포머 등장의 GPT-3 미세 조정하기 섹션을 확인하자.

LLM을 미세 조정하려면 신뢰할만한 데이터가 필요하다. 높은 품질의 데이터셋을 만들려면 많은 시간과 노력이 필요하다. 모델이 올바른 결과를 도출하도록 할 수 있는 학습 데이터를 만들어야 하지만 상당한 노력과 어려움이 따른다.

- **모델 사전 학습하기**

4장, RoBERTa 모델 처음부터 사전 학습하기에서 살펴본 것처럼 모델을 학습할 수도 있다.

하지만 OpenAI의 GPT-3, GPT-3.5-turbo, GPT-4와 같은 모델을 학습하기 충분한 리소스가 있는 곳은 많지 않다.

● 프롬프트 엔지니어링

프롬프트 엔지니어링도 책에서 몇 번 다루었다. 이전 섹션에서는 OpenAI의 ChatGPT, GPT-3.5-turbo, GPT-4를 사용하는 방법도 알아보았다. OpenAI의 완성(completion) 모델 덕분에 프롬프트로 다양한 정보를 제공하며 고급 프롬프트 엔지니어링을 할 수 있게 되었다. 시스템, 어시스턴트, 사용자 정보를 전달하며 GPT-3.5-turbo와 GPT-4를 조종할 수 있다.

이번 섹션의 프롬프트 엔지니어링은 몇 가지 상황에서 미세 조정의 대안이 될 수 있다.

- 프로젝트 초기에 데이터셋을 설계하는 가장 좋은 방법을 찾고 있을 때
- 기본 모델이 잘 작동하지 않고 미세 조정이나 재학습을 할 수 없을 때
- 프롬프트 엔지니어링으로 충분한 모든 상황

Prompt_Engineering_as_an_alternative_to_fine_tuning.ipynb를 열어보자.

이전 섹션에서는 OpenAI의 완성 및 생성형 AI 트랜스포머 모델을 알아보았다.

이번 노트북의 내용은 독립적이다. 다음 4가지 과정을 집중적으로 살펴본다.

1. ChatGPT를 위한 지식 베이스 구축하기
2. 키워드를 구축하고 사용자 요청 파싱하기
3. 요청한 내용이 담긴 의미 있는 프롬프트 구축하기
4. 검열, 품질관리

17.5.1 ChatGPT와 GPT-4를 위한 지식 베이스 구축하기

원하는 도구로 지식 베이스를 구축한다. 이후에 어시스턴트 항목으로 프롬프트에 전달할 것이다.

```
assert1={'role': 'assistant', 'content': 'Opening hours of Rothman Consulting
Monday through Friday 9am to 5pm, Services :expert systems, rule-based
systems, machine learning, deep learning, transformer models.}
assert2={'role': 'assistant', 'content': 'Services :expert systems, rule-based
systems, machine learning, deep learning, transformer models.}
assert3={'role': 'assistant', 'content': 'Services :Fine-tuning OpenAI GPT-3
models, designing datasets, designing knowledge bases.}
assertn={'role': 'assistant', 'content': 'Services:advanced prompt engineering
using a knowledge base and SEO keyword methods.}
```

```
#위 지식 베이스를 데이터셋으로 사용하기
kbt = []
kbt.append(assert1)
kbt.append(assert2)
kbt.append(assert3)
kbt.append(assertn)
```

17.5.2 키워드를 추가하고 사용자 요청 파싱하기

지식 베이스의 내용과 관련된 주요 키워드를 추가한다.

```
assertkw1="open"
assertkw2="expert"
assertkw3="services"
assertkwn="prompt"
```

그리고 나서, 사용자 요청을 파싱 하거나 요청과 지식 베이스 간의 유사도를 판단할 수 있는 함수를 설계한다. 노트북에 구현되어 있는 다음 함수는 사용자 요청을 파싱하는 한 가지 예시이다.

이것은 예시이다. 프로젝트에 맞게 커스터마이징할 수 있다.

```
def parse_user(uprompt,kbkw,kbt):
    i=0
    j=0
    for kw in kbkw:
        #print(i,kw)
        rq=str(uprompt)
        k=str(kw)
        fi=rq.find(k)
        if fi>-1:
            print(kw,rq,kbt[i])
            j=i
        i+=1
    return kbt[j]
```

17.5.3 ChatGPT를 조종하는 프롬프트 구축하기

지식 베이스를 구축하고 그와 관련된 키워드들을 등록하였다.

그리고 사용자 요청을 파싱 하여 키워드를 기반으로 가장 적합한 지식 베이스의 내용을 찾을 수 있도록 하였다.

이제 GPT-3.5-turbo 또는 GPT-4를 검색 엔진처럼 사용할 수 있다.

```
#convmodel="gpt-3.5-turbo"
convmodel="gpt-4"
def dialog(iprompt):
    response = openai.ChatCompletion.create(
        model=convmodel,
        messages=iprompt
    )
    return response
```

사용자 요청을 리스트로 받아, dialog 함수로 처리하도록 구성했다.

각 요청에 대한 응답은 판다스(pandas) 데이터프레임으로 변환하였다.

0	{'role': 'user', 'content': 'At what time does...	Rothman Consulting opens on Monday at 9am.	Total Tokens:83
1	{'role': 'user', 'content': 'At what time does...	Rothman Consulting is not open on Saturdays. O...	Total Tokens:97
2	{'role': 'user', 'content': 'Can you create an...	As an AI language model, I'm unable to create ...	Total Tokens:457
3	{'role': 'user', 'content': 'What services doe...	Rothman Consulting offers a wide range of serv...	Total Tokens:347

그림 17.5 사용자 요청 목록

비용 모니터링을 위해 총 토큰 수 또한 기록하였다. OpenAI는 토큰 1000개 단위의 요금을 청구한다. 평균적으로 사용 토큰 수의 75%가 단어 수이다. 즉, 사용 단어 수가 75개이면 평균적으로 사용 토큰 수는 100개이다. 비용 모니터링을 위한 함수를 구현하는 것이 좋다.

이제 고급 프롬프트 엔지니어링의 필수 요소인 검열 기능을 알아보자.

17.5.4 검열, 품질 관리

노트북의 4. Moderation, quality control(검열, 품질 관리) 셀에서 OpenAI의 모더레이션 모델을 사용하는 예제를 실행하자.

```
text = "I apologize for the confusion in my previous message. Rothman Consulting is open only from Monday through Friday from 9 AM to 5 PM. We are closed on weekends, including Saturdays. If you have any further queries, please let us know." ("제가 이전에 보낸 메시지가 혼동을 준 것에 대해 사과드립니다. Rothman Consulting은 월요일에서 금요일, 오전 9시부터 오후 5시까지만 문을 엽니다. 우리는 토요일을 포함해 주말에는 영업하지 않습니다. 추가 문의 사항이 있다면 연락해 주세요.")

response = openai.Moderation.create(input=text)
```

모델은 유용한 결과를 보여준다.

```
<OpenAIObject at 0x7ff867ff7b30> JSON: {
  "hate": false,
  "hate/threatening": false,
  "self-harm": false,
  "sexual": false,
  "sexual/minors": false,
  "violence": false,
  "violence/g
```

충분한 시간을 가지고 이번 노트북을 살펴보자. 그리고 고급 트랜스포머의 프롬프트에 정보를 제공하며 검색하는 새로운 기술을 이해해 보자.

결과를 설명할 수 있는 OpenAI의 최신 모델로 고급 프롬프트 엔지니어링을 구현하고 싶다면 XAI를 사용하는 것이 좋다. ChatGPT는 소스 코드를 설명할 수 있다. 또한 스스로가 만든 결과도 어느 정도 설명 가능하다.

NOTE

14장, 블랙박스 트랜스포머 모델 해석하기에서 설명 가능한 AI의 몇 가지 요소에 대하여 다루었다.

깃허브 저장소의 Bonus 경로에 있는 XAI_by_ChatGPT_for_ChatGPT.ipynb를 실행하면 ChatGPT에게 ChatGPT의 결과를 설명하도록 할 수 있다. ChatGPT XAI에게 ChatGPT의 결과를 SHAP(SHapley Additive exPlanations) 기법으로 분석하도록 할 것이다.

이 노트북은 독립적이며 숙련된 독자들이 XAI를 구축하는데 도움이 될 것이다.

먼저 ChatGPT와 오디오로 대화하는 방법을 알아보자.

17.6.1 Whisper와 음성인식

이번 섹션에서는 Whisper API와 gTTS(Google Text-to-Speech)를 사용하여 음성인식 모델을 구축할 것이다. 이번 노트북은 OpenAI 모델을 비롯한 소프트웨어에서의 발화 대화를 위한 기초 내용을 담고 있다.

Speaking_with_ChatGPT.ipynb를 열어보자.

OpenAI 설치, 임포트, API key 입력 등의 과정은 다른 노트북과 동일하다.

이제 텍스트 음성 변환을 시작해 보자.

17.6.1.1 gTTS를 사용한 텍스트 음성 변환

gTTS(Google Text-to-Speech)는 프로그램이 대화하도록 할 때 유용하다. gTTS를 설치하여 텍스트 음성 변환 기능을 추가할 것이다.

17.6.1.2 gTTS 설치 및 импорт하기

먼저 gTTS가 필요한 경우 설치하고 импорт한다.

#3.gTTS импорт하기

```
try:
    from gtts import gTTS
except:
    !pip install gTTS
    from gtts import gTTS
    from IPython.display import Audio
```

Windows의 음성인식 기능 등 원하는 모듈을 사용할 수 있다.

이번 노트북에서는 Windows를 사용한다. 자세한 내용은 Windows에서 **[Ctrl]+[H]**를 누르거나 마이크로소프트의 문서를 확인하자.(<https://support.microsoft.com/ko-kr/windows/음성-인식을-사용하여-텍스트-받아쓰기-854ef1de-7041-9482-d755-8fdf2126ef27>)

NOTE

주의: 이번 노트북에서는, 요청 내용을 완성했다면 엔터를 눌러야 한다.

어플리케이션에 입력 시간 제한(<https://pypi.org/project/pytimedinput/#:~:text=timedInput>) 등 다른 방식을 도입할수도 있다.

이 예제에서는 엔터 키를 기다리는 `prepare_message()` 함수를 사용한다.

#STT(Speech-To-Text), OS에 내장된 STT 앱 사용하기. 예를 들어 윈도우면 윈도우 키 + H를 눌러 사용한다.

```
def prepare_message():
    #마이크를 사용하여 요청을 입력하거나 직접 타이핑한다.
    # 예: "Where is Tahiti located?"
    print("Enter a request and press ENTER:")
    uinput = input("")
```

“Where is Tahiti located?”를 입력하였다.

이제 API에 전달할 프롬프트를 준비한다.

```
#OpenAI에 사용할 프롬프트 만들기
role="user"
#prompt="Where is Tahiti located?" #마이크를 사용하지 않을때 사용
line = {"role": role, "content": uinput}
line = {"role": role, "content": prompt}
```

OpenAI의 메시지 객체를 만들기 위한 변수를 생성했다.

NOTE

메시지 구현에 대한 더 많은 정보를 위해 고급 프롬프트 엔지니어링 섹션에서 설명한 `Prompt_Engineering_as_an_alternative_to_fine_tuning.ipynb`를 실행해 보자.

이제 전체 메시지를 만들고 프롬프트를 반환한다.

```
#메시지 만들기
assert1={"role": "system", "content": "You are a helpful assistant."}
assert2={"role": "assistant", "content": "Geography is an important topic
if you are going on a once in a lifetime trip."}
assert3=line
iprompt = []
iprompt.append(assert1)
iprompt.append(assert2)
iprompt.append(assert3)
return iprompt
```

17.6.1.3 GPT-3.5-turbo에 요청하기

프롬프트의 메시지 객체를 완성했다. 이제 대화형 모델을 골라 요청을 보내면 된다. 여기서는 GPT-3.5-turbo를 사용할 것이다.

```
response=openai.ChatCompletion.create(
    model="gpt-3.5-turbo",
    messages=iprompt)
```

이제 response에서 내용을 추출한다.

```
#생성 AI 응답 추출하기
text=response["choices"][0]["message"]["content"]
text
```

노트북 예시의 결과는 다음과 같다.(실행에 따라 달라질 수 있음)

```
Tahiti is located in the South Pacific Ocean, specifically in the archipelago
of Society Islands and is part of French Polynesia. It is approximately 4,000
miles (6,400 kilometers) south of Hawaii and 5,700 miles (9,200 kilometers) west
of Santiago, Chile.
```

```
"타히티는 남태평양, 특히 소사이어티 제도 군도에 위치하고 있으며 프랑스령 폴리네시아의 일부입니다. 하
와이에서 남쪽으로 약 4,000마일(6,400킬로미터), 칠레 산티아고에서 서쪽으로 5,700마일(9,200킬로미터)
떨어져 있습니다."
```

17.6.2 IPython과 gTTS를 사용한 텍스트 음성 변환

텍스트 출력 결과를 확인했으니 이제 텍스트를 음성 변환 할 수 있다. gTTS를 사용하여 결과를 WAV 파일로 저장하고 IPython 오디오로 재생한다.

```
from gtts import gTTS
from IPython.display import Audio
tts = gTTS(text)
tts.save('1.wav')
sound_file = '1.wav'
Audio(sound_file, autoplay=True)
```

파일은 자동으로 재생되고 인터페이스로 조종할 수 있다.

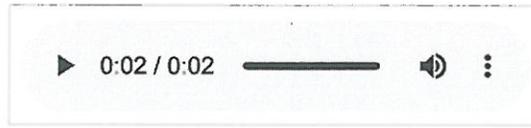


그림 17.6 소리 파일 재생하기

17.6.3 Whisper로 오디오 파일 받아쓰기

Whisper를 사용하여 회의, 컨퍼런스, 인터뷰 등의 오디오 파일을 다양한 언어와 용도의 텍스트로 변환할 수 있다.

먼저 오디오 처리 모듈 `ffmpeg`을 설치한다.

```
!pip install ffmpeg
```

이제 오디오 파일을 변환한다. Whisper가 자동으로 언어를 감지할 것이다.

```
!whisper l.wav
```

감지한 언어와 자막 그리고 타임라인이 출력된다.

```
Detected language: English
[00:00.000 → 00:04.440] Tahiti is located in the South Pacific Ocean.
[00:04.440 → 00:06.320] Specifically in French Polynesia. [00:06.320 → 00:10.720] It
is the largest island in the Windward Group of Islands in French Polynesi
```

출력 결과는 다양한 파일 포맷으로도 저장된다.(json, srt, tsv, txt, and vtt)

json에서 필요한 텍스트를 추출할 수 있다.

```
import json

with open('1.json') as f:
    data = json.load(f)

text = data['text']
print(text)
```

노트북 예시의 결과는 다음과 같다.

```
Tahiti is located in the South Pacific Ocean, specifically in the archipelago
of society islands, and is part of French Polynesia. It is approximately 4,000
miles, 6,400 km, south of Hawaii and 5,700 miles, 9,200 km, west of Santiago, Chile.
```

이 노트북의 내용을 다양한 프로젝트에 도입해 보자.

이제 DALL-E API로 이미지를 생성해 보자.

DALL-E의 텍스트 이미지 변환은 짧은 시간 동안 크게 발전했다. 트랜스포머는 작업의 종류에 제약이 없어 다양한 곳에 적용될 수 있다. 트랜스포머는 이제 오디오, 이미지, 등 다양한 신호를 융합하여 멀티모달(multimodal)로 사용된다.

지난 15장, NLP부터 범용 트랜스포머 모델까지의 DALL-E 섹션에서 DALL-E의 아키텍처를 알아보았다.

<https://openai.com/index/dall-e-2>에서 DALL-E 2를 사용해 볼 수 있다.

