CHAPTER 1

Hello Transformers

박세현
LAMDA Lab in SKKU

Transformer

- 2017년 Google에서 발표
- 기계 번역에서 순환 신경망(RNN)보다 뛰어난 성능
  - 높은 번역 품질
  - 저렴한 학습 비용
- ULMFiT 기법
  - 크고 다양한 말뭉치에 대해 LSTM 네트워크를 훈련 시 적은 레이블의 데이터로 텍스트 분류기 생성 가능

Transformer

- 대표 모델
  - GPT(Generative Pretrained Transformer)
  - BERT(Bidirectional Encoder Representations from Transformers)
- 비지도 학습과의 결합
  - 작업별 아키텍처를 새롭게 학습할 필요가 없음
  - 자연어 처(Natural Language Processing, NLP)의 기존 성능을 뛰어넘음

Transformer

- GPT, BERT 출시 이후 모델

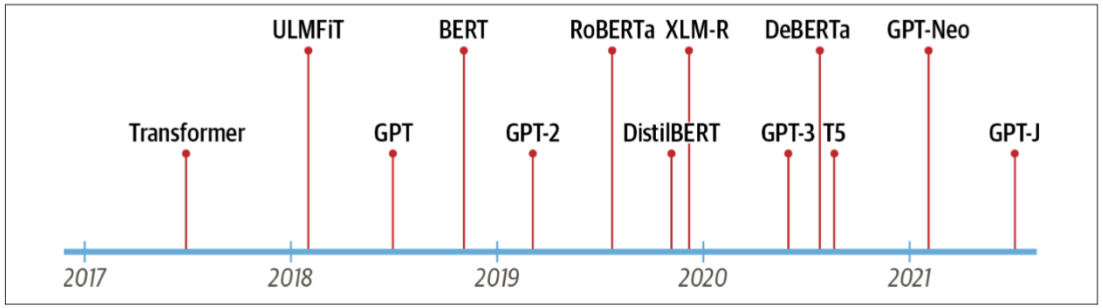


Figure 1-1. The transformers timeline

- 핵심 개념
  - 인코더-디코더(Encoder-Decoder) 프레임워크
  - 어텐션 메커니즘(Attention Mechanism)
  - 전이 학습(Transfer Learning)

인코더-디코더 프레임워크

- 순환 구조(RNN)

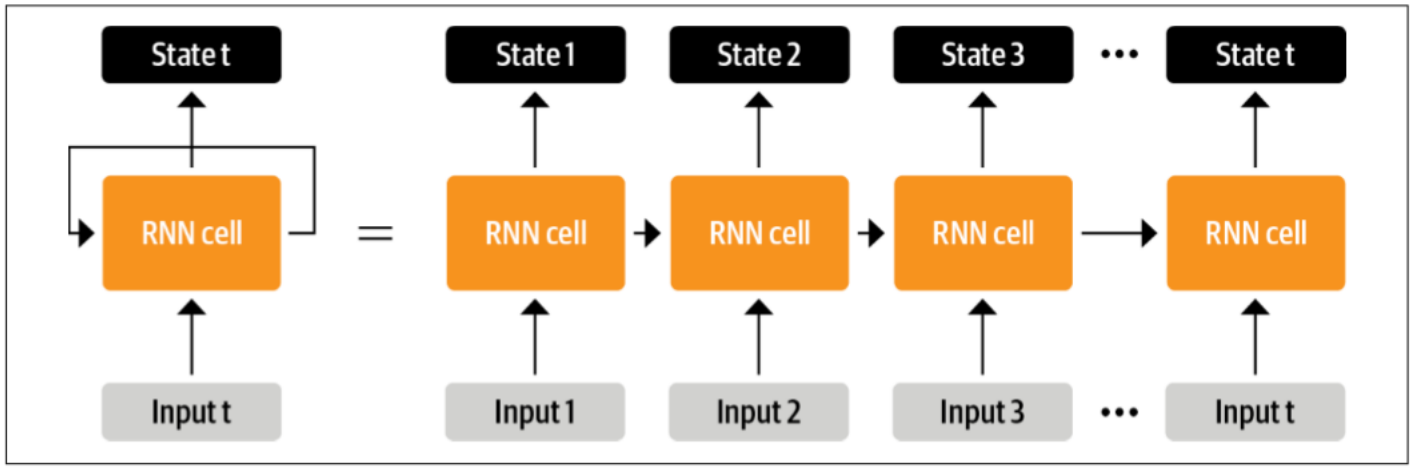
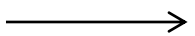


Figure 1-2. Unrolling an RNN in time

  - 입력 데이터가 네트워크를 통해 hidden state를 출력
  - 일부 정보가 루프를 통해 스스로에게 피드백
  - 이전 상태의 정보를 추적하고 출력 예측에 사용 가능

인코더-디코더 프레임워크

- 순환 구조(RNN)
  - 기계 번역 개발에서 중요한 역할
- 기계 번역
  - encoder-decoder
  - sequence-to-sequence



입력과 출력 모두 임의의 길이의 시퀀스인 상황에 적합

인코더-디코더 프레임워크

- 인코더
  - 입력 시퀀스의 정보를 마지막 hidden state의 숫자로 인코딩
- 디코더
  - 출력 시퀀스 생성
- 인코더와 디코더의 구성 요소는 시퀀스를 모델링 할 수 있는 모든 종류의 신경망 구조가 될 수 있음

인코더-디코더 프레임워크

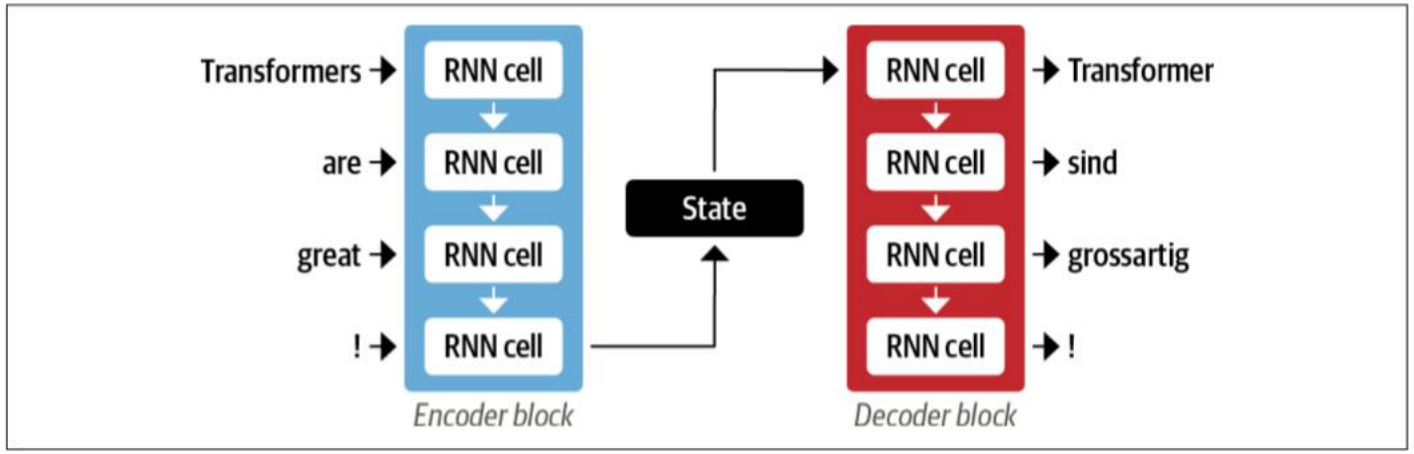


Figure 1-3. An encoder-decoder architecture with a pair of RNNs (in general, there are many more recurrent layers than those shown here)

- 입력 단어 - 인코더를 통해 순차적 공급
- 출력 단어 - 위에서 아래로 한 번에 하나씩 생성

인코더-디코더 프레임워크

- 약점
  - 인코딩의 최종 hidden state가 병목 현상을 일으킴
    -> 최종 hidden state가 전체 입력 시퀀스의 의미를 나타내야 함
  - 긴 시퀀스의 경우 시작 부분의 정보가 손실될 수 있음
- 어텐션 메커니즘으로 해결 가능
  - 모든 hidden state에 접근 가능 -> 병목 현상 해결

어텐션 메커니즘

- 어텐션
  - 인코더가 디코더가 접근할 수 있는 각 단계마다 hidden state를 출력
  - 모든 상태를 동시에 사용하면 엄청난 양의 입력 발생 -> 우선 순위
  - 디코더가 인코더 상태에 서로 다른 가중치 부여