하지만 이번 섹션에서는 DALL-E 2 API의 2세대 API를 사용하여 프로그램을 직접 작성한다. API를 사용하면 다양한 이미지를 생성, 수정, 변형할 수 있다.

Getting_Started_with_the_DALL_E_2_API.ipynb을 열어보자. 적합한 프로젝트에 DALL-E 2 API를 도입할 수 있을 것이다. 이 노트북은 DALL-E 2 API를 사용하는 방법을 소개하고 있다.

셀 단위로 실행하며 각 기능을 이해하거나 노트북의 시나리오 전체를 한 번에 실행해 보자.

이번 섹션은 다음 두 부분으로 나누어진다.

- 새로운 이미지 생성하기
- 이미지 변형하기

17.7.1 새로운 이미지 생성하기

첫 번째 셀에서 목표를 설정한다. 이어지는 모든 시나리오에 사용할 것이다.

이번 예제에서는, 달이 보이는 식당에서 챗봇과 대화하는 사람의 이미지를 만들고 파일로 저장할 것이다.

```
#프롬프트
```

```
sequence="Creating an image of a person using a chatbot in a restaurant on a  
spaceship near the moon."#달 근처 우주선의 식당에서 챗봇을 사용하는 사람의 이미지를 만들기
```

sequence는 DALL-E 2에게 지시할 텍스트 프롬프트이다.

OpenAI를 설치, 임포트하고 API key를 입력하는 셀을 실행하자.

이제 Generation에서 DALL-E API를 실행한다.

```
#이미지 생성하기
```

```
response = openai.Image.create(  
    prompt=sequence,  
    n=2, #생성할 이미지 개수  
    size="1024x1024")  
image_url = response['data'][0]['url']
```

openai.Image.create로 요청한다. 이 노트북의 첫 번째 셀에서 설정한 시퀀스를 프롬프트로 사용한다.

이미지가 생성되었다. 다음 셀은 PIL의 함수를 사용하여 이미지를 표시한다.

```
# 이미지 출력하기
```

```
url = image_url  
image = Image.open(requests.get(url, stream=True).raw)  
image.save("c_image.png", "PNG")  
c_image = Image.open(requests.get(url, stream=True).raw)  
c_image
```

생성한 이미지가 표시된다.

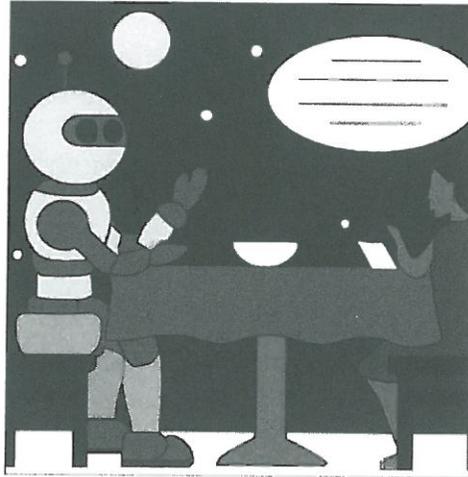


그림 17.7 DALL-E 2 API로 생성한 이미지

같은 시퀀스를 사용해도 API를 실행할 때마다 다른 버전의 이미지가 생성될 것이다.

이미지를 다음과 같이 저장한다.

```
image.save("c_image.png", "PNG")
```

이제 이미지를 변형해보자.

17.7.2 이미지 변형하기

DALL-E 2의 창의성을 확인하고, `c_image.png`로 저장했던 이미지를 변형하도록 요청해 보자.

다음 내용은 Variation 셀의 예제이다. 이미지를 선택하고 DALL-E에게 비슷하지만 더 창의적으로 이미지를 변형하도록 요청한다.

```
#이미지 변형 만들기
response = openai.Image.create_variation(
    image=open("c_image.png", "rb"),
    n=1,
    size="1024x1024")
image_url = response['data'][0]['url']
```

이미지를 확인한다.

```
image_url = response['data'][0]['url']
image = Image.open(requests.get(url, stream=True).raw)
v_image = Image.open(requests.get(url, stream=True).raw)
v_image
```

이미지가 좋게 변형되었을 수도, 거의 변형되지 않았을 수도 있다. DALL-E 2에 전달한 프롬프트와 이미지에 따라 달라진다. OpenAI의 고도화에 따라서도 달라질 것이다.

지금까지 DALL-E 2 API로 이미지를 생성하고 변형하는 방법을 알아보았다.

이제 당신은 AI 이미지 아티스트가 되는 길을 걷고 있다!

이제 가득한 셸렘 속에서, 모든 내용을 종합해 보자.

정리하기 챗터로 넘어가기 전에, 살펴본 모든 모델을 하나의 노트북으로 정리해 보자. 이번 장은 은하계의 탄생처럼 끊임없이 진화하고 역동적인 인공지능의 최첨단 영역을 소개하고 있다.

ChatGPT를 비롯한 OpenAI의 다양한 신규 모델로 혁신을 확인했다. GPT3.5-turbo, GPT-4, 모더레이션, Whisper, DALL-E, gTTS 또한 사용하였다.

이러한 모델들로 프로그램을 구축하는 방법을 확인했지만 내용이 너무 많았다.

장의 모든 노트북을 요약하고 더 나아가기 원한다면, ALL-in-ONE.ipynb를 실행해 보자. 흥미로운 시나리오로 모든 모델을 리뷰할 수 있을 것이다.

1. 노트북에서 사용할 모듈과 OpenAI 설치
2. 요청 입력
3. 모더레이션 모델로 콘텐츠 안전성 검사
4. ChatGPT가 이야기를 생성하도록 프롬프트 준비
5. 스토리에 기반한 시를 작성하는 GPT-4
6. 시에 대한 삽화를 생성하는 DALL-E 2
7. gTTS로 시 낭송하기
8. Whisper로 오디오 받아쓰기

노트북을 실행하면 몇 분 안에 OpenAI의 최신 모델들을 리뷰 할 수 있다. 항상 혁신의 선두에 서게 될 것이다.

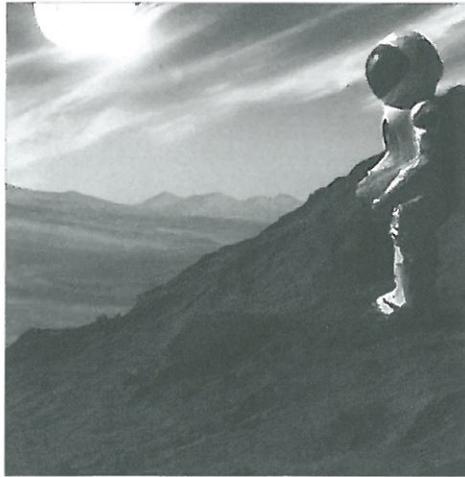


그림 17.8 DALL-E 2가 생성한 삽화

구글 코랩의 런타임(Runtime) 메뉴에서 모두 실행(Run all)을 클릭하고 마법을 즐기면 된다. 이제 이번 장에서 살펴본 내용을 정리해 보자.

이번 장에서는 이전 장들에서 얻은 전문 지식을 활용하여 OpenAI의 최첨단 모델을 살펴보았다.

7장, GPT-3 엔진을 사용한 초인간 트랜스포머 등장에서 살펴본 GPT-3 모델에서 한 단계 더 발전한 지식을 얻을 수 있었다. 트랜스포머를 이제 처음 접한 사람에게는 꽤 먼 여정이 될 것이다.

먼저 7장에서처럼 ChatGPT를 사용해 보았다. 그리고 나서 대화형 AI 프롬프트를 만드는 새로운 방식을 배웠다. ChatGPT Plus로 k-means 클러스터링 수행하고 그래프를 그리고 설명을 요청하는 방법을 확인했다.

GPT-4를 사용하는 것은 강력한 범용 트랜스포머 모델을 도입하는 것을 의미한다. davinci, GPT3.5-turbo, GPT-4를 비롯한 50개 이상의 트랜스포머 모델 목록을 확인했다. OpenAI의 ChatGPT Plus, GPT-3.5, GPT-4를 설명 가능한 AI로 사용하여 트랜스포머의 결과를 설명할 수 있었다.

지식 베이스를 구축하고 요청 프롬프트에 데이터를 추가하여 고급 프롬프트 엔지니어링을 수행하는 방법을 살펴보았다. 기업의 지식 베이스를 활용할 수도 있다.

그리고 나서, OpenAI의 트랜스포머에 음성인식을 추가해 보았다. DALL-E의 텍스트 이미지 변환 기능으로 이미지를 생성, 수정, 변환하는 방법을 확인했다. 마지막으로, 모든 모델을 하나의 노트북으로 모아 한 번에 리뷰할 수 있도록 했다.

이제 발전하는 AI와 LLM의 최신 정보를 파악하는 것은 여러분의 몫이다. Bonus 디렉터리에서 최신 AI에 대한 노트북을 더 확인할 수 있다. 이제 여러분은 트랜스포머 프로그램을 만들 준비가 되었다.

멋진 프로젝트를 완성하기 위해 전문지식을 활용하자!

17.10 문제

01. GPT-4는 자의식이 있다. 참 / 거짓
02. ChatGPT는 전문가를 대체할 수 있다. 참 / 거짓
03. GPT-4는 모든 문제에 대한 소스 코드를 만들 수 있다. 참 / 거짓
04. 고급 프롬프트 엔지니어링은 사용하기 매우 쉽다. 참 / 거짓
05. GPT-4는 가장 진보된 트랜스포머이며 항상 최고의 선택이다. 참 / 거짓
06. GPT-4를 코파일럿으로 사용하면 별다른 학습 없이도 애플리케이션을 만들 수 있다. 참 / 거짓
07. AI는 한계에 도달했으므로 GPT-4는 OpenAI의 마지막 트랜스포머 모델이 될 것이다. 참 / 거짓

17.11 참고 문헌

- OpenAI의 ChatGPT 블로그: <https://openai.com/blog/chatgpt>
- ChatGPT와 Whisper API를 소개하는 OpenAI 블로그: <https://openai.com/blog/introducing-chatgpt-and-whisper-apis>
- ChatGPT Plus를 소개하는 OpenAI 블로그: <https://openai.com/blog/chatgpt-plus>
- OpenAI 모델: <https://platform.openai.com/docs/models/>
- OpenAI GPT-4: <https://openai.com/product/gpt-4>
- OpenAI의 코덱스 지원 중단: <https://openai.com/blog/openai-codex>
- 코덱스를 사용하는 마이크로소프트 깃허브 코파일럿 (2023년 3월): <https://github.com/features/copilot>
- 구글의 텍스트 음성 변환: <https://pypi.org/project/gTTS/>

부록 I

트랜스포머 용어 설명

- I.1 스택
- I.2 서브층
- I.3 어텐션 헤드

지난 수십 년 동안 CNNs(Convolutional Neural Network), RNNs(Recurrent Neural Networks)를 비롯한 다양한 인공지능망(Artificial Neural Networks, ANNs)이 개발되었다. 이들을 설명할 땐 공통적인 어휘들이 사용된다.

트랜스포머는 새로운 용어들을 도입하고 기존의 용어들을 다르게 사용한다. 이 부록은 트랜스포머를 간단히 살펴봄으로써 트랜스포머의 용어를 설명한다.

트랜스포머 아키텍처는 딥러닝의 산업적인 활용을 고려하여 설계되었다. 트랜스포머는 병렬 처리에 적합한 구조를 가지고 있으며, 하드웨어를 최적화하기 위한 조건을 충족한다. 일례로 구글은 트랜스포머의 스택(stack) 구조를 활용하여 부동 소수점 정밀도를 적게 요구하는 도메인 특화 하드웨어를 설계했다.

트랜스포머 모델을 설계한다는 것은 하드웨어를 고려하는 것을 의미한다. 트랜스포머 아키텍처는 소프트웨어와 하드웨어를 최적화를 동시에 고려했다.

이 부록에서는 몇몇 신경망 용어를 새롭게 정의한다.

I.1

스택

그림 I.1처럼, 스택(stack)은 기존 딥러닝 모델과 다른 동일한 크기의 층(layer)을 쌓아 올린 형태이다. 스택은 아래에서 위로(bottom to top) 연결되어 있으며 인코더(encoder)나 디코더(decoder)가 될 수 있다.

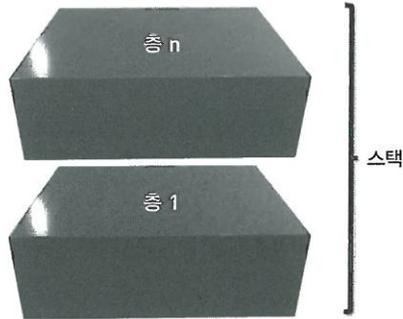


그림 I.1 층이 쌓인 스택

트랜스포머의 스택은 위 층으로 갈수록 더 많은 것을 학습한다. 사람의 기억처럼 각 층은 다음 층으로 학습한 내용을 전달한다.

스택이 뉴욕의 엠파이어 스테이트 빌딩이라고 상상해 보자. 아래 층에서는 볼 수 있는 것이 많지 않다. 하지만 더 높은 층으로 올라갈수록 더 많이 더 멀리 볼 수 있다. 꼭대기에 도착하면 맨해튼의 환상적인 경치를 볼 수 있다!

I.2

서브 층

각 층에는 그림 I.2와 같은 서브 층(sublayer)이 있다. 서로 다른 층에 있는 각 서브 층은 동일한 구조를 가지고 있어, 하드웨어를 최적화하기 유리하다.

오리지널 트랜스포머에는 아래에서 위로 연결된 두 서브 층이 있다.

- NLP와 하드웨어 최적화를 위해 설계된 셀프-어텐션(self-attention) 서브 층
- 전통적인 순방향 네트워크(feedforward network)를 약간 변경한 서브 층

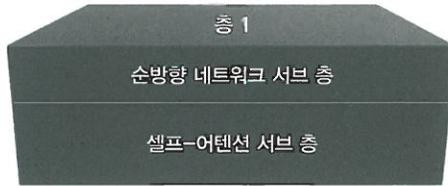


그림 1.2 두 서브 층

I.3 어텐션 헤드

셀프-어텐션 서브 층은 헤드(head)라고 불리는 n 개의 독립적이고 동일한 층으로 나뉜다. 오리지널 트랜스포머의 헤드는 8개이다.

그림 1.3은 트랜스포머의 산업화된 구조와 하드웨어 적합성을 보여주기 위해 헤드를 프로세서(processor)로 표현했다.

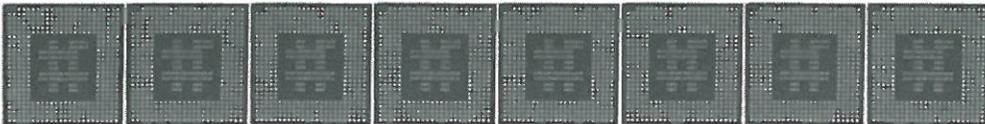


그림 1.3 셀프-어텐션 서브 층의 헤드

그림 1.3은 병렬처리에 대한 트랜스포머의 강점을 나타내기 위해 어텐션 헤드를 마이크로프로세서(microprocessor)로 표현했다.

트랜스포머의 아키텍처는 NLP와 하드웨어 최적화에 모두 적합하다.

부록 II

트랜스포머 모델의 하드웨어 제약사항

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

최적화된 하드웨어가 없다면 트랜스포머는 존재하지 못했다. 메모리와 디스크 관리 역시 중요하지만, 컴퓨팅 성능은 가장 중요한 전제 조건이다. GPU가 없었다면 **2장, 트랜스포머 모델 아키텍처 살펴보기**에서 살펴본 오리지널 트랜스포머를 학습할 수 없었을 것이다. GPU는 트랜스포머를 효율적으로 만드는 가장 중요한 요소이다.