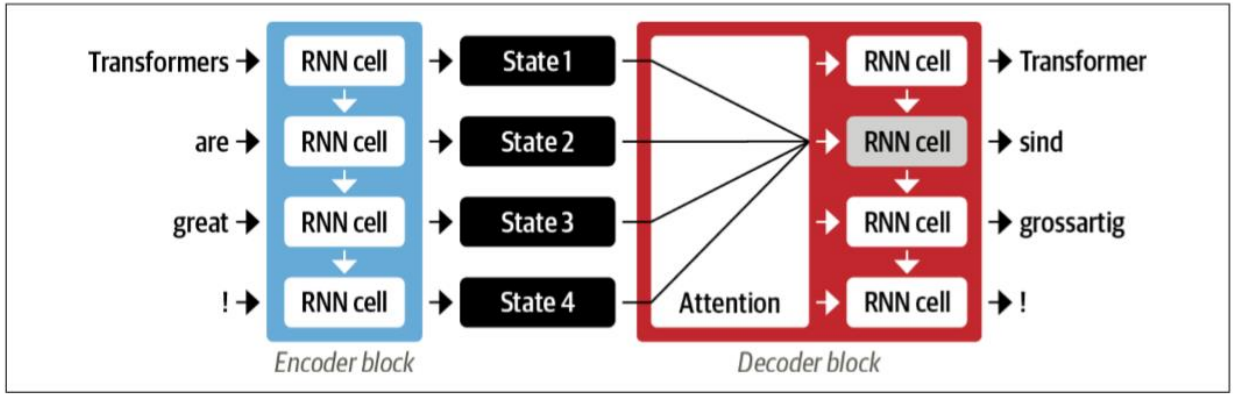


Figure 1-4. An encoder-decoder architecture with an attention mechanism for a pair of RNNs

어텐션 메커니즘

- 어텐션 기반 모델
  - 각 시간 단계에서 관련성이 높은 토큰에 집중
  - 입력 문장과 번역 문장의 단어 사이의 복잡한 정렬 학습 가능

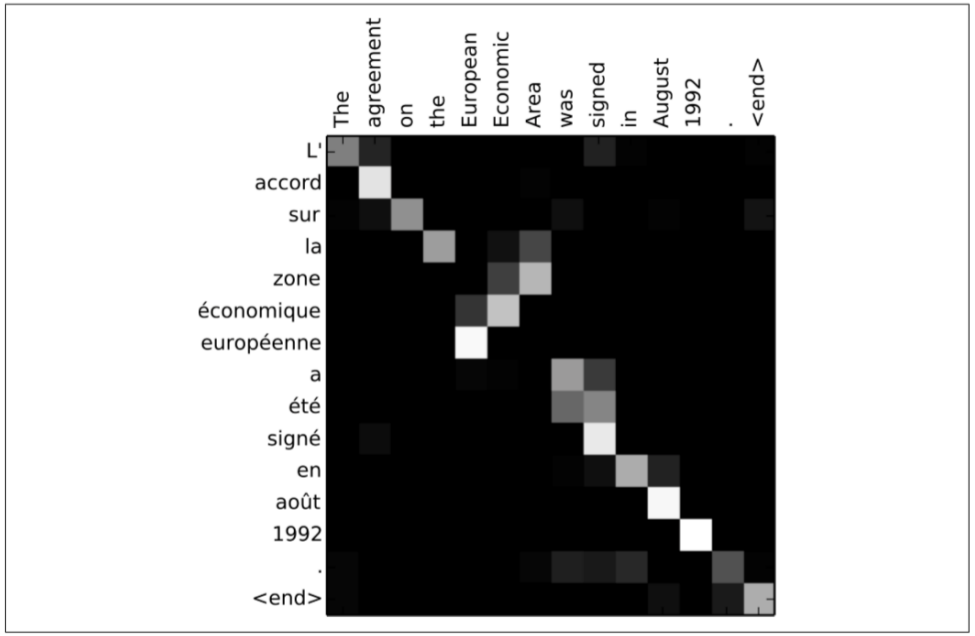


Figure 1-5. RNN encoder-decoder alignment of words in English and the generated translation in French (courtesy of Dzmitry Bahdanau)

어텐션 메커니즘

- 단점
  - 계산이 순차적 -> 입력 시퀀스 전체에서 병렬화 불가능

※ 새로운 모델 패러다임 - 셀프 어텐션(self-attention)
  - 순환 구조 배제
  - 각 출력이 피드 포워드 신경망(FF NN)에 공급
  - 순환 구조보다 훨씬 빠르게 학습 가능