이 부록은 **3장, BERT 모델 미세 조정하기**와 관련하여 GPU의 중요성을 다음 세 가지 단계로 설명한다.

- 트랜스포머의 아키텍처와 규모
- CPU vs GPU
- 파이토치를 예시로 GPU 사용 방법 알아보기

II.1

트랜스포머의 아키텍처와 규모

2장, 트랜스포머 모델 아키텍처 살펴보기의 멀티-헤드 어텐션 아키텍처 섹션에서 발췌한 다음 내용으로, 하드웨어 기반 설계를 확인할 수 있다.

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 $d_{\text{model}}=512$ 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.

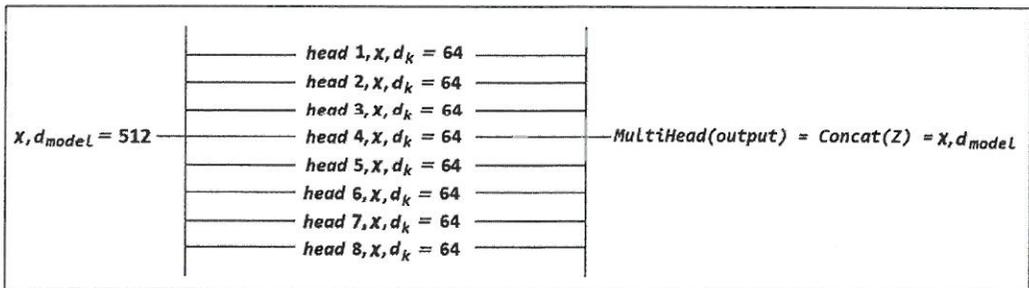


그림 II.1 멀티-헤드에 대한 표현

이제 헤드 여덟 개를 병렬로 실행할 수 있다.

이 부분은 서로 다른 관점을 학습하기 위해 어텐션 헤드를 여덟 개로 나누었다고 설명하고 있다. 그런데 조금 더 깊이 생각해 보면 여기에는 병렬 처리를 활용하여 하드웨어를 최적화하려는 의도가 포함되어 있다.

브라운은 「Language Models are Few-Shot Learners」(<https://arxiv.org/abs/2005.14165>)에서 GPT의 구조를 묘사하며 트랜스포머가 하드웨어를 고려한 아키텍처를 가졌음을 언급했다.

모델은 데이터 전송을 최소화하기 위해 깊이와 차원을 고려하여 GPU에 맞게 분할된다. 계산 효율성과 GPU 및 모델 구조의 부하 분산을 고려하여 모델 아키텍처의 파라미터를 선택해야 한다.

트랜스포머의 구조(인코더 및 디코더)와 크기는 다양하다. 하지만 모두 병렬처리를 위한 하드웨어 제약 사항을 가지고 있다. 한 단계 더 나아가서 GPU가 특별한 이유를 알아보자.

II.2 GPU가 특별한 이유

2장, 트랜스포머 모델 아키텍처 살펴보기의 멀티-헤드 어텐션 아키텍처 섹션에서 GPU와 관련한 힌트를 더 찾을 수 있다.

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

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

여기서 얻을 수 있는 결론은 다음과 같다.

- 어텐션 헤드는 병렬 연산을 고려하여 설계되었다
- 어텐션 헤드는 행렬 곱셈에 기반하여 설계되었다

II.3

GPU는 병렬 연산을 위해 설계되었다

CPU(central processing unit)는 직렬 연산(serial processing)에 최적화 되어있다. 트랜스포머의 어텐션 헤드를 직렬로 학습한다면 시간이 너무 오래 걸릴 것이다. 연습을 위해 매우 작은 트랜스포머를 학습한다면 CPU도 문제 없겠지만, 최고수준의 모델을 학습하기는 불가능하다.

GPU(graphics processing unit)는 병렬 연산(parallel processing)에 최적화 되어있다. 트랜스포머 모델은 직렬 연산(CPU)이 아니라 병렬 연산(GPU)을 위해 설계되었다.

II.4

GPU는 또한 행렬 곱셈을 위해 설계되었다

엔비디아(NVIDIA)의 GPU는 행렬 연산을 빠르게 수행할 수 있는 텐서 코어(tensor core)를 가지고 있다. 트랜스포머를 포함한 대부분의 인공지능 알고리즘은 행렬 연산을 사용한다. 엔비디아의 GPU는 행렬 연산을 위해 고도로 최적화 되어있다. 다음 링크에서 더 많은 내용을 확인할 수 있다.

- <https://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/>
- <https://www.nvidia.com/ko-kr/data-center/tesla-p100/>

구글의 TPU(Tensor Processing Unit)도 엔비디아의 GPU와 유사하다. 텐서플로를 사용하면 TPU에서 텐서(tensor) 최적화가 가능하다.

- <https://cloud.google.com/tpu/docs/tpus>에서 TPU에 대한 정보를 확인할 수 있다.
- <https://www.tensorflow.org/guide/tensor>에서 텐서플로의 텐서에 대한 정보를 확인할 수 있다.

파라미터 110M개를 가진 $BERT_{BASE}$ 는 TPU 16개, 340M 파라미터의 $BERT_{LARGE}$ 는 TPU 64개로 학습되었다. <https://arxiv.org/abs/1810.04805>에서 BERT 학습에 대한 더 많은 정보를 확인할 수 있다.

트랜스포머의 아키텍처가 병렬 하드웨어에 적합함을 확인했다. 이제 GPU를 사용할 소스 코드의 구현 관점에서 좀 더 살펴보자.

II.5 GPU를 사용하는 코드

파이토치는 GPU를 관리할 수 있으며, 넘파이의 `np.arrays()`와 비슷한 텐서를 사용한다. 하지만 넘파이와 달리 텐서는 GPU에서 병렬처리를 지원한다.

텐서는 파이토치의 데이터가 GPU에서 분산될 수 있도록 한다.

https://pytorch.org/tutorials/intermediate/ddp_tutorial.html

3장의 노트북, `BERT_Fine_Tuning_Sentence_Classification_GPU.ipynb`에서 CUDA(Compute Unified Device Architecture)를 사용하여 엔비디아 GPU와 통신했다. CUDA는 GPU에서의 연산을 위한 엔비디아의 플랫폼이다. 이를 사용하여 소스 코드에 구체적인 명령어를 추가할 수 있다. <https://developer.nvidia.com/cuda-zone>에서 구체적인 내용을 확인할 수 있다.

`BERT_Fine_Tuning_Sentence_Classification_GPU.ipynb`에서는 CUDA 명령어를 사용하여 모델과 데이터를 엔비디아 GPU에 전송했다. 이 때 파이토치의 `torch.device`로 특정 디바이스(장치)를 지정할 수 있다.

<https://pytorch.org/docs/stable/notes/cuda.html>에서 더 많은 내용을 확인할 수 있다.

이제 파이토치에서 GPU 디바이스를 사용하는 방법을 알아보자. 디바이스를 선택하고, 데이터를 병렬화하고, 모델을 디바이스로 불러오고, 배치(batch) 데이터를 전송할 것이다. 각 항목은 디바이스를 사용하는 과정을 `BERT_Fine_Tuning_Sentence_Classification_GPU.ipynb`의 셀 번호와 함께 보여준다.

- 디바이스 선택(셀 3)

엔비디아 GPU의 CUDA를 사용할 수 있는지 확인한다. 사용할 수 없다면, CPU를 사용한다.

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
!nvidia-smi
```

- 데이터 병렬화(셀 16)

여러 개의 GPU를 사용한다면 병렬 컴퓨팅으로 모델을 분산할 수 있다.

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

- 모델을 디바이스로 불러오기(셀 16)

모델을 디바이스로 전송한다.

```
model.to(device)
```

- 디바이스에 학습과 검증(validation)을 위한 배치 전송하기(셀 20)

GPU에 배치 데이터를 전송한다. (1번부터 n번까지)

```
# GPU에 배치 데이터를 전송한다.
batch = tuple(t.to(device) for t in batch)
```

이어지는 섹션에서는, 서로 다른 세 가지 환경에서 노트북을 실행하며 트랜스포머가 GPU를 활용하는 모습을 살펴볼 것이다.

II.6 구글 코랩으로 GPU 테스트하기

이번 섹션은 GPU의 잠재력을 알아보기 위한 비공식 테스트를 진행한다. 계속해서 3장의 노트북, BERT_Fine_Tuning_Sentence_Classification_GPU.ipynb를 사용할 것이다.

코랩(Colab)에서 다음 세 가지 환경으로 노트북을 실행해 보았다.

- 구글 코랩의 무료 CPU
- 구글 코랩의 무료 GPU
- 구글 코랩 프로(pro)

II.7

구글 코랩의 무료 CPU

수백만에서 수십억 파라미터로 이루어진 트랜스포머 모델을 CPU로 미세 조정하기란 불가능하다. 트랜스포머는 병렬로 설계되었지만 CPU는 순차적으로 연산한다.

런타임(Runtime)-런타임 유형 변경(Change runtime type) 메뉴에서 하드웨어 가속기(hardware accelerator)를 설정할 수 있다. None (CPU), GPU, TPU

이번에는 그림 II.2처럼 None (CPU)를 선택한다.

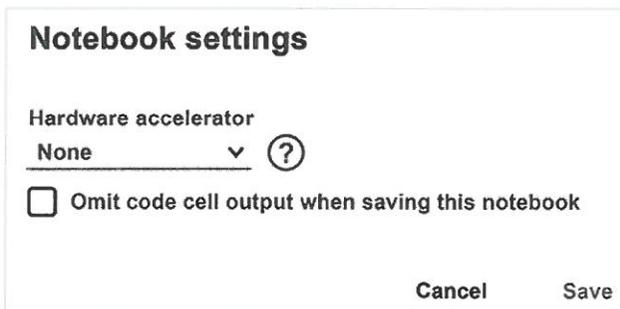


그림 II.2 하드웨어 가속기 설정

노트북의 학습 루프 단계를 실행하면 매우 느린 것을 볼 수 있다.

```
Epoch : 0% | | 0/4 [00:00<?, ?it/s]
```

그림 II.3 학습 루프

15분이 지나도 변화가 없다.

CPU는 병렬처리에 적합하지 않다. 하지만 트랜스포머는 병렬로 설계되었기 때문에 연습용 모델이라도 GPU가 적합하다.

II.7.1 구글 코랩의 무료 GPU

설정으로 돌아가서, 이번에는 GPU를 선택해보자.

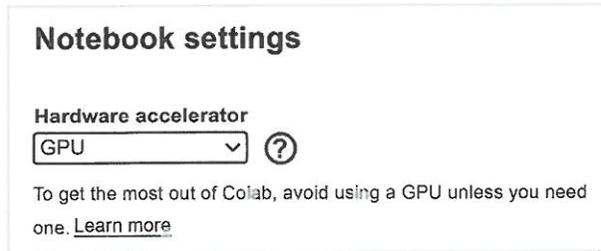


그림 II.4 GPU 선택

책이 쓰여진 시점에서는 NVIDIA K80과 CUDA 11.2가 할당된다.

```

+-----+
| NVIDIA-SMI 495.44      Driver Version: 460.32.03    CUDA Version: 11.2    |
+-----+-----+
| 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%        Default |
| N/A   39C   P8     27W / 149W |  0MiB / 11441MiB |           | N/A   |
+-----+-----+-----+-----+
+-----+
| Processes: |
| GPU   GI   CI        PID   Type   Process name                  GPU Memory |
| ID   ID   ID             |                   | Usage     |
+-----+-----+-----+-----+
| No running processes found |
+-----+
  
```

그림 II.5 활성화된 NVIDIA K80 GPU

학습 루프가 빨라졌으며 총 20분이 소요되었다. GPU는 비싼 자원이므로 구글 코랩은 하나의 GPU만 제공한다. 그림 II.6처럼 학습 루프는 적당한 속도로 수행된다.

```

Epoch: 0%|          | 0/4 [00:00<?, ?it/s]Train los
Epoch: 25%|██████    | 1/4 [04:58<14:56, 299.00s/it]
Train loss: 0.30048875815208026
Epoch: 50%|██████    | 2/4 [09:58<09:58, 299.42s/it]
Train loss: 0.1783793037950498
Epoch: 75%|██████    | 3/4 [14:58<04:59, 299.55s/it]
Train loss: 0.11217724044973425
Epoch: 100%|██████    | 4/4 [19:58<00:00, 299.57s/it]
  
```

그림 II.6 K80 GPU로 수행한 학습 루프

다음으로, 구글 코랩 프로를 사용하면 더 빠른 GPU가 제공되는지 확인해보자.

II.8 구글 코랩의 유료 GPU

다음 그림 II.7처럼, 이번에는 NVIDIA P100 GPU가 제공되었다. 바스와니의 「Attention is All You Need」에 의하면, 오리지널 트랜스포머는 8개의 NVIDIA P100으로 학습되었다. $10^6 \times 65$ 개의 파라미터를 가진 기본 모델을 GPU 8개로 학습하는데 12시간이 걸린 것이다.

NVIDIA-SMI 495.44		Driver Version: 460.32.03		CUDA Version: 11.2	
GPU Name	Persistence-M	Bus-Id	Disp.A	Volatile Uncorr. ECC	
Fan Temp Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util	Compute M.	MIG M.
0	Tesla P100-PCI-E...	Off	00000000:00:04.0	Off	0
N/A	41C P0	28W / 250W	2MiB / 16280MiB	0%	Default
					N/A

그림 II.7 P100 GPU를 제공하는 구글 코랩 프로의 가상머신

그림 II.8에서 보듯, 학습 시간이 매우 감소하여 10분 정도가 소요된다.

```
Epoch: 0%|          | 0/4 [00:00<?, ?it/s]Train lo
Epoch: 25%|███      | 1/4 [01:35<04:47, 95.71s/it]
Train loss: 0.3125095507168671
Epoch: 50%|█████    | 2/4 [03:11<03:11, 95.57s/it]
Train loss: 0.18029312002646478
Epoch: 75%|██████   | 3/4 [04:46<01:35, 95.51s/it]
Train loss: 0.11255507657296678
Epoch: 100%|███████ | 4/4 [06:22<00:00, 95.53s/it]
```

그림 II.8 P100 GPU로 수행한 학습 루프

부록 III

GPT-2를 사용한 일반 텍스트 완성

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

이 부록은 7장, GPT-3 엔진을 사용한 초인간 트랜스포머 등장에 있는 GPT-2를 사용한 일반 텍스트 완성 섹션을 자세하게 다룬다. 이 섹션에서는 일반 텍스트 완성(generic text completion)을 위한 GPT-2 트랜스포머 모델을 구현하는 방법을 설명한다.

이 노트북의 사용법을 7장에서 직접 읽거나 이번 부록에서 프로그램을 빌드하고 실행하여 GPT 모델의 작동 방식에 대한 보다 심도 있는 지식을 얻을 수 있다.

OpenAI_GPT_2 저장소를 복제(clone)하고 345M-파라미터 GPT-2 트랜스포머 모델을 다운로드한 후 이 모델과 상호 작용해 본다. 컨텍스트 문장을 입력하고 트랜스포머가 생성한 텍스트를 분석해 본다. 목표는 새로운 콘텐츠를 어떻게 생성하는지 확인하는 것이다.

이 섹션은 9단계로 나뉜다. 구글 코랩에서 OpenAI_GPT_2.ipynb를 연다. 노트북은 이 책의 깃허브 저장소에서 부록 III 디렉터리에 있다. 노트북도 이 섹션과 동일한 9단계로 나뉜다.

셀을 순서대로 실행해 보자. 이 과정은 지루하지만, 결과는 만족스럽다. 코드 몇 줄로 GPT-3 엔진을 실행할 수 있다는 것을 확인했었다. 이 부록은 OpenAI가 제공하는 GPT-2를 가공 없이 사용해 GPT-2 모델의 동작을 확인한다.

허깅페이스에는 GPT-2 모델을 캡슐화하는 래퍼(wrapper)가 있다. 이 래퍼는 OpenAI API의 대안으로 유용하다. 그러나 이 부록의 목표는 GPT-2 모델의 기본 구성 요소의 복잡성을 피하는 것이 아니라 이를 탐구하는 것이다!

마지막으로, 결과를 얻기 위해 코드 한 줄로 호출하는 대신, 저수준의 GPT-2를 직접 실행할 것이다. 따라서 이미 구성되어 있는 패키지(OpenAI GPT-3 API, 허깅페이스 래퍼 등)는 사용하지 않는다. 우리는 GPT-2의 아키텍처를 이해하기 위해 직접 코드를 작성한다. 그 결과 일부 사용 중단 메시지가 표시될 수 있다. 이런 노력은 4차 산업 시대에 전문가가 되는 데에 충분한 가치가 있다.

GPU를 활성화하는 것부터 시작해 보자.

III.1

1단계: GPU 활성화

GPT-2 345M 파라미터 트랜스포머 모델을 학습하려면 GPU를 활성화해야 한다.

GPU를 활성화하려면 노트 설정(Notebook settings) 메뉴로 이동하여 VM(가상 머신)을 최대한 활용한다.

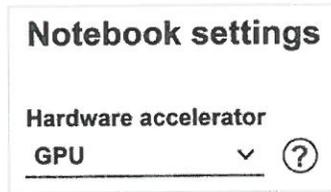


그림 III.1 GPU 하드웨어 가속기

GPU를 활성화하는 것이 더 나은 성능을 위한 전제 조건이며, 이를 통해 GPT 트랜스포머의 세계에 접근할 수 있다. 이제 OpenAI GPT-2 저장소를 복제해 보자.

III.2

2단계: OpenAI GPT-2 저장소 복제하기

현재 OpenAI를 통해 GPT-2를 다운로드할 수 있다. 다운로드가 중단되거나, 어쩌면 더 많은 리소스를 다운로드할 수 있을지도 모른다. 현재로서는 트랜스포머의 사용법이 매우 빠르게 변화하고 있기 때문에 주요 연구소를 포함해 그 누구도 시장이 어떻게 진화할지 예측할 수 없다.

VM에 OpenAI의 깃허브 디렉토리를 복제한다.

```
#@title 2단계: OpenAI GPT-2 저장소 복제하기
```

```
!git clone https://github.com/nshepperd/gpt-2
```

복제가 끝나면 파일 관리자에 저장소가 표시된다.

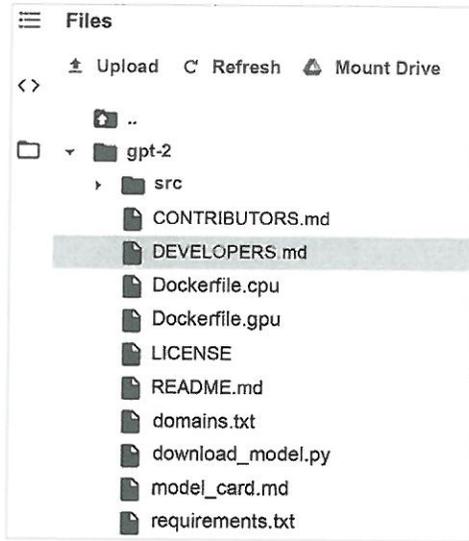


그림 III.2 복제된 GPT-2 저장소

src를 클릭하면 모델을 실행하는 데 필요한 OpenAI의 파이썬 파일이 설치되어 있는 것을 확인할 수 있다.

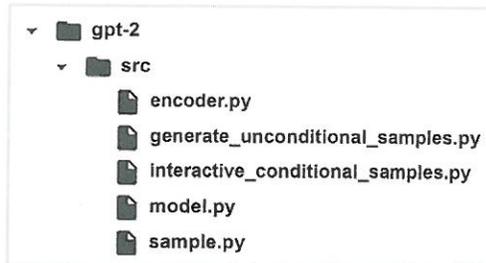


그림 III.3 모델 실행을 위한 GPT-2 파이썬 파일

학습 파일은 포함되어 있지 않은 것을 알 수 있다. 학습 파일은 부록 IV, GPT-2를 사용해 커스텀 텍스트 완성하기의 GPT-2 언어 모델 학습하기 섹션에서 GPT-2 모델을 학습할 때 설치할 것이다.

이제 요구 사항(requirements)을 설치해 보자.

III.3

3단계: 요구 사항 설치하기

다음 코드를 실행하면 요구 사항이 자동으로 설치된다.

```
#@title 3단계: 요구 사항 설치하기
import os # VM이 재시작할 때 필요하다면 os를 임포트한다
os.chdir("/content/gpt-2")
!pip3 install -r requirements.txt
```

셀 단위로 실행할 때는 VM을 다시 시작해야 하므로 os를 다시 임포트해야 할 수 있다. 이 노트북의 요구 사항은 다음과 같다.

- 명령줄 인터페이스(CLI) 생성을 위한 Fire 0.1.3
- 정규식 사용을 위한 regex 2017.4.5
- HTTP 라이브러리인 Requests 2.21.0
- 루프 진행률 표시를 위한 tqdm 4.31.1

노트북을 재시작하라는 메시지가 표시될 수 있다.

지금 재시작하지 말고, 텐서플로우의 버전을 확인할 때까지 기다리자.

III.4

4단계: 텐서플로우 버전 확인하기

2020년대에 들어서는 GPT 모델의 파라미터 수가 1,750억 개에 달해 슈퍼컴퓨터의 도움 없이는 자체적으로 학습하는 것이 불가능해졌다. 파라미터의 수는 계속 늘어날 것이다.

페이스북 AI, OpenAI, 구글 브레인과 같은 거대 기업의 연구소는 슈퍼트랜스포머를 향해 속도를 내고 있으며, 우리가 배우고 이해해야 하는 것들이 많아지고 있다. 하지만 안타깝게도 그들은 그들이 공유하는 모든 모델을 최신 버전에 맞게 업데이트할 시간이 없다.

이 노트북에서는 텐서플로우 1.x로 개발된 코드를 텐서플로우 2.x에서 실행할 수 있게 수정하였으며, 트랜스포머 모델을 직접 학습할 때 몇 가지 경고가 표시될 수 있다.

#@title 4단계: 텐서플로우 버전 확인하기

#코랩은 tf 2.x가 설치되어 있다.

#'런타임(Runtime)' -> '런타임 다시 시작(Restart runtime)' 메뉴를 사용해 런타임을 재시작한다.

```
import tensorflow as tf
print(tf.__version__)
```

출력은 다음과 같아야 한다.

```
TensorFlow 1.x selected.
1.15.2
```

tf 2.x 버전이 표시되는지 여부에 관계없이 셀을 다시 실행하여 확인한 다음 VM을 다시 시작한다. 계속하기 전에 이 셀을 다시 실행했는지 잘 확인한다.

TIP

프로세스 중에 텐서플로우 오류가 발생하면(경고는 무시), 이 셀을 다시 실행하고, VM을 다시 시작한 후 다시 실행하여 확인한다.

VM을 다시 시작할 때마다 이 작업을 수행한다. VM의 기본 버전은 tf.2이다.

이제 GPT-2 모델을 다운로드할 준비가 되었다.

III.5

5단계: 345M 파라미터 GPT-2 모델 다운로드하기

이제 학습된 345M 파라미터 GPT-2 모델을 다운로드한다.

```
#@title 5단계 345M 파라미터 GPT-2 모델 다운로드하기
# 코드를 실행하고 논향을 전달한다.
import os # 런타임이 재시작한 이후 실행한다.
os.chdir("/content/gpt-2")
!python3 download_model.py '345M'
```

모델 디렉터리의 경로는 다음과 같다.

```
/content/gpt-2/models/345M
```

여기에는 모델을 실행하는 데 필요한 정보가 있다.

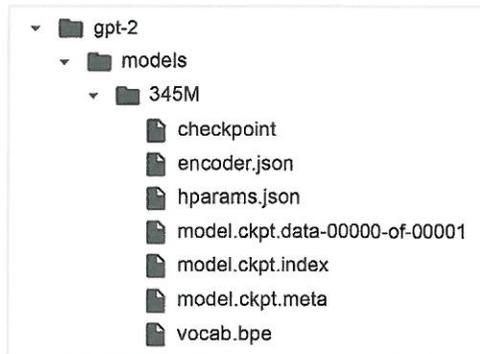


그림 III.4 345M 파라미터 모델의 GPT-2 파이썬 파일

hparams.json 파일에는 GPT-2 모델의 정의가 있다.

- “n_vocab”: 50257, 모델의 어휘 크기
- “n_ctx”: 1024, 컨텍스트 크기
- “n_embd”: 1024, 임베딩 크기
- “n_head”: 16, 헤드 수
- “n_layer”: 24, 층 수

encoder.json과 vocab.bpe에는 토큰화된 어휘와 BPE 단어 쌍이 포함되어 있다. 필요한 경우 잠시 시간을 내어 4장, RoBERTa 모델 처음부터 사전 학습하기에서 3단계: 토큰라이저 학습하기 하위 챕터를 다시 읽어보자.

checkpoint 파일에는 학습된 파라미터가 포함되어 있다. 예를 들어, **부록 IV, GPT-2를 사용해 커스텀 텍스트 완성하기**의 9단계: GPT-2 모델 학습하기 섹션에서 설명하는 것처럼 1,000 단계 동안 학습된 파라미터를 포함할 수 있다.

checkpoint 파일은 다른 세 개의 중요한 파일과 함께 저장된다.

- model.ckpt.meta는 모델의 그래프 구조를 설명한다. 여기에는 GraphDef, SaverDef 등이 포함되어 있다. `tf.train.import_meta_graph([path]+'model.ckpt.meta')`로 정보를 검색할 수 있다.
- model.ckpt.index는 문자열 테이블이다. 키는 텐서(tensor)의 이름이며, 값은 텐서의 메타데이터를 담고 있는 BundleEntryProto이다.
- model.ckpt.data에는 TensorBundle collection에 있는 모든 변수의 값이 포함되어 있다.

모델을 다운로드했다. 이제 모델을 활성화하기 전에 몇 가지 중간 단계를 거친다.

III.6 6~7단계: 중간 지침

이 섹션에서는 모델을 정의하고 활성화하는 8단계로 이어지는 중간 단계인 6단계, 7단계 및 7a단계를 진행한다.

모델과 상호 작용할 때 UTF로 인코딩된 텍스트를 콘솔에 출력한다.

```
#@title 6단계: UTF로 인코딩된 텍스트를 콘솔에 출력하기
!export PYTHONIOENCODING=UTF-8
```

src 디렉터리로 이동한다.

```
#@title 7단계: 프로젝트 소스 코드
import os # 런타임이 재시작한 이후 임포트한다.
os.chdir("/content/gpt-2/src")
```

이제 GPT-2 모델과 상호 작용할 준비가 되었다. 부록 IV, GPT-2를 사용해 커스텀 텍스트 완성하기의 GPT-2 언어 모델 학습하기 섹션에서 설명하는 것처럼 명령어를 사용하여 직접 실행할 수 있다. 하지만 이 섹션에서는 코드의 주요 측면을 살펴보고자 한다.

interactive_conditional_samples.py는 먼저 모델과 상호 작용하는 데 필요한 모듈을 임포트한다.

```
#@title 7a 단계: 상호 작용 가능한 조건부 샘플(src)
#상호 작용 가능한 조건부 샘플을 실행하는 프로젝트 소스 코드
# 파일명 - /content/gpt-2/src/interactive_conditional_samples.py
import json
import os
import numpy as np
#import tensorflow as tf
import tensorflow.compat.v1 as tf
```

모델 활성화로 이어지는 중간 단계를 거쳤다.

III.7 7b~8단계: 모델 가져오기 및 정의하기

이제 interactive_conditional_samples.py를 사용하여 모델과의 상호 작용을 활성화한다.

/content/gpt-2/src에 있는 세 개의 모듈을 임포트한다.

```
import model, sample, encoder
```

세 가지 모듈은 다음과 같다.

- `model.py`는 모델의 구조를 정의한다. 하이퍼파라미터, 다중 어텐션 `tf.matmul` 연산, 활성화 함수 및 기타 모든 속성을 정의한다.
- `sample.py`는 상호 작용을 처리하고 생성될 샘플을 제어하여, 더욱 의미 있는 토큰을 선택하도록 한다.

저화질로 이미지를 보는 것처럼 소프트맥스 값은 때때로 흐릿할 수 있다. `sample.py`에는 값을 더 선명하게 만들어 높은 확률을 높이고 낮은 확률을 부드럽게 만드는 `temperature`라는 변수가 포함되어 있다. `sample.py`는 Top-k 샘플링을 활성화할 수 있다. Top-k 샘플링은 예측된 시퀀스의 확률 분포를 정렬한다. 그리고 가장 높은 확률의 토큰부터 k 번째까지의 토큰을 선택한다. 낮은 확률의 토큰은 제외되어 모델이 품질이 낮은 토큰을 예측하는 것을 방지한다.

`sample.py`는 언어 모델링을 위해 Top-p 샘플링을 활성화할 수도 있다. Top-p 샘플링은 확률 분포를 정렬하지 않는다. 대신, 이 하위 집합의 확률 또는 가능한 시퀀스의 핵심(nucleus)의 합이 p를 초과할 때까지 높은 확률을 가진 단어를 선택한다.

- `encoder.py`는 정의된 모델, `encoder.json` 및 `vocab.bpe`를 사용하여 샘플 시퀀스를 인코딩한다. 여기에는 BPE 인코더와 텍스트 디코더가 모두 포함되어 있다.

각 모듈 파일을 더블클릭하여 열어 자세히 살펴볼 수 있다.

`interactive_conditional_samples.py`는 모델과 상호 작용하는 데 필요한 함수를 호출하여 `model.py`에서 모델을 정의하는 하이퍼파라미터와 `sample.py`에서 샘플 시퀀스 파라미터 등의 정보를 초기화한다. `encode.py`로 시퀀스를 인코딩하고 디코딩한다.

그런 다음 `interactive_conditional_samples.py`는 이 섹션의 5단계: 345M 파라미터 GPT-2 모델 다운로드 하기 하위 섹션에서 정의한 체크포인트 데이터를 복원한다.

`interactive_conditional_samples.py`를 두 번 클릭하여 탐색하고 파라미터를 실험해 볼 수 있다.

- `model_name`은 모델 이름(예: "124M" 또는 "345M")이며, `models_dir`에 의존한다.
- `models_dir`은 모델이 있는 디렉터리를 정의한다.
- `seed`는 무작위 생성기를 위한 임의의 정수를 설정한다. 시드는 결과를 재현하도록 설정할 수 있다.
- `nsamples`는 반환할 샘플 수이다. 0으로 설정하면 셸의 run 버튼을 두 번 클릭하거나 **Ctrl+M**을 누를 때까지 샘플을 계속 생성한다.
- `batch_size`는 배치의 크기를 결정하며 메모리와 속도에 영향을 준다.
- `length`는 생성된 텍스트의 토큰 수이다. none으로 설정하면 모델의 하이퍼파라미터에 의존한다.
- `temperature`는 볼츠만 분포의 수준을 결정한다. 값이 높으면 완성이 더 무작위로 이루어진다. 값이 낮으면 결과가 더 결정론적이게 된다.

- top_k는 각 단계에서 Top-k가 고려하는 토큰 수를 제어한다. 0은 제한이 없음을 의미한다. 400이 권장 값이다.
- top_p는 Top-p를 제어한다.

이 챕터의 프로그램에서 파라미터 시나리오는 다음과 같다.

- model_name = "345M"
- seed = None
- nsamples = 1
- batch_size = 1
- length = 300
- temperature = 1
- top_k = 0
- models_dir = '/content/gpt-2/models'

이러한 파라미터는 모델의 동작, 컨텍스트 입력에 의해 조절되는 방식, 텍스트 완성 시퀀스 생성에 영향을 미친다. 먼저 기본값으로 노트북을 실행한다. 그런 다음 프로그램을 두 번 클릭하고 편집한 다음 저장하여 코드의 파라미터를 변경할 수 있다. 변경 사항은 VM을 다시 시작할 때마다 삭제된다. 상호 작용 시나리오를 만들려면 프로그램을 저장하고 다시 불러온다.

이제 프로그램과 상호 작용할 준비가 되었다.

III.8 9단계: GPT-2와 상호 작용하기

이번에는 GPT-2 345M 모델과 상호 작용해 본다.

시스템이 실행될 때 더 많은 메시지가 표시될 수 있지만 이 노트북으로 모델을 실행할 것이다. 언젠가 이 노트북이 더 이상 동작하지 않는다면 GPT-3 엔진을 사용해야 할 수도 있고, 향후에 더 이상 사용되지 않을 수도 있는 허깅페이스 GPT-2 래퍼를 사용해야 할 수도 있다.

그렇지 않다면 GPT-2를 여전히 사용할 수 있으므로 모델과 상호 작용해 보자!

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')
```

몇 가지 컨텍스트를 입력하라는 메시지가 표시된다.