어텐션 메커니즘

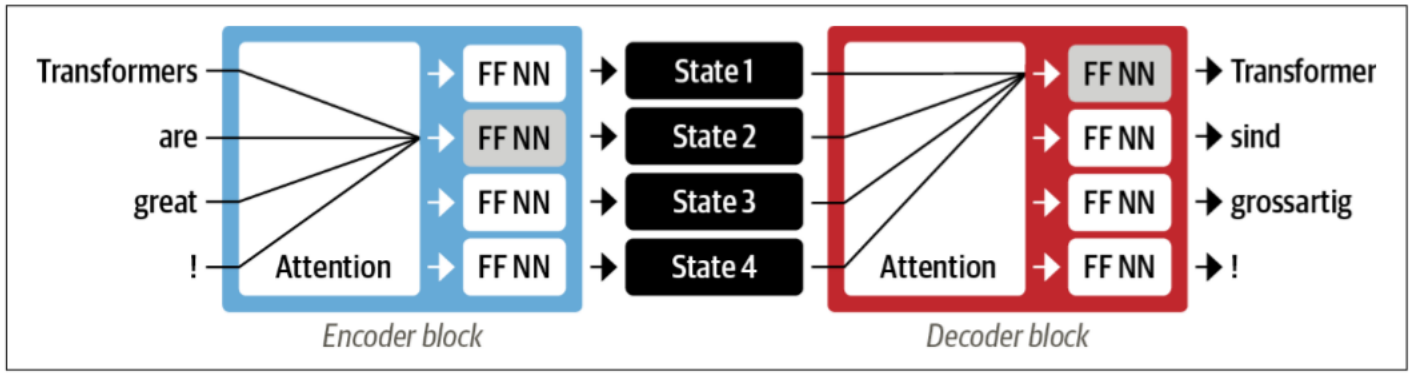


Figure 1-6. Encoder-decoder architecture of the original Transformer

- 동일한 계층에 있는 모든 상태에 어텐션이 작동
- 인코더와 디코더 모두 셀프 어텐션 메커니즘을 갖음

* 자연어 처리 응용에서 충분한 라벨링 데이터 확보가 어려움
  -> 전이 학습 필요

전이 학습

- 이전 작업에서 학습한 지식 활용 -> 미세 조정 가능
- 구조
  - 헤드: 작업별 네트워크
  - 바디
    - 바디의 가중치는 소스 도메인의 광범위한 특징 학습
    - 가중치는 새로운 작업에 대한 새 모델을 초기화하는 데 사용
- 기존의 지도 학습에 비해 훨씬 적은 레이블 데이터로 다양한 다운스트림 작업에 더 효율적