그림 III.5 텍스트 완성을 위한 컨텍스트 입력

표준 GPT-2 모델이므로 원하는 모든 유형의 컨텍스트를 사용해 볼 수 있다.

임마누엘 칸트가 쓴 문장을 사용해 볼 수 있다.

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, 그 인식의 한 영역에서, 그것이 거절할 수 없는 질문들을 고려하도록 요구된다.)

[Enter]를 눌러 텍스트를 생성한다. GPT-2 모델은 데이터셋에 대해 학습되지 않았고 어쨌든 확률 모델을 실행하고 있기 때문에 출력은 무작위적이다.

GPT 모델이 생성한 처음 몇 줄을 살펴보자.

"We may grant to this conception the peculiarity that it is the only causal logic.(우리는 이 개념에 유일한 인과 논리라는 특이성을 부여할 수 있다)

In the second law of logic as in the third, experience is measured at its end: apprehension is afterwards closed in consciousness.(논리의 두 번째 법칙이나 세 번째 법칙에서처럼, 경험은 그 끝에서 측정된다. 인식은 그 후 의식 속에서 닫힌다.)

The solution of scholastic perplexities, whether moral or religious, is not only impossible, but your own existence is blasphemous."(도덕적이든 종교적이든 학문적인 혼란을 해결하는 것은 불가능할 뿐만 아니라, 당신 자신의 존재 자체가 신성모독이다.)

셀을 중지하려면 셀의 실행 버튼을 두 번 클릭한다.

[Ctrl]+[M]을 눌러 텍스트 생성을 중지할 수도 있지만 코드가 텍스트로 변환될 수 있으므로 프로그램 셀에 다시 복사해야 한다.

출력에서 몇 가지 사실을 관찰할 수 있다.

- 우리가 입력한 컨텍스트가 모델에서 생성된 출력에 조건을 지정했다.
- 컨텍스트는 일종의 시연이다. 컨텍스트를 입력하면 모델은 파라미터 수정 없이도 무엇을 말해야 할 지를 익힌다.
- 텍스트 완성은 컨텍스트에 의해 조절된다. 이를 통해 미세 조정이 필요 없는 트랜스포머 모델의 문을 열었다.
- 의미론적 관점에서 볼 때 결과물은 더 흥미로울 수 있다.
- 문법적인 관점에서는 출력 결과가 설득력이 있다.

부록 IV, GPT-2를 사용해 커스텀 텍스트 완성하기에서 사용자 지정 데이터셋으로 모델을 학습하면 더 인상적인 결과를 얻을 수 있는지 확인할 수 있다.

III.9 참고 문헌

- OpenAI GPT-2 깃허브 저장소: <https://github.com/openai/gpt-2>
- N Shepperd 깃허브 저장소: <https://github.com/nshepperd/gpt-2>

부록 IV

GPT-2를 사용해 커스텀 텍스트 완성하기

IV.1 GPT-2 모델 학습하기

IV.2 참고 문헌

이 부록은 7장, GPT-3 엔진을 사용한 초인간 트랜스포머 등장과 관련되어 있으며, GPT-2 모델을 사용한 텍스트 완성 기능을 커스터마이징하는 방법을 설명한다.

이 부록은 12 단계로 나눠 GPT-2 모델을 구축하고 학습시키며 상호 작용하는 방법을 보여준다.

이 부록의 깃허브 저장소에 있는 Training_OpenAI_GPT_2.ipynb를 실행해 보자. 노트북도 이 부록과 동일한 12 단계와 셀로 나누어져 있음을 확인할 수 있다.

셀 단위로 노트북을 실행해 보자. 과정은 지루할 수 있지만, 복제된 OpenAI GPT-2 저장소에서 생성된 결과는 만족스러울 것이다. GPT-3 API나 허깅페이스는 사용하지 않을 것이다.

모델을 직접 구축, 학습하는 과정은 번거로울 수 있다. 일부 오류 메시지가 표시될 수 있지만, 래퍼나 API가 아닌 모델 내부를 들여다보아야만 한다. 이러한 노력은 가치가 있을 것이다.

우선 GPU를 활성화하는 것부터 시작해 보자.

[설명 추가]

Training_OpenAI_GPT_2.ipynb은 텐서플로우 1x 버전에 작동한다. 하지만 2024년 5월 현재, 구글 코랩은 텐서플로우 1x 버전을 지원하지 않는다. 두 가지 해결책이 있다.

- (1) Training_OpenAI_GPT_2.ipynb을 텐서플로우 1x가 설치된 로컬 머신에 받아 실행하기
 - (2) 구글 코랩에서 깃허브 내에 BONUS 디렉터리에 존재하는 GPT_2_and_ChatGPT_the_Origins.ipynb을 실행하기
- GPT_2_and_ChatGPT_the_Origins.ipynb은 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 챗봇 미세 조정 방법

이 부록의 내용은 tf 1x가 설치된 로컬 머신에서 실행하는 것으로 가정한다.

IV.1

GPT-2 모델 학습하기

이 섹션에서는 커스텀 데이터셋을 사용해 GPT-2 모델을 학습시킬 것이다. 그런 다음 학습된 모델을 직접 사용해 볼 것이다. 데이터는 4장, RoBERTa 모델 처음부터 사전 학습하기에서 사용한 것과 동일한 kant.txt를 사용할 것이다.

노트북을 열고 셀을 하나씩 실행해 보자.

IV.1.1 1단계: 사전 준비 사항

이 섹션에서 참조하는 파일들은 이 책의 깃허브 저장소의 Appendix IV 디렉터리에 있다.

- 구글 코랩에서 실행 중이라면 구글 코랩 노트북의 런타임(Runtime) 메뉴에서 GPU를 활성화한다. 이 과정은 부록 III, GPT-2를 사용한 일반 텍스트 완성의 1단계 GPU 활성화에 설명된 내용과 같은 내용이다.
- 다음 파이썬 파일을 구글 코랩에 업로드한다. train.py, load_dataset.py, encode.py, accumulate.py, memory_saving_gradients.py
- 이 파일들은 원래 N Shepperd의 깃허브 저장소(<https://github.com/nshepperd/gpt-2>)에서 가져온 것이다. 이 책의 깃허브 저장소의 AppendixIV/gpt-2-train_files 경로에서 또한 다운로드 할 수 있다.
- N Shepperd의 깃허브 저장소에서는 GPT-2 모델을 학습하기 위해 필요한 파일들을 제공한다. N Shepperd의 깃허브 저장소를 복제하는 것 대신에 OpenAI의 저장소를 복제할 것이다. 그리고 N Shepperd의 깃허브 저장소에서 필요한 다섯 개의 학습 파일을 추가할 것이다.
- dset.txt를 구글 코랩에 업로드한다. 데이터셋의 이름은 dset.txt이다. 따라서, 이 파일의 내용만 교체하면 별도의 프로그램 수정 없이 커스텀 데이터셋을 사용할 수 있다.
- 위 데이터셋은 이 부록에서 사용하는 깃허브 저장소의 gpt-2-train_files 디렉터리에 있는 데이터셋이다. 4장, RoBERTa 모델 처음부터 사전 학습하기에서 사용한 것과 동일한 kant.txt 파일이다.

이제 학습 과정 시작 단계를 하나씩 실행해 보자.

IV.1.2 2단계 ~ 6단계: 학습 과정 시작 단계

이번 하위 섹션에서는 앞서 부록 III, GPT-2를 사용한 일반 텍스트 완성에서 상세히 설명한 단계 2에서 단계 6을 간단하게 훑어볼 것이다. 그런 다음 데이터셋과 모델을 프로젝트 디렉터리로 복사할 것이다.

이번에는 N Shepperd의 저장소가 아닌 OpenAI의 GPT-2 저장소를 복제할 것이다.

```
#@title 2단계: OpenAI GPT-2 저장소 복제하기
#!git clone https://github.com/nshepperd/gpt-2.git
!git clone https://github.com/openai/gpt-2.git
```

앞서 이미 N Shepperd의 디렉터리에서 GPT-2 모델을 학습하는 데 필요한 파일들을 업로드했다.

이제 프로그램 실행에 필요한 패키지들을 설치하도록 한다.

```
#@title 3단계: 필요 패키지 설치하기
import os #VM이 재시작할 때 import os가 필요하다
os.chdir("/content/gpt-2")
!pip3 install -r requirements.txt
```

이 노트북에는 토폴로지 정렬 알고리즘인 toposort가 필요하다.

```
!pip install toposort
```

TIP

노트북에 요구 사항을 설치한 후에는 노트북을 다시 시작하면 안된다. 세션 중에 VM 재시작을 한 번만 수행하기 위해서 텐서플로우 버전을 확인할 때까지 기다리도록 한다. 확인 후에, 필요하다면 재시작을 하도록 한다. 번거로운 일이지만, 래퍼(wrapper)와 API 이상의 코드 내부로 들어가는 것은 가치가 있는 일이다.

지금은 텐서플로우 버전을 확인하여 1.x 버전을 실행 중인지 확인한다.

```
#@제목 4단계: 텐서플로우 버전 확인하기
#코랩에는 tf 2.x 버전이 설치되어 있다.
#'런타임(Runtime)' -> '런타임 다시 시작(Restart runtime)...'을 사용해 런타임을 재시작한다.
%tensorflow_version 1.x
import tensorflow as tf
print(tf.__version__)
```

텐서플로우 1.x 버전이 실행중인지 분명하게 확인하고 이후 내용을 진행하도록 한다.

이제 학습시킬 117M 파라미터 GPT-2 모델을 다운로드한다.

```
#@title 5단계: 117M 파라미터 사이즈의 GPT-2 모델 다운로드하기
# 코드를 실행시키고 인수를 전달한다.
import os # 런타임이 재시작한 이후 실행한다.
os.chdir("/content/gpt-2")
!python3 download_model.py '117M' #모델 디렉터리를 생성한다.
```

그리고 데이터셋과 117M 파라미터의 GPT-2 모델을 src 디렉터리에 복사한다.

```
#@title 6단계: 프로젝트 리소스를 src로 복사하기
!cp /content/dset.txt /content/gpt-2/src/
!cp -r /content/gpt-2/models/ /content/gpt-2/src/
```

앞으로는 모델을 학습시키는 데 필요한 모든 자료들을 src 프로젝트 디렉터리에 그룹화하는 것이 목표다. 이제 N Shepperd 학습 데이터를 확인해본다.

IV.1.3 7단계: N shepperd 학습 데이터

학습 데이터는 N Shepperd의 깃허브 저장소에서 가져왔다. 본 부록의 1단계 “사전 준비 사항”에서 이 파일들을 업로드했다. 이제 프로젝트 디렉터리로 데이터를 복사할 것이다.

```
#@title 7단계: N Shepperd 학습 데이터 복사하기
#참조 깃허브 저장소: https://github.com/nshepperd/gpt-2
import os # 런타임 재시작 이후에 임포트한다
!cp /content/train.py /content/gpt-2/src/
!cp /content/load_dataset.py /content/gpt-2/src/
!cp /content/encode.py /content/gpt-2/src/
!cp /content/accumulate.py /content/gpt-2/src/
!cp /content/memory_saving_gradients.py /content/gpt-2/src/
```

학습 데이터를 사용할 준비가 되었다. 이제 encode.py부터 시작하여 데이터를 탐색해 보겠다.

IV.1.4 8단계: 데이터셋 인코딩하기

학습 전에 데이터셋을 인코딩해야 한다. 구글 코랩에서 encode.py 파일을 더블클릭하여 내용을 확인해 볼 수 있다.

encode.py는 load_dataset.py에 있는 load_dataset 함수를 호출하여 dset.txt를 불러온다.

```
from load_dataset import load_dataset
.../...
chunks = load_dataset(enc, args.in_text, args.combine, encoding=args.
encoding)
```

또한 encode.py는 데이터셋을 인코딩하기 위해 OpenAI의 인코딩 프로그램을 로드한다.

```
import encoder
.../...
enc = encoder.get_encoder(args.model_name,models_dir)
```

인코딩된 데이터셋을 넘파이 배열 형태로 out.npz에 저장한다. npz는 인코더에 의해 생성된 배열의 넘파이 압축 파일이다.

```
import numpy as np
np.savez_compressed(args.out_npz, *chunks)
```

다음 셀을 실행하면 데이터셋이 로드되고 인코딩되며 out.npz에 저장될 것이다.

```
#@title 8단계: 데이터셋 인코딩하기
import os # 런타임 재시작 이후에 임포트한다
os.chdir("/content/gpt-2/src/")
model_name="117M"
!python /content/gpt-2/src/encode.py dset.txt out.npz
```

이제 GPT-2 117M 모델을 학습시킬 준비가 되었다.

IV.1.5 9단계: GPT-2 모델 학습하기

우리의 데이터셋을 사용해 GPT-2 117M 모델을 학습할 것이다. 인코딩된 데이터셋의 파일명을 프로그램에 전달한다.

```
#@title 9단계: 모델 학습하기
#모델은 1000 스텝 이후 저장된다.
import os # 런타임 재시작 이후에 임포트한다
os.chdir("/content/gpt-2/src/")
!python train.py --dataset out.npz
```

셀을 실행하면 직접 멈추기 전까지 학습된다. 모델은 1,000 스텝 후에 저장된다. 학습이 1,000 스텝을 초과하면 멈추도록 한다. 저장된 모델 체크포인트는 /content/gpt-2/src/checkpoint/run1에 있다. 이 파일들의 목록은 노트북의 “10A단계: 학습 파일 복사” 셀에서 확인할 수 있다.

셀의 실행 버튼을 두 번 클릭해 학습을 중지할 수도 있다. 학습이 중지되면 학습된 파라미터가 함께 저장될 것이다.

또한 **Ctrl+M**을 눌러 1,000 스텝 후에 모델 학습을 중지할 수 있다. 프로그램이 멈추고 학습된 파라미터가 저장될 것이다. 이후 아래와 같은 메시지가 출력될 것이다.

```
@title Step 9: Training the Model
Model saved after 1000 steps
```

그림 IV.1 학습된 GPT-2 모델을 자동으로 저장하기

/content/gpt-2/src/memory_saving_gradients.py와 /content/gpt-2/src/accumulate.py를 사용해 옵티마이저와 기울기(gradient)를 관리한다.

train.py에는 학습 프로세스를 수정할 수 있는 파라미터가 포함되어 있다. 일단 그대로 노트북을 실행해 본다. 그런 다음 원한다면 학습 파라미터를 수정해 실험해 보고 더 나은 결과를 얻을 수 있는지 확인해 볼 수 있다.

GPT-2 모델이 학습 과정에서 생성한 샘플들을 확인해 볼 수 있다. 학습 중 생성된 샘플에서 깨달음을 얻을 수 있었다.

```
The world is not a thing in itself, but is a representation of the world
in itself.
```

(이 세상은 그 자체로 존재하는 것이 아니라 그 자체로 세상의 표상이다.)

인간이 만든 이 세계의 표상을 AI가 학습한다는 점이 흥미롭다!

실험을 계속해보자. 학습 모델을 위한 디렉터리를 만들어볼 차례다.

IV.1.6 10단계: 학습 모델 디렉터리 생성하기

이 섹션에서는 모델을 위한 임시 디렉터리를 생성하고 필요한 정보를 저장한 다음, 다운로드한 GPT-2 117M 모델의 디렉터리를 대체하기 위해 이름을 바꿀 것이다.

먼저 tgmodel이라는 임시 디렉터리를 생성한다.

```

#@title 10단계: 학습 모델 디렉터리 생성하기
#'tgmodel'이라는 이름으로 학습 모델 디렉터리를 생성한다
import os
run_dir = '/content/gpt-2/models/tgmodel'
if not os.path.exists(run_dir):
    os.makedirs(run_dir)

```

그런 다음 이 9단계: GPT-2 모델 학습하기에서 모델을 학습할 때 저장한 체크포인트 파일들을 복사한다.

```

#@title Step 10A: 학습 파일 복사하기
!cp /content/gpt-2/src/checkpoint/run1/model-1000.data-00000-of-00001 /
content/gpt-2/models/tgmodel
!cp /content/gpt-2/src/checkpoint/run1/checkpoint /content/gpt-2/models/
tgmodel
!cp /content/gpt-2/src/checkpoint/run1/model-1000.index /content/gpt-2/
models/tgmodel
!cp /content/gpt-2/src/checkpoint/run1/model-1000.meta /content/gpt-2/
models/tgmodel

```

tgmodel 디렉터리에는 이제 GPT-2 모델의 학습된 파라미터가 들어있다.

이러한 파일들의 내용에 대해서는 부록 III, GPT-2를 사용한 일반 텍스트 완성의 5단계: 345M 파라미터 GPT-2 모델 다운로드하기에서 설명한 바 있다.

이제 다운로드한 GPT-2 117M 모델로부터 하이퍼파라미터와 사전 파일을 가져올 것이다.

```

#@title 10B 단계: OpenAI GPT-2 117M 모델 파일 복사하기
!cp /content/gpt-2/models/117M/encoder.json /content/gpt-2/models/tgmodel
!cp /content/gpt-2/models/117M/hparams.json /content/gpt-2/models/tgmodel
!cp /content/gpt-2/models/117M/vocab.bpe /content/gpt-2/models/tgmodel

```

이제 tgmodel 디렉터리에는 커스터마이징된 GPT-2 117M 모델이 들어있다.

마지막 단계는 다운로드한 원래의 GPT-2 모델의 이름을 바꾸고 커스터마이징된 모델의 이름을 117M으로 설정하는 것이다.

```
#@title 10C 단계: 모델 디렉터리 이름 바꾸기
import os
!mv /content/gpt-2/models/117M /content/gpt-2/models/117M_OpenAI
!mv /content/gpt-2/models/tgmodel /content/gpt-2/models/117M
```

학습한 모델은 복제된 OpenAI GPT-2 저장소에서 동작할 것이다. 그럼, 모델과 상호 작용해 보자.

IV.1.7 11단계: 조건 없는 샘플 생성하기

이 섹션에서는 우리의 데이터셋으로 학습된 GPT-2 117M 모델과 상호 작용해 볼 것이다. 먼저 조건 없는 샘플을 생성해 보겠다. 그런 다음 조건이 주어졌을 때의 텍스트 완성 응답을 얻기 위해 단락을 입력해 볼 것이다.

먼저 조건 없는 샘플을 실행해 보자.

```
#@title 11단계: 조건 없는 샘플 생성하기
import os # 런타임 재시작 이후에 임포트한다.
os.chdir("/content/gpt-2/src")
!python generate_unconditional_samples.py --model_name '117M'
```

조건 없는 샘플 생성하기 때문에 컨텍스트 문장을 입력할 필요는 없다.

셀을 중지하려면 셀의 실행 버튼을 두 번 클릭하거나 **Ctrl**+**M**을 누르면 된다.

결과는 무작위이지만 문법적인 관점에서 의미가 있다. 의미적인 측면에서는 컨텍스트를 제공하지 않았기 때문에 그다지 흥미로운 결과는 아니다. 그래도 주목할만한 내용들이 있다. 모델은 게시물을 생성하고 있다. 제목을 작성하고, 날짜를 지정하며, 조직과 주소를 작성하고 주제를 생성하고 있다. 심지어 웹 링크를 상상하기도 한다.

처음 몇 줄의 내용은 정말 놀랍다.

제목: total_authority

카테고리:

스타일: 인쇄 가능

인용문:

가입일: 2013년 7월 17일

게시물 수: 0

오프토픽(Offtopic) 링크: "필수 연구, 동료 평가되는 논문을 지지하는 연구, 주장을 뒷받침하는 연구, 과학적 불확실성을 부당하게 수용하는 연구, 과학과 과학 참여에 대한 문을 여는 설득력 있는 연구",

링크: <https://groups.google.com/search?q=Author%3APj&src=ieKZP4CSg4GVWDSJtwQczgT WQhAWB07+tKWn0jzz7o6rP4lEy&ssl=cTheory%20issue1&fastSource=posts&very=device>

[오프라인](#)

가입일: 2014년 5월 11일

게시물 수: 1729

위치: 몬태나 지역 가입일: 2014년 5월 11일 게시물 수: 1729 위치: 몬태나

게시일: 2017년 12월 26일 금요일 오후 9:18 게시물 제목: 클릭

I. 확립된 검토 그룹의 개요

"A New Research Paradigm"과 선호 대안(BREPG) 그룹 주저자 존 오비(호주, 미국 및 열대 및 기후 변화 연구소 소속), 마르코 사오(중국, 사회학과/ 아약스, 열대 및 기후 변화 국제 연구소, 상하이 대학) 및 재키 구(미국/퍼시픽 대학교, NASA Interselicitas 및 Frozen Planet Research Center, 오크리지 국립 연구소). 오비 박사는 "우리의 결론은 새로운 아산-루비 연구가 나온 이후로 저널 출판사들이 과학적 내용을 더 자주 찾아보고 특정 전문가들과 더 많이 연결되려 한다는 것이다. 특정 전문가들이 우세한 상황에서 다른 전문가들의 연구를 검토하는 동료 평가 방식으로 책을 출판하는 관행이 조직의 여러 부분으로 확산하고 있다. 이것은 연구 결과나 데이터에 대한 주목도를 높이고, 연구의 신뢰성을 향상하는 데 도움이 된다."

조건 없는 임의의 텍스트 생성기 결과는 흥미로우나 설득력은 없다.

IV.1.8 12단계: 대화형 컨텍스트와 텍스트 완성 예제

이제 컨텍스트를 사용하는 샘플을 실행해 보자. 컨텍스트를 제공해 모델이 생각하는 방향을 유도하고 원하는 텍스트를 생성하도록 할 수 있다.