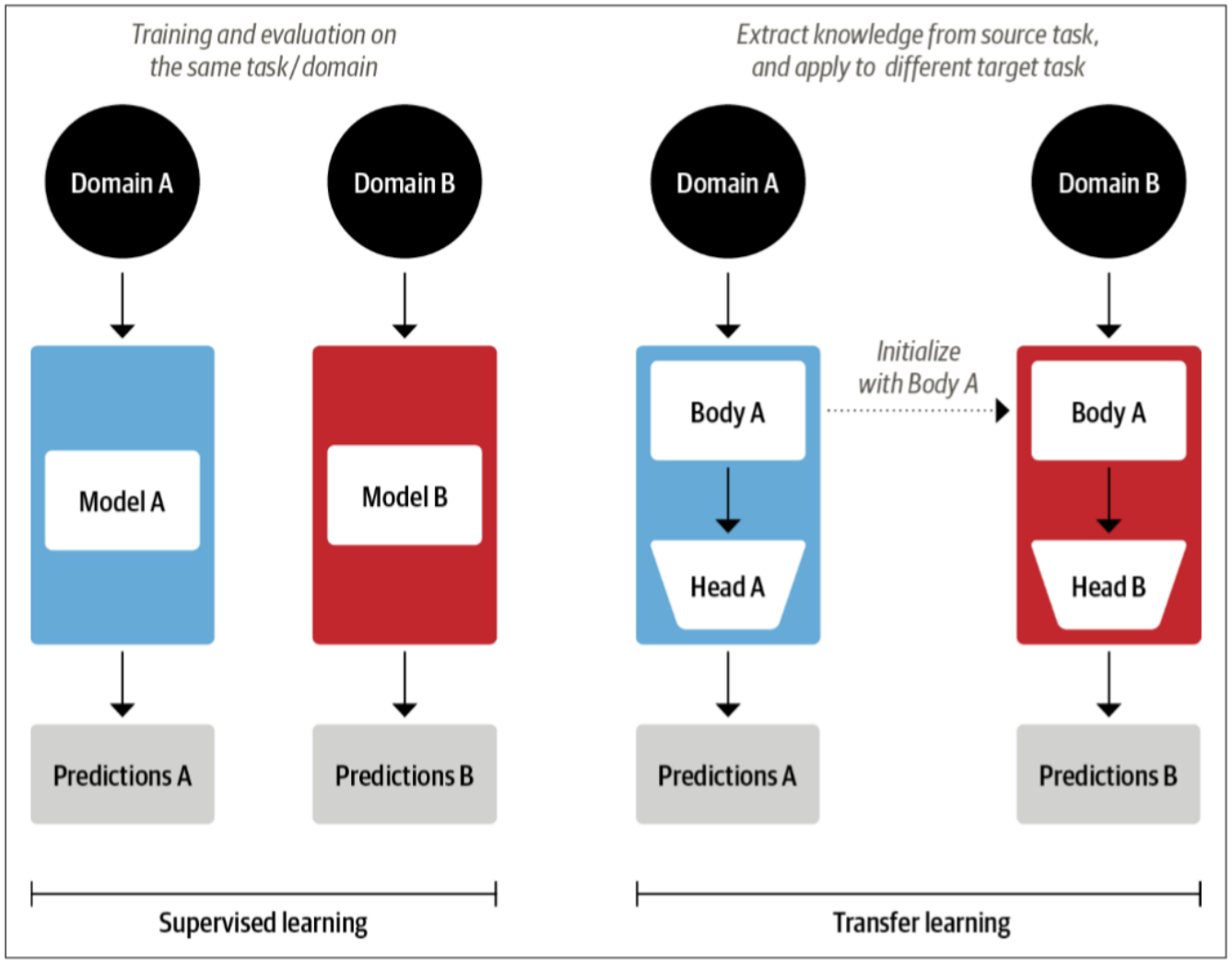


Figure 1-7. Comparison of traditional supervised learning (left) and transfer learning (right)

지도 학습 vs 전이 학습

전이 학습

- 사전 학습 모델:
  대규모 데이터 세트에 대해 모델 학습
- 사전 학습된 모델은 비교적 적은 수의 라벨링된 예제로 하위 모델에서 미세 조정 가능
- 미세 조정 모델:
  처음 학습된 지도 모델 보다 더 높은 정확도

전이 학습

- 2017년, 2018년에 전이 학습을 NLP 적용할 수 있는 접근법 제안
  - OpenAI 연구원들의 비지도 사전 학습을 통한 감정 분류 작업
- ULMFiT
  - 사전 훈련된 LSTM 모델을 다양한 작업에 적용하는 프레임워크 도입

전이 학습

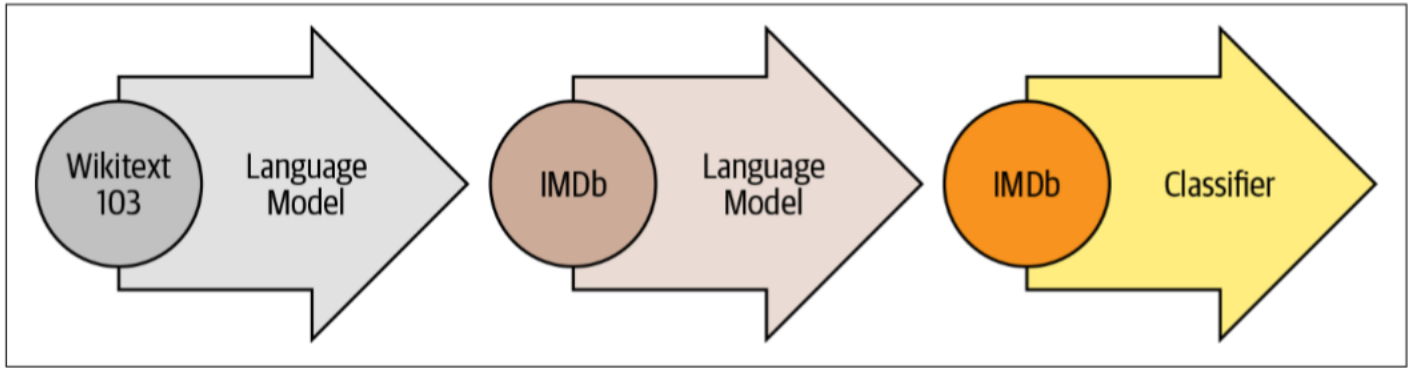


Figure 1-8. The ULMFiT process (courtesy of Jeremy Howard)

- ULMFiT
  - 사전훈련
  - 도메인 적응
  - 미세 조정

전이 학습

- 사전 훈련
  - 이전 단어를 기반으로 다음 단어 예측 - 언어 모델링
- 도매인 적용
  - 도메인 내 말뭉치에 적용
  - 언어 모델링 사용
  - 대상 말뭉치에서 다음 단어 예측
- 미세 조정
  - 대상 작업에 대한 분류 계층 사용

전이 학습

- ULMFiT
  -> Transformer가 도약할 수 있도록 부족한 부분 제공
- 2018년 셀프 어텐션과 전이 학습의 결합한 트랜스 포머 등장
  - GPT
  - BERT