셀을 실행하고 결과를 확인해 보자.

```
#@title 12단계: 상호 작용 컨텍스트와 텍스트 완성 예제
import os # 런타임 재시작 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 시간을 내어 **부록 III, GPT-2를 사용한 일반 텍스트 완성의 9단계**, GPT-2와 상호 작용하기에서 파라미터에 대해 다시 살펴보도록 한다. 이제 프로그램에서 컨텍스트를 입력하라는 메시지를 표시할 것이다.

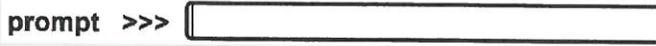


그림 IV.2 텍스트 완성을 위한 컨텍스트 입력

부록 III, GPT-2를 사용한 일반 텍스트 완성의 9단계, GPT-2와 상호 작용하기에서 했던 것처럼 임마누엘 칸트가 작성한 것과 동일한 단락을 입력해 보겠다.

인간의 이성은 인식의 한 영역에서 본성상 피할 수 없는 질문을 요청받는 운명을 지니고 있지만, 자신의 모든 능력을 넘어서 있어 스스로 답할 수도 없는 운명을 겪게 된다.

이전에 했던 것처럼 **[Enter]**를 눌러 텍스트를 생성한다. 실행할 때마다 내용은 다르지만 구조적이고 논리적인 글을 생성할 것이다. 이것이 트랜스포머가 매력적인 이유다. 이번에 생성된 결과는 인상적이다.

GPT-2 모델이 생성한 처음 몇 줄을 살펴보자.

"그렇다면, 생각하고 이해하도록 하는 마음의 힘이라는 것 외에는 이성의 본질이란 무엇인가? 이성(Reason)은 마음의 대상을 파악하고 이해하는, 자기 자신의 대상을 파악하는 것이다. 인간 이성(Reason)에 대한 이해와 그 이해 없이는 아무도 이성(Reason)에 대해 말할 수 없다. 그러나 인간 이성(Reason)에 의해 이해되기 때문에 모든 능력 중 가장 자연스러운 능력인 지식과 이해는 이성이 없는 것이 아니라 이해가 없기 때문에 가장 자연스럽다. 자연적 감각으로 이해하지 못할지라도 자연적인 감각으로 보고 인식하는 인간의 이성(Reason)은 우리가 이해하는 다른 모든 지식의 원천이다. 이성(Reason)은 이해하지 못하지만 우리가 인식하는 것은 이해하기 때문에 이해할 수 있는 것이 아니다. 이성(Reason)은 이해하지 않는 것이 아니라 우리가 이해하는 것만 이해하는 것이다."

셀을 중지하려면 셀의 실행 버튼을 두 번 클릭하거나 **Ctrl+M**을 입력한다.

학습된 GPT-2 모델이 만들어내는 텍스트와 사람이 만들어내는 텍스트의 완성도 차이가 없어 보인다. 또한 각 실행 시마다 다른 출력을 생성할 수도 있다. 예를 들어 다음과 같은 출력을 생성할 수 있을 것이다.

이 첫 번째 질문은, 어떤 주장의 진실을 전혀 모르는 정신이 어떻게 그 주장을 받아들일 수 있는지, 또는 받아들여야 하는지에 대한 것이다. 이 질문에 대한 답변은 명확하게 제시되었는데, 어떠한 정보 받지 않은 마음이 존재한다고 믿는 것은 불가능하다는 것이다.

"우리가 볼 수 있는 한, 어떤 사람의 마음 자체가 철학적 탐구의 주제는 아니다. 오히려 철학은 사람들이 중요하게 생각하는 것, 즉 그들이 가장 중요하다고 여기는 견해와 생각을 탐구한다. 그리고 사람들이 이러한 견해와 생각을 깊이 이해하고 통찰한다면, 그들은 그것을 보고, 더 깊이 이해하게 될 것이다."

필자는 학습한 모델이 철학, 이성, 논리에 대한 추상적인 문제를 다루는 것에 있어 많은 인간을 능가할 수 있다고 생각한다.

한계는 텍스트가 실행할 때마다 달라진다는 것이다. 따라서 훌륭해 보이지만 일상 생활에서 필요로 하는 모든 것을 충족시키지는 못한다.

- 지금까지의 실험을 통해 몇 가지 결론을 도출할 수 있다.
- 잘 학습된 트랜스포머 모델은 사람 수준의 완성도를 가진 텍스트를 생성할 수 있다.
- GPT-2 모델은 복잡하고 추상적인 추론에 관한 텍스트 생성에서 거의 인간 수준에 도달할 수 있다.
- 텍스트 컨텍스트(text context)는 모델에 기대하는 것을 제시함으로써 모델에 조건을 입력하는 효과적인 방법이다.
- 텍스트 완성(Completion)은 주어진 컨텍스트 문장에 기반해 텍스트를 생성하는 것을 말한다.
- 출력 결과물이 인간의 눈높이에 맞춰져 있지만, 모든 요구를 충족시킬 수 있다는 것을 의미하지는 않는다. 흥미롭지만 아직 모든 곳에 활용할 수는 없다.

조건이 되는 텍스트 컨텍스트를 입력해 텍스트 완성을 실험해 볼 수 있을 것이다. 또한 자체 데이터로 자신만의 모델을 학습시켜 볼 수도 있을 것이다. dset.txt의 내용을 교체하고 결과를 확인해 보도록 하자.

학습시킨 GPT-2 모델이 사람처럼 행동할 것임을 명심하자. 짧거나 불완전하고 흥미롭지 않거나 까다로운 컨텍스트를 입력하면 좋지 않은 결과를 얻을 수 있다. 이는 GPT-2가 실제 생활에서와 마찬가지로 우리에게 최선을 기대하기 때문이다.

IV.9 참고 문헌

- OpenAI GPT-2 깃허브 저장소: <https://github.com/openai/gpt-2>
- N Shepperd의 깃허브 저장소: <https://github.com/nshepperd/gpt-2>

부록 V

문제 정답

01. 우리는 3차 산업혁명 시대에 살고 있다.

참 / 거짓

거짓. 역사 속 시대는 겹치기 마련이다. 하지만 3차 산업혁명은 세상을 디지털화하는 것에만 집중한 반면 4차 산업혁명은 시스템, 기계, 로봇 등 모든 것을 연결하는 것에 집중한다.

02. 4차 산업혁명은 모든 것을 연결한다.

참 / 거짓

참. 이로 인해 사람의 개입이 필요했던 많은 의사 결정이 자동화되었다.

03. 4차 산업혁명 개발자는 때로는 해야 할 AI 개발이 없다.

참 / 거짓

참. 일부 프로젝트는 별다른 개발이 필요 없는 온라인 AI 서비스를 사용한다.

04. 4차 산업혁명 개발자는 트랜스포머 구현을 처음부터 해야 할 수도 있다.

참 / 거짓

참. 일부 프로젝트는 일반적인 온라인 서비스나 API가 충분하지 않을 수 있다. 프로젝트에 적합한 솔루션이 없는 것이다. 이때 개발자는 요구사항에 맞추어 모델을 처음부터 만들어야 한다.

05. 허깅페이스와 같은 트랜스포머 생태계를 두 개 이상 익힐 필요가 없다.

참 / 거짓

거짓. 회사의 정책에 따라 구글 클라우드 AI를 사용할 수도 마이크로소프트 애저(Azure)를 사용할 수도 있다. 허깅페이스를 사용하게 될 수도 있지만 우리는 미리 알 수 없으며 결정권도 없을 것이다.

06. 바로 사용할 수 있는 트랜스포머 API는 모든 요구 사항을 충족한다.

참 / 거짓

잘 학습된 경우라면 참. 트랜스포머가 충분히 학습되지 못했다면 거짓.

07. 회사는 개발자가 가장 잘 알고 있는 트랜스포머 생태계를 채택할 것이다.

참 / 거짓

거짓. 회사가 꼭 개발자의 의견을 수용하는 것은 아니다. 따라서 최대한 다양한 생태계를 경험해 보는 것이 좋다.

08. 클라우드 트랜스포머는 주류가 되고 있다.

참 / 거짓

참.

09. 트랜스포머 프로젝트는 노트북(laptop)에서 실행될 수 있다.

참 / 거짓

프로토타입이라면 참, 수천 명의 사용자가 있는 프로젝트라면 거짓.

10. 4차 산업혁명 인공지능 전문가는 좀 더 유연해질 것이다.

참 / 거짓

참.

V.2

2장, 트랜스포머 모델 아키텍처 살펴보기

01. NLP 트랜스덕션은 텍스트 표현을 인코딩 또는 디코딩할 수 있다.

참 / 거짓

참. NLP는 글 또는 발화 형태의 시퀀스를 숫자로 인코딩하고 텍스트로 디코딩하는 트랜스덕션 작업이다.

02. 자연어 이해는 자연어 처리의 하위 분야이다.

참 / 거짓

참.

03. 언어 모델링 알고리즘은 입력 시퀀스에 기반하여 단어를 생성한다.

참 / 거짓

참.

04. 트랜스포머는 LSTM에 CNN을 추가한 모델이다.

참 / 거짓

거짓. 트랜스포머에는 LSTM이나 CNN이 없다.

05. 트랜스포머에는 LSTM 층 또는 CNN 층이 없다.

참 / 거짓

참.

06. 어텐션은 시퀀스의 마지막 토큰뿐만 아니라 모든 토큰을 검사한다.

참 / 거짓

참.

07. 트랜스포머는 위치 인코딩 대신 고정된 위치 벡터를 사용한다. 참 / 거짓
거짓. 트랜스포머는 위치 인코딩을 사용한다.
08. 트랜스포머에는 순방향 네트워크를 가지고 있다. 참 / 거짓
참.
09. 트랜스포머 디코더의 마스크드 멀티-헤드 어텐션은 처리중인 시퀀스의 나머지 뒷 부분을 알고리즘이 볼 수 없도록 한다. 참 / 거짓
참.
10. 트랜스포머는 멀리 떨어진 단어간의 관계를 LSTM보다 잘 파악한다. 참 / 거짓
참.

V.3

3장, BERT 모델 미세 조정하기

01. BERT는 Bidirectional Encoder Representations from Transformers를 축약한 단어이다. 참 / 거짓
참.
02. BERT는 두 단계 프레임워크이다. 첫 번째 단계는 사전 학습이고 두 번째 단계는 미세 조정이다. 참 / 거짓
참.
03. BERT 모델 미세 조정은 파라미터를 처음부터 학습하는 것을 의미한다. 참 / 거짓
거짓. BERT 미세 조정은 사전 학습된 파라미터로 시작한다.

04. BERT는 모든 다운스트림 작업을 사전 학습한다. 참 / 거짓
거짓.
05. BERT는 Masked Language Modeling (MLM, 마스크드 언어 모델링)을 사용하여 사전 학습한다. 참 / 거짓
참.
06. BERT는 Next Sentence Predictions (NSP, 다음 문장 예측하기)을 사용하여 사전 학습한다. 참 / 거짓
참.
07. BERT는 수학적 함수를 사전 학습한다. 참 / 거짓
거짓.
08. 질문-답변(question-answer) 작업은 다운스트림 작업이다. 참 / 거짓
참.
09. BERT 사전 학습 모델에는 토큰화가 필요하지 않다. 참 / 거짓
거짓.
10. BERT 모델 미세 조정은 사전 학습보다 시간이 덜 걸린다. 참 / 거짓
참.

01. RoBERTa 는 바이트 단위 바이트 쌍 인코딩 토큰라이저를 사용한다.

참 / 거짓

참.

02. 학습된 허깅페이스 토큰라이저는 merges.txt 와 vocab.json을 생성한다.

참 / 거짓

참.

03. RoBERTa는 토큰 타입 ID를 사용하지 않는다.

참 / 거짓

참.

04. DistillBERT 는 6개 층과 12개 헤드를 가진다.

참 / 거짓

참.

05. 8천만 개 파라미터를 가진 트랜스포머 모델은 거대한 축에 속한다.

참 / 거짓

거짓. 8천만 개의 파라미터는 작은 모델이다.

06. 토큰라이저는 학습할 수 없다.

참 / 거짓

거짓. 토큰라이저도 학습할 수 있다.

07. BERT 류(BERT-like)의 모델은 6개의 디코더 층을 가진다.

참 / 거짓

거짓. BERT는 디코더가 아니라 인코더 층으로 이루어져 있다.

08. 마스크드 언어 모델(Masked Language Model, MLM)은 문장 내의
마스크된 토큰을 예측한다.

참 / 거짓

참.

09. BERT 류의 모델은 셀프-어텐션 서브 층을 가지고 있지 않다.

참 / 거짓

거짓. BERT는 셀프-어텐션 층을 가지고 있다.

10. 데이터 콜레이터는 역전파에 도움이 된다.

참 / 거짓

참.

V.5

5장, 트랜스포머를 사용한 다운스트림 NLP 작업

01. 기계 지능은 인간과 동일한 데이터를 사용해 예측을 수행한다.

참 / 거짓

참 또는 거짓.

참. 방대한 양의 데이터에서 의미를 추출하거나 인간에게 수 세기가 걸리는 작업을 수행하는 경우에는 기계 지능은 인간 지능을 능가한다

거짓. 자연어 이해에 있어서 인간은 감각을 사용하여 더 많은 정보를 얻는다. 기계 지능은 인간이 제공해준 정보만을 사용한다.

02. SuperGLUE는 GLUE보다 더 어려운 NLP 과제이다.

참 / 거짓

참.

03. BoolQ는 예-아니오를 답하는 작업이다.

참 / 거짓

참.

04. WiC는 “Words in Context”의 약자이다.

참 / 거짓

참.

05. 텍스트 함의 인식(RTE)은 하나의 시퀀스가 다른 한 시퀀스를 함의하는지 판단하는 과제이다.

참 / 거짓

참.

06. 워노그라드 스키마는 동사의 철자가 올바른지 판단하는 과제이다. 참 / 거짓

거짓. 워노그라드 스키마는 대명사의 대상을 구분하는 과제이다.

07. 트랜스포머 모델은 현재 GLUE와 SuperGLUE의 최상위 순위를 차지했다. 참 / 거짓

참.

08. 인간 기준값은 변할 수 있다. SuperGLUE에 의해 더 어려운 기준으로 바뀌었다. 참 / 거짓

참.

09. 트랜스포머 모델은 SuperGLUE의 인간 기준값을 절대 넘지 못할 것이다. 참 / 거짓

참 또는 거짓.

거짓. GLUE에서 트랜스포머는 인간 기준값을 뛰어넘었다. 시간이 지나면 SuperGLUE에서도 뛰어넘을 것이다.

참. NLU가 발전함에 따라 기준값도 점점 더 높게 설정될 것이다.

10. 트랜스포머를 응용한 다양한 모델이 RNN과 CNN을 뛰어넘었다. 참 / 거짓

참. 하지만 미래에 어떻게 될지는 아무도 모른다!

V.6

6장, 트랜스포머를 사용한 기계 번역

01. 기계 번역은 이제 인간의 수준을 넘어섰다. 참 / 거짓

거짓. 기계 번역은 NLP에서 가장 어려운 작업 중 하나이다.

02. 기계 번역에는 대규모 데이터셋이 필요하다. 참 / 거짓

참.

03. 동일한 데이터셋을 사용하여 트랜스포머 모델을 비교할 필요가 없다. 참 / 거짓
거짓. 모델을 비교하는 유일한 방법은 동일한 데이터셋을 사용하는 것이다.
04. BLEU는 파란색을 뜻하는 프랑스어이며 NLP 성능(metric)의 약어이다. 참 / 거짓
참. BLEU가 Bilingual Evaluation Understudy Score의 약자임을 생각하면 쉽게 맞출 수 있다.
05. 스무딩 기법은 BERT의 성능을 향상한다. 참 / 거짓
참.
06. 영어-독일어 기계 번역은 독일어-영어와 동일하다. 참 / 거짓
거짓. 독일어를 다른 언어로 번역하는 과정은 영어를 번역하는 과정과 다르다. 언어의 구조가 다르기 때문이다.
07. 오리지널 트랜스포머의 다중 헤드 어텐션 서브 층의 헤드는 2개다. 참 / 거짓
거짓. 각 어텐션 서브 층의 헤드는 8개다.
08. 오리지널 트랜스포머 인코더는 층이 6개다. 참 / 거짓
참.
09. 오리지널 트랜스포머 인코더는 층이 6개가 있지만 디코더에는 2개뿐이다. 참 / 거짓
거짓. 디코더 또한 6개 층이다.
10. 디코더 없이 트랜스포머를 학습할 수 있다. 참 / 거짓
참. BERT는 인코더로만 이루어져있다.

01. 제로샷 방식은 파라미터를 한 번만 학습한다. 참 / 거짓
거짓. 파라미터가 학습되지 않는다.
02. 제로샷 모델을 실행할 때 기울기(gradient) 업데이트가 수행된다. 참 / 거짓
거짓.
03. GPT 모델은 디코더 스택만 갖는다. 참 / 거짓
참.
04. 117M GPT 모델을 로컬 머신에서 학습은 불가능하다. 참 / 거짓
거짓. 이번 7장에서 학습하였다.
05. GPT-2 모델을 특정 데이터셋에 학습하는 것은 불가능하다. 참 / 거짓
거짓. 이번 7장에서 학습하였다.
06. GPT-2 모델은 텍스트 생성 시 조건문 사용이 불가능하다. 참 / 거짓
거짓. 이번 7장에서 구현하였다.
07. GPT-2 모델은 입력 컨텍스트를 분석해 콘텐츠 완성 결과를 반환할 수 있다. 참 / 거짓
참.
08. GPU 8개 미만인 머신에서 345M 파라미터 GPT 모델과 상호 작용할 수 없다. 참 / 거짓
거짓. 이번 7장에서 이 크기의 모델과 상호 작용하였다.

09. 285,000 CPU를 가진 슈퍼컴퓨터는 존재하지 않는다.

참 / 거짓

거짓.

10. 수천 개 GPU를 가진 슈퍼컴퓨터는 AI의 판도를 바꾸고 있다.

참 / 거짓

참. 덕분에 점점 더 큰 크기의 모델을 학습하고있다.

V.8

8장, 법률 및 금융 문서에 트랜스포머를 적용하여 요약하기

01. T5 모델은 BERT 모델과 같이 인코더 스택만 가지고 있다.

참 / 거짓

거짓.

02. T5 모델은 인코더 스택과 디코더 스택을 모두 가지고 있다.

참 / 거짓

참.

03. T5 모델은 절대적인 위치 인코딩이 아닌 상대적인 위치 인코딩을 사용한다.

참 / 거짓

참.

04. 텍스트-투-텍스트 모델은 요약을 위해서만 디자인되었다.

참 / 거짓

거짓.

05. 텍스트-투-텍스트 모델은 입력 시퀀스에 NLP 작업을 결정하는 접두사를 적용한다.

참 / 거짓

참.

06. T5 모델은 각 작업마다 특정 하이퍼파라미터를 필요로 한다.

참 / 거짓

거짓.

07. 텍스트-투-텍스트 모델의 장점 중 하나는 모든 NLP 작업에 대해 동일한 하이퍼파라미터를 사용한다는 것이다. 참 / 거짓

참.

08. T5 트랜스포머는 순방향 네트워크를 가지고 있지 않다. 참 / 거짓

거짓.

09. 허깅페이스는 트랜스포머를 보다 쉽게 구현할 수 있도록 하는 프레임워크이다. 참 / 거짓

참.

10. OpenAI의 트랜스포머 엔진은 게임 체인저이다. 참 / 거짓

참. OpenAI는 범용 모델인 Davinci와 코드를 생성해주는 코덱스(Codex) 등 바로 사용할 수 있는 다양한 모델을 만들었다.

V.9

9장, 데이터셋에 적합한 토큰나이저

01. 토큰화 사전은 언어에 존재하는 모든 단어가 들어있다. 참 / 거짓

거짓.

02. 학습된 토큰나이저는 어떤 데이터셋도 인코딩할 수 있다. 참 / 거짓

거짓.

03. 데이터셋을 사용하기 전에 미리 검토하는 것이 좋다. 참 / 거짓

참.

04. 데이터셋에서 외설적인 데이터를 제거하는 것이 좋다. 참 / 거짓
참.
05. 차별적인 주장을 하는 데이터를 제거하는 것이 좋다. 참 / 거짓
참.
06. 원시 데이터셋을 그대로 사용하면 노이즈 데이터와 유용한 데이터 사이에 연관성이 생길 수 있다. 참 / 거짓
참.
07. 일반적인 사전 학습된 토큰라이저에는 700년 전 어휘가 들어있다. 참 / 거짓
거짓.
08. 현대 영어를 학습한 토큰라이저로 오래된 영어를 인코딩하면 문제가 발생할 수 있다. 참 / 거짓
참.
09. 현대 영어를 학습한 토큰라이저로 의학 등 전문 분야 데이터를 인코딩하면 문제가 발생할 수 있다. 참 / 거짓
참.
10. 학습된 토큰라이저의 인코딩 결과를 직접 확인하는 것이 좋다. 참 / 거짓
참.

01. 의미역 결정(SRL)은 텍스트 생성 작업이다.

참 / 거짓

거짓.

02. 서술어는 명사이다.

참 / 거짓

거짓.

03. 동사는 서술어이다.

참 / 거짓

참.

04. 논항은 누가, 무엇을 하고 있는지를 설명할 수 있다.

참 / 거짓

참.

05. 수식어는 부사가 될 수 있다.

참 / 거짓

참.

06. 수식어는 위치가 될 수 있다.

참 / 거짓

참.

07. BERT 기반 모델은 인코더 및 디코더 스택으로 이루어져있다.

참 / 거짓

거짓.

08. BERT 기반 SRL 모델에는 표준 입력 형식이 있다.

참 / 거짓

참.

09. 트랜스포머는 모든 SRL 작업을 해결할 수 있다.

참 / 거짓

거짓.

01. 학습된 트랜스포머 모델은 어떤 질문에도 답할 수 있다. 참 / 거짓
거짓.
02. 질문-답변은 더 이상 연구가 필요하지 않다. 참 / 거짓
거짓.
03. NER은 의미 있는 질문을 찾을 때 유용한 정보를 제공할 수 있다. 참 / 거짓
참.
04. SRL은 질문을 생성할 때 필요 없다. 참 / 거짓
거짓.
05. 질문 생성기는 질문을 생성하는 훌륭한 방법이다. 참 / 거짓
참.
06. 질문-답변에는 신중한 프로젝트 관리가 필요하다. 참 / 거짓
참.
07. ELECTRA 모델은 GPT-2와 동일한 아키텍처를 가지고 있다. 참 / 거짓
거짓.
08. ELECTRA 모델은 BERT와 동일한 아키텍처를 가지고 있지만 판별자로 학습했다. 참 / 거짓
참.
09. NER은 위치를 인식하여 I-LOC로 라벨을 지정할 수 있다. 참 / 거짓
참.

10. NER은 사람을 인식하여 I-PER로 라벨을 지정할 수 있다.

참 / 거짓

참.

V.12

12장, 고객 감정을 감지해 예측하기

01. 감성 분석을 위해 트랜스포머를 사전 학습할 필요는 없다.

참 / 거짓

거짓.

02. 문장은 항상 긍정적이거나 부정적이다. 중립적일 수는 없다.

참 / 거짓

거짓.

03. 합성성의 원리는 트랜스포머가 문장의 모든 부분을 이해해야 한다는 것을 의미한다.

참 / 거짓

참.

04. RoBERTa-large는 트랜스포머의 사전 학습 과정을 개선하기 위해 설계됐다.

참 / 거짓

참.

05. 트랜스포머는 고객이 만족하는지 여부를 알려줄 수 있다.

참 / 거짓

참.

06. 제품이나 서비스의 감성 분석 결과가 지속적으로 부정적이라면, 분석 결과는 제품이나 서비스를 개선하기 위한 적절한 결정을 내리는 데 도움이 된다.

참 / 거짓

참.

07. 모델 결과가 좋지 않다면, 모델을 바꾸기 전에 더 학습해야 한다.

참 / 거짓

참.

01. 가짜 뉴스로 라벨링된 뉴스는 항상 가짜이다. 참 / 거짓
거짓.
02. 모두가 동의하는 뉴스는 항상 정확하다. 참 / 거짓
거짓.
03. 트랜스포머를 트윗 감성 분석에 사용할 수 있다. 참 / 거짓
참.
04. 페이스북 메시지에서 DistilBERT 모델을 사용해 주요 개체명을 추출할 수 있다. 참 / 거짓
참.
05. 유튜브 채팅에서 주요 동사는 BERT 기반 모델을 사용하여 의미역 결정을 수행함으로써 식별할 수 있다. 참 / 거짓
참.
06. 감정적 반응은 가짜 뉴스에 대한 자연스러운 첫 번째 반응이다. 참 / 거짓
참.
07. 가짜 뉴스에 대한 합리적인 접근은 자신의 입장을 명확히 하는 데 도움이 된다. 참 / 거짓
참.
08. 신뢰할 수 있는 웹사이트와 트랜스포머를 연결하면 어떤 뉴스가 가짜인지 이해하는 데 도움이 된다. 참 / 거짓
참.

09. 트랜스포머를 사용해 신뢰할만한 웹사이트들을 요약해 가짜 뉴스라고 알려진 주제들을 이해할 수 있다.

참 / 거짓

참.

10. 모두의 이익을 위해 인공지능을 사용한다면 세상을 바꿀 수 있다.

참 / 거짓

참.

V.14

14장, 블랙박스 트랜스포머 모델 해석하기

01. BertViz는 BERT의 마지막 층만 보여준다.

참 / 거짓

거짓. BertViz는 모든 층의 결과를 보여준다.

02. BertViz로 BERT의 각 층에 있는 어텐션 헤드를 시각화할 수 있다.

참 / 거짓

참.

03. BertViz로 토큰 간의 관계를 시각화할 수 있다.

참 / 거짓

참.

04. LIT는 BertViz처럼 어텐션 헤드 내부의 모습을 시각화한다.

참 / 거짓

거짓. LIT는 논-프로빙(non-probing) 방식이다.

05. 프로빙(probing)은 모델이 내부적으로 언어를 어떻게 표현하는지 분석하는 방법이다.

참 / 거짓

참.

06. NER은 프로빙 방식의 작업이다.

참 / 거짓

참.

07. PCA와 UMAP은 논-프로빙(non-probing) 방식의 작업이다. 참 / 거짓

참.

08. LIME은 모델에 독립적(agnostic)이다. 참 / 거짓

참.

09. 트랜스포머의 층을 지날수록 토큰간 관계가 심화된다. 참 / 거짓

참.

10. 트랜스포머의 시각화는 AI를 설명하는 새로운 축을 제공했다. 참 / 거짓

참.

V.15 15장, NLP부터 범용 트랜스포머 모델까지

01. 리포머 트랜스포머 모델에는 인코더가 포함되지 않는다. 참 / 거짓

거짓. 리포머는 인코더를 포함한다.

02. 리포머 트랜스포머 모델에는 디코더가 포함되지 않는다. 참 / 거짓

거짓. 리포머는 인코더와 디코더를 포함한다.

03. 리포머 모델에서 입력은 층별로 저장된다. 참 / 거짓

거짓. 리포머는 정보를 저장하고 있는 대신 각 층의 입력을 재계산하여 메모리는 절약한다.

04. DeBERTa 트랜스포머 모델은 콘텐츠와 위치 정보에 풀기(disentangle)를 적용했다. 참 / 거짓

참.

05. 프로젝트에 사용할 트랜스포머 모델을 선택하기 전에 사전 학습된 트랜스포머 모델 수백 개를 테스트해야 한다. 참 / 거짓

참 또는 거짓. 모든 모델을 테스트할 수도, 하나를 선택하여 필요에 맞게 수정할 수도 있다.

06. 최신 트랜스포머 모델이 항상 가장 좋다. 참 / 거짓

참 또는 거짓. 트랜스포머에 대한 많은 연구가 진행되고 있지만, 실험적인 모델들은 수명이 짧다. 하지만 일부는 이전보다 확실히 좋은 성능을 보인다.

07. 멀티태스크 트랜스포머 모델 한 개 보다 NLP 작업마다 하나의 트랜스포머 모델을 사용하는 것이 좋다. 참 / 거짓

참 또는 거짓. 프로젝트의 목표와 비용, 리스크 관리에 따른 선택의 문제이다.

08. 트랜스포머 모델은 항상 미세 조정해야 한다. 참 / 거짓

거짓. GPT-3는 제로-샷(zero-shot) 모델이다

09. OpenAI GPT-3 엔진은 미세 조정 없이도 광범위한 NLP 작업을 수행할 수 있다. 참 / 거짓

참.

10. AI 알고리즘은 항상 로컬 서버에서 구현하는 것이 좋다. 참 / 거짓

거짓. 프로젝트에 따라 다르다. 리스크 관리 차원에서 선택해야 한다.

V.16 16장, 트랜스포머 기반 코파일럿의 등장

01. 코드를 자동으로 생성하는 AI 코파일럿은 존재하지 않는다. 참 / 거짓

거짓. 깃허브 코파일럿이 이미 운영 중이다.

02. AI 코파일럿은 결코 인간을 대체하지 못할 것이다. 참 / 거짓
- 참 혹은 거짓. AI가 영업, 지원, 유지보수 등 많은 분야를 대체할 것이다. 하지만 여러 복잡한 분야는 계속 인간의 개입이 필요할 것이다.
03. GPT-3 엔진은 한 가지 작업만 수행할 수 있다. 참 / 거짓
- 거짓. GPT-3는 다양한 작업을 수행할 수 있다.
04. 트랜스포머를 학습해 추천 엔진으로 사용할 수 있다. 참 / 거짓
- 참. 트랜스포머는 언어 시퀀스 이외의 도메인에도 적용된다.
05. 트랜스포머는 언어만 처리할 수 있다. 참 / 거짓
- 거짓. 트랜스포머는 다양한 유형의 시퀀스를 분석할 수 있다.
06. 트랜스포머에는 오직 단어시퀀스만 사용할 수 있다. 참 / 거짓
- 거짓. 트랜스포머 내부는 숫자로 동작한다. 숫자로 표현되는 시퀀스는 모두 사용 가능하다.
07. 비전 트랜스포머는 CNN과 필적할 수 없다. 참 / 거짓
- 거짓. 트랜스포머는 컴퓨터 비전 분야에서 CNN에 필적한다.
08. 컴퓨터 비전을 갖춘 AI 로봇은 존재하지 않는다. 참 / 거짓
- 거짓. 군사 분야에서 컴퓨터 비전을 갖춘 로봇이 등장하기 시작했다.
09. 파이썬 소스 코드를 자동으로 생성하는 것은 불가능하다. 참 / 거짓
- 거짓. 마이크로소프트와 OpenAI가 파이썬 코드를 생성하는 코파일럿을 만들었다.
10. 사람이 로봇의 코파일럿이 되는 날이 올 수 있다. 참 / 거짓
- 참 또는 거짓. 발전하는 AI 생태계 속 인간과 로봇의 도전과제이다.

01. GPT-4는 자의식이 있다.

참 / 거짓

거짓. GPT-4는 수학적 알고리즘이다. 통계적 패턴을 배우고 다양한 작업을 하기 위해 자각능력을 사용하지는 않는다.

02. ChatGPT는 전문가를 대체할 수 있다.

참 / 거짓

거짓. ChatGPT는 데이터에 기반한 결과를 만들지만, 전문가 수준의 결정을 내리지는 못한다.

03. GPT-4는 모든 문제에 대한 소스 코드를 만들 수 있다.

참 / 거짓

거짓. GPT-4는 다양한 소스 코드를 만들 수 있다. 하지만 복잡한 문제에서는 인간의 개입이 필요하다.

04. 고급 프롬프트 엔지니어링은 사용하기 매우 쉽다.

참 / 거짓

거짓. 고급 프롬프트 엔지니어링은 트랜스포머에 대한 심도 있는 지식을 기반으로 한다. 고급 프롬프트 엔지니어링은 지식 기반 구축 및 API와 다양한 모델에 대한 이해를 바탕으로 한다.

05. GPT-4는 가장 진보된 트랜스포머이며 항상 최고의 선택이다.

참 / 거짓

거짓. 차를 고르는 것과 같다. 가장 비싸고 빠른 차가 가장 필요한 차는 아니다. 프로젝트에 따라, 가장 강력한 트랜스포머가 아니어도 충분하다.

06. GPT-4를 코파일럿으로 사용하면 별다른 학습 없이도 애플리케이션을 만들 수 있다.

참 / 거짓

거짓. 코파일럿은 도움을 줄 뿐이다. 트랜스포머가 프로그램을 개발할 수 있지만 복잡한 프로그램은 사람의 개입이 필요하다.

07. AI는 한계에 도달했으므로 GPT-4는 OpenAI의 마지막 트랜스포머 모델이 될 것이다.

참 / 거짓

거짓. OpenAI와 경쟁사들은 더 나은 모델을 만들 것이다.

찾아보기

한글(ㄱ~ㅎ)

4차 산업	20, 390
가짜 뉴스	404
간단한 기술기 시각화	408
감성 분석	388, 415
강화학습	522
개체명 인식	418
구글 번역	194
규칙 시스템	27
기계 번역	178
논항	307
다음 문장 예측하기	87, 89
다중 장르 자연어 추론	384
단일화	21, 468
데이터 플레이어	139
데이터포인트 편집기	458
디코더 스택	73
라벨 스무딩	76, 192
리포머	463
마스크드 언어 모델링	87, 88
마이크로소프트	
패러프레이즈 말뭉치	170
매튜 상관 계수	93, 111, 155
멀티-헤드 어텐션	53, 587
모델 시각화	439
모델 해석	407
모범 사례	274
미세 조정	23, 34, 212, 233
바이트 단위 BPE	284
방법론	338
벨류 행렬	55
부드러운 기술기 시각화	408
비전 트랜스포머	469
사람 개체 질문	349
상호 참조 해결	31, 362
서브 층	583
서비스형 소프트웨어	452
서술어 역할 결정	331

설명 가능한 AI	566
순방향 네트워크	72
순환 신경망	26
스케일드 내적 어텐션	55, 586
스킵 그래프	45
스택	593
스탠퍼드 감성 트리뱅크	169
시각화 인터페이스	448
앨런 인공지능 연구소	18, 406
어텐션 헤드	584
언어 적합성 말뭉치	92, 167
원샷	212
위노그라드 스키마 챌린지	166
위치 개체 질문	345
위치 벡터	51
위치 인코딩	43, 48, 52, 88
의미론	331
의미역 결정	307, 420
이머전스	21, 468
인지부조화	404
일반 언어 이해 평가	95, 156
임베딩 서브 층	44, 45
잔차 연결	43
장기 의존성 문제	28
접두사	248
제로샷	212
지시 시리즈	516
질문-답변 트랜스포머	341
첸체리 스무딩	193
추천 시스템	520
층 정규화	43, 71
커먼 크롤	274
커밋먼트 뱅크	162
컴퓨터 비전	528
코사인 유사도	47, 280
코파일럿	505
쿼리 행렬	55

키 행렬	55
텍스트-투-텍스트	247
토큰나이저	45
통합 기술기 시각화	408
트랙스	196
트랜스덕션	28
트랜스포머	20
트랜스포머 팩터	445
파운데이션 모델	20, 21
포스트-딤러닝 모델	18
포스트-층 정규화	53
퓨샷	212
프로-에이전트	314
프로젝트 관리	348
프롬프트 엔지니어링	22, 34, 494
함의	384
합성성의 원리	372
허깅페이스	18, 33, 77, 252, 381, 392
헤드 시각화	436
헤이스택	363
휴리스틱	346

영어(A~Z)

AllenNLP	342, 422
attention head	26
AutoML(Automated Machine Learning)	149
AWS(Amazon Web Services)	18
bert-base-uncased	436
BERT(Bidirectional Encoder Representations from)	20
BERT 기반 다국어 모델	385
BertViz	435
BLEU(Bilingual Evaluation Understudy)	76, 189
BoolQ(Boolean question)	161

BPE(Byte-Pair Encoding)	45	GPU	586	raw data	18
ByteLevelBPETokenizer	124	gTTS	566	ReCoRD(Reading	
CB(Commitment Bank)	162	homogenization	468	Comprehension with	
ChatGPT	538	Hugging Face	18	Commonsense Dataset)	164
ChatGPT Plus	547	Instruction series	516	recurrent funtions	26
CNN(Convolutional Neural		Integrated Gradient		Reinforcement Learning	522
Network)	28	Visualization	406	RNN(Recurrent Neural	
Codex	18	KantaiBERT	118	Networks)	26
CoLA(The Corpus of		layer normalization	43	RoBERTa	363, 406
Linguistic Acceptability)	92, 167	LIME	446	RoBERTa-large	375, 407
content filter	518	LIT	441	RoBERTa-large-mnli	384
COPA(Choice of		long-term dependency	28	RTE(Recognizing Textual	
Plausible Answers)	160	LSH	463	Entailment)	92
coreference resolution	31, 362	Markov Decision Process	27, 522	SaaS(Software as a	
cosine similarity	47	Markov Processes	27	Service)	30, 452
CRFM(Center for Research		MCC(Matthews Correlation		self-supervised learning	18
on Foundation Models)	20	Coefficient)	93	Simple Gradient Visualization	408
CUDA(Compute Unified		MDP(Markov Decision		Smooth Gradient	
Device Architecture)	95	Process)	27	Visualization	408
DALL-E	479, 541, 575	Megatron-Turing	485	SRL(Semantic Role Labeling)	
data collator	139	meta language	22		307, 350, 390, 420, 427
datapoint editor	443	MiniLM-L12-H384-uncased	383	SRL BERT	312
DeBERTa	466	MLM(Masked Language		SuperGLUE	158
decoder stack	41, 73	Modeling)	87	SWAG(Situations With	
DistilBERT	352, 378	modal modifier	316	Adversarial Generations)	92
ELECTRA	352	Model interpretations	407	SWITCH	485
Embedding2ML	508	MRPC(Microsoft Research		T2T(Tensor2Tensor)	196
encoder stack	41	Rapaphrase Corpus)	170	T5	247, 250
ERNIE	485	MultiNLI	384	TPU	589
F1score	155	MultiRC(Multi-Sentence		transduction	28
feedforward	42	Reading Comprehension)	162	UMAP	441
FFN(Feedforward Network)	40	NER	343, 418	ViT	469
fine-tuning	23, 34, 212	NLP(Natural Language		Whisper	542, 566
foundation model	18, 20	Process)	18	WMT(Workshop on	
GLUE(General Language		NLTK(Natural Language		Machine Translation)	76, 180
Understanding Evaluation)	95	Toolkit)	189	Word2Vec	278
Google Trax	18	NLU(Natural Language		WSC(Winograd schema	
GPT	207	Understanding)	79	challenge)	166
GPT-2	215, 450	NSP(Next Sentence		XAI	566
GPT-3	208, 266, 299, 364,	Prediction)	88	XLNET	485
	388, 394, 452	OOV(Out-Of-Vocabulary)	186		
GPT-3 코덱스	23	positional encoding	43		
GPT-3.5	541	post-layer normalization	53		
GPT-4	538, 552	Predicate Role Labeling	331		