전이 학습

- GPT
  - 디코더 부분만 사용
  - ULMFiT과 동일한 언어 모델링 접근 방식 사용
  - 북코퍼스에 대해 사전 학습
- BERT
  - 인코더 부분 사용
  - 마스크드 언어 모델링(Masked Language Modeling)
    - 텍스트에서 무작위로 마스크된 단어를 예측
  - 북코퍼스 및 영어 위키백과에 대해 사전 학습

전이 학습

- GPT와 BERT는 NLP 벤치마크에서 최고 성능 달성
  -> 트랜스포머 시대의 개막
- 초기에는 연구소마다 서로 다른 프레임워크로 모델 공개
  – 실무자가 자신의 애플리케이션에 적용 어려움
- Transformer 라이브러리 출시
  – 50개 이상의 트랜스포머 구조를 지원하는 통합 API 구축
  – 실무자들이 쉽게 모델을 통합 가능

Hugging Face Transformers

- 새로운 기계 학습 아키텍처를 작용하는 방법
  1. PyTorch 또는 TensorFlow 기반으로 코드 구현
  2. 서버에서 사전 학습된 가중치 로드
  3. 입력을 전처리, 모델 통과, 작업별 후처리
  4. 데이터 로더 구현하, 손실함수와 옵티마이저 정의, 모델 학습

  -> 커스텀 로직 필요

  -> 새로운 사례에 적용하는데 오랜 시간 소요

Hugging Face Transformers

- Transformer 라이브러리
  - 트랜스포머 모델에 대한 표준화된 인터페이스 제공
  - 새로운 사례에 적용할 수 있게 코드 및 도구 제공
  - PyTorch, TensorFlow, JAX 프레임워크 지원, 프레임워크 전환 가능
  - 다운스트림 작업에서 쉽게 미세 조정 가능

트랜스포머 기능

- 텍스트 분류

```
text = """Dear Amazon, last week I ordered an Optimus Prime action figure
from your online store in Germany. Unfortunately, when I opened the package,
I discovered to my horror that I had been sent an action figure of Megatron
instead! As a lifelong enemy of the Decepticons, I hope you can understand my
dilemma. To resolve the issue, I demand an exchange of Megatron for the
Optimus Prime figure I ordered. Enclosed are copies of my records concerning
this purchase. I expect to hear from you soon. Sincerely, Bumblebee."""
```

```
from transformers import pipeline

classifier = pipeline("text-classification")
```

```
import pandas as pd

outputs = classifier(text)
pd.DataFrame(outputs)
```

	label	score
0	NEGATIVE	0.901546

트랜스포머 기능

- 명명된 개체 인식

```
ner_tagger = pipeline("ner", aggregation_strategy="simple")
outputs = ner_tagger(text)
pd.DataFrame(outputs)
```

	entity_group	score	word	start	end
0	ORG	0.879010	Amazon	5	11
1	MISC	0.990859	Optimus Prime	36	49
2	LOC	0.999755	Germany	90	97
3	MISC	0.556569	Mega	208	212
4	PER	0.590256	##tron	212	216
5	ORG	0.669692	Decept	253	259
6	MISC	0.498350	##icons	259	264
7	MISC	0.775361	Megatron	350	358
8	MISC	0.987854	Optimus Prime	367	380
9	PER	0.812096	Bumblebee	502	511

* 해시 기호(#)는 토큰화 모델에 의해 생성된 것

트랜스포머 기능

- 질문 답변

```
reader = pipeline("question-answering")
question = "What does the customer want?"
outputs = reader(question=question, context=text)
pd.DataFrame([outputs])
```

	score	start	end	answer
0	0.631291	335	358	an exchange of Megatron

트랜스포머 기능

- 요약

```
summarizer = pipeline("summarization")
outputs = summarizer(text, max_length=45, clean_up_tokenization_spaces=True)
print(outputs[0]['summary_text'])
```

```
 Bumblebee ordered an Optimus Prime action figure from your online store in
Germany. Unfortunately, when I opened the package, I discovered to my horror
that I had been sent an action figure of Megatron instead.
```

트랜스포머 기능

- 번역

```
translator = pipeline("translation_en_to_de",
                      model="Helsinki-NLP/opus-mt-en-de")
outputs = translator(text, clean_up_tokenization_spaces=True, min_length=100)
print(outputs[0]['translation_text'])
```

```
Sehr geehrter Amazon, letzte Woche habe ich eine Optimus Prime Action Figur aus
Ihrem Online-Shop in Deutschland bestellt. Leider, als ich das Paket öffnete,
entdeckte ich zu meinem Entsetzen, dass ich stattdessen eine Action Figur von
Megatron geschickt worden war! Als lebenslanger Feind der Decepticons, Ich
hoffe, Sie können mein Dilemma verstehen. Um das Problem zu lösen, Ich fordere
einen Austausch von Megatron für die Optimus Prime Figur habe ich bestellt.
Anbei sind Kopien meiner Aufzeichnungen über diesen Kauf. Ich erwarte, bald von
Ihnen zu hören. Aufrichtig, Bumblebee.
```

트랜스포머 기능

- 택스트 생성

```
generator = pipeline("text-generation")
response = "Dear Bumblebee, I am sorry to hear that your order was mixed up."
prompt = text + "\n\nCustomer service response:\n" + response
outputs = generator(prompt, max_length=200)
print(outputs[0]['generated_text'])
```

```
Dear Amazon, last week I ordered an Optimus Prime action figure from your online
store in Germany. Unfortunately, when I opened the package, I discovered to my
horror that I had been sent an action figure of Megatron instead! As a lifelong
enemy of the Decepticons, I hope you can understand my dilemma. To resolve the
issue, I demand an exchange of Megatron for the Optimus Prime figure I ordered.
Enclosed are copies of my records concerning this purchase. I expect to hear
from you soon. Sincerely, Bumblebee.

Customer service response:
Dear Bumblebee, I am sorry to hear that your order was mixed up. The order was
completely mislabeled, which is very common in our online store, but I can
appreciate it because it was my understanding from this site and our customer
service of the previous day that your order was not made correct in our mind and
that we are in a process of resolving this matter. We can assure you that your
order
```

Hugging Face 에코시스템

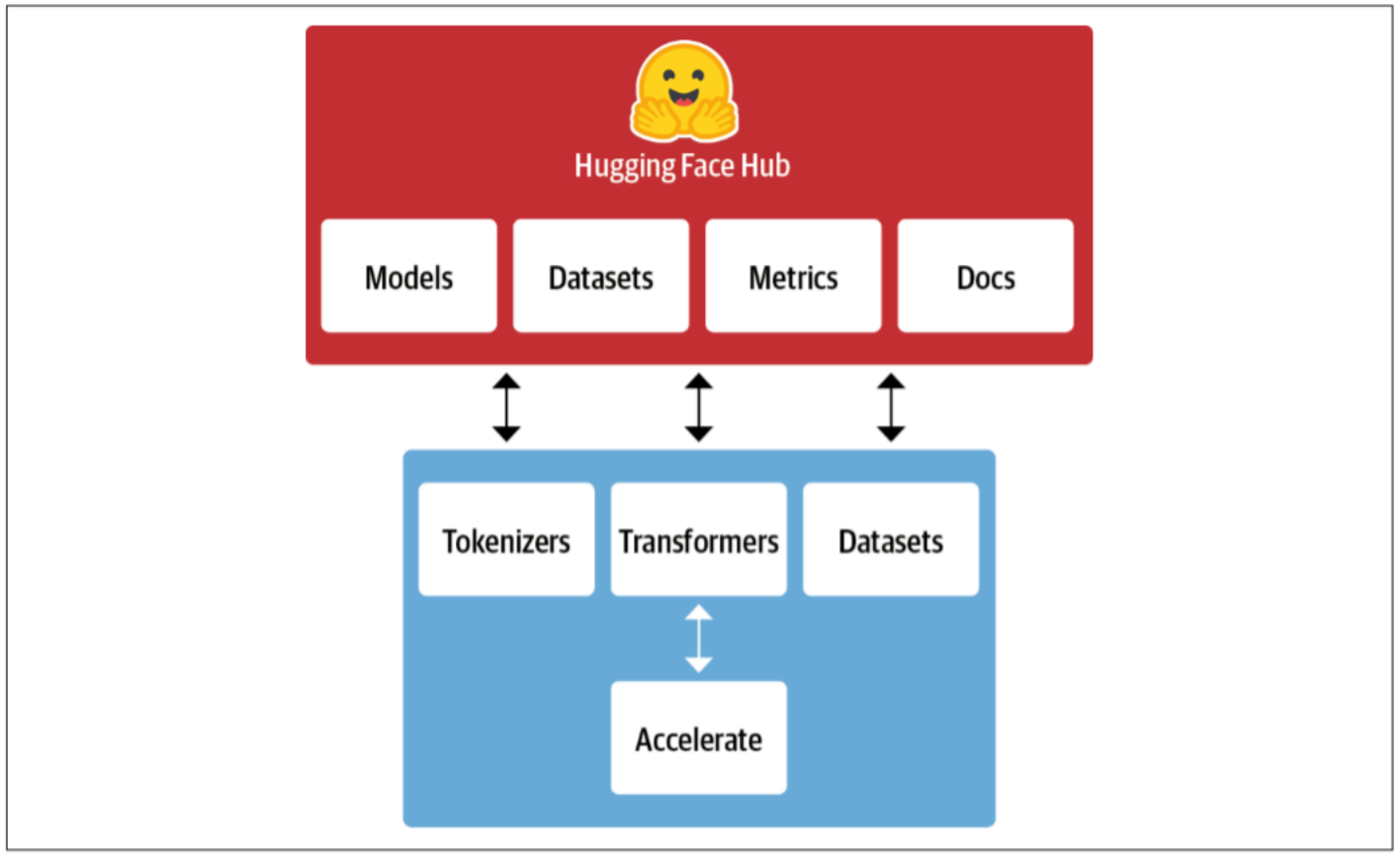


Figure 1-9. An overview of the Hugging Face ecosystem

- 라이브러리
  - 코드
- 허브
  - 사전 학습 가중치
  - 데이터 세트
  - 평가 스크립트

Hugging Face 허브

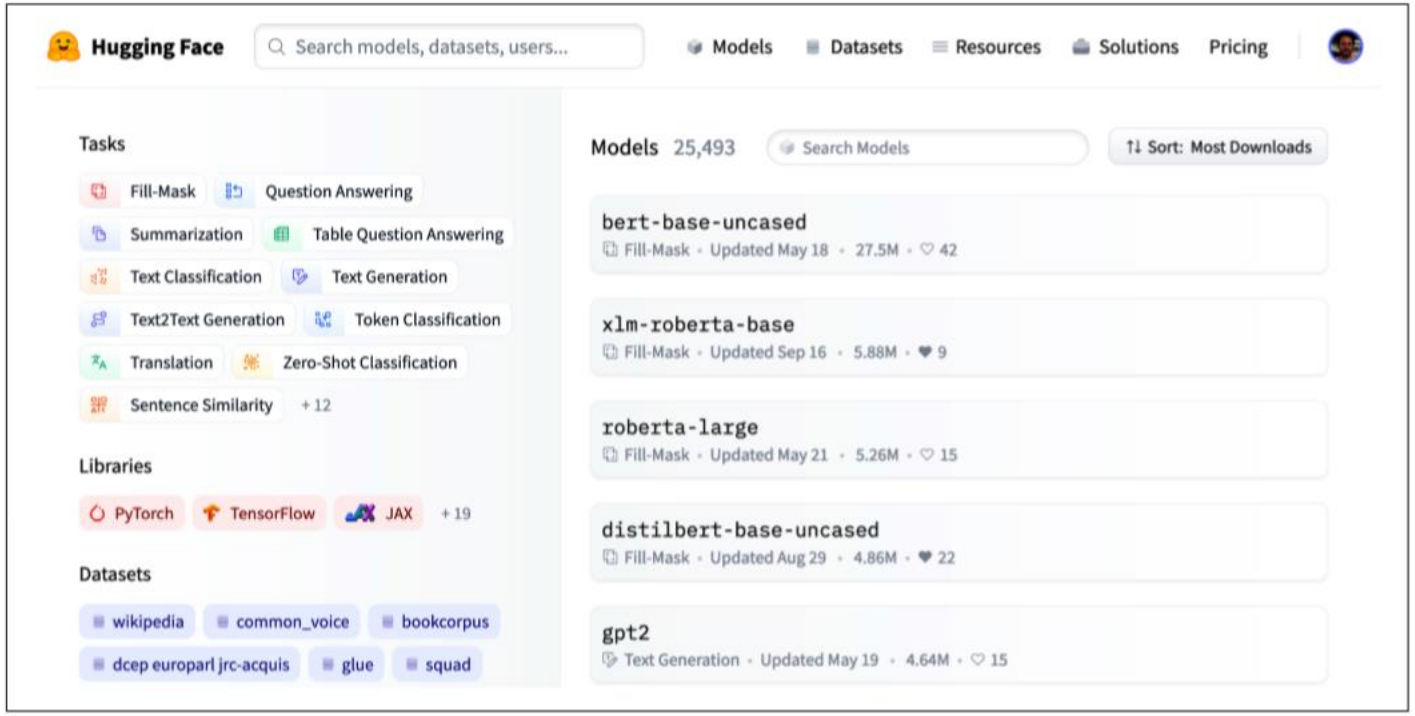


Figure 1-10. The Models page of the Hugging Face Hub, showing filters on the left and a list of models on the right

- 20,000개 이상의 주요 모델 제공

Hugging Face 허브

- 모델 가중치
- 메트릭 계산을 위한 데이터 세트
- 스크립트 호스팅 -> 결과 재현, 애플리케이션 추가 데이터 활용
- 모델 및 데이터 세트 카드 - 문서화
  -> 적합 여부 판단

Hugging Face 허브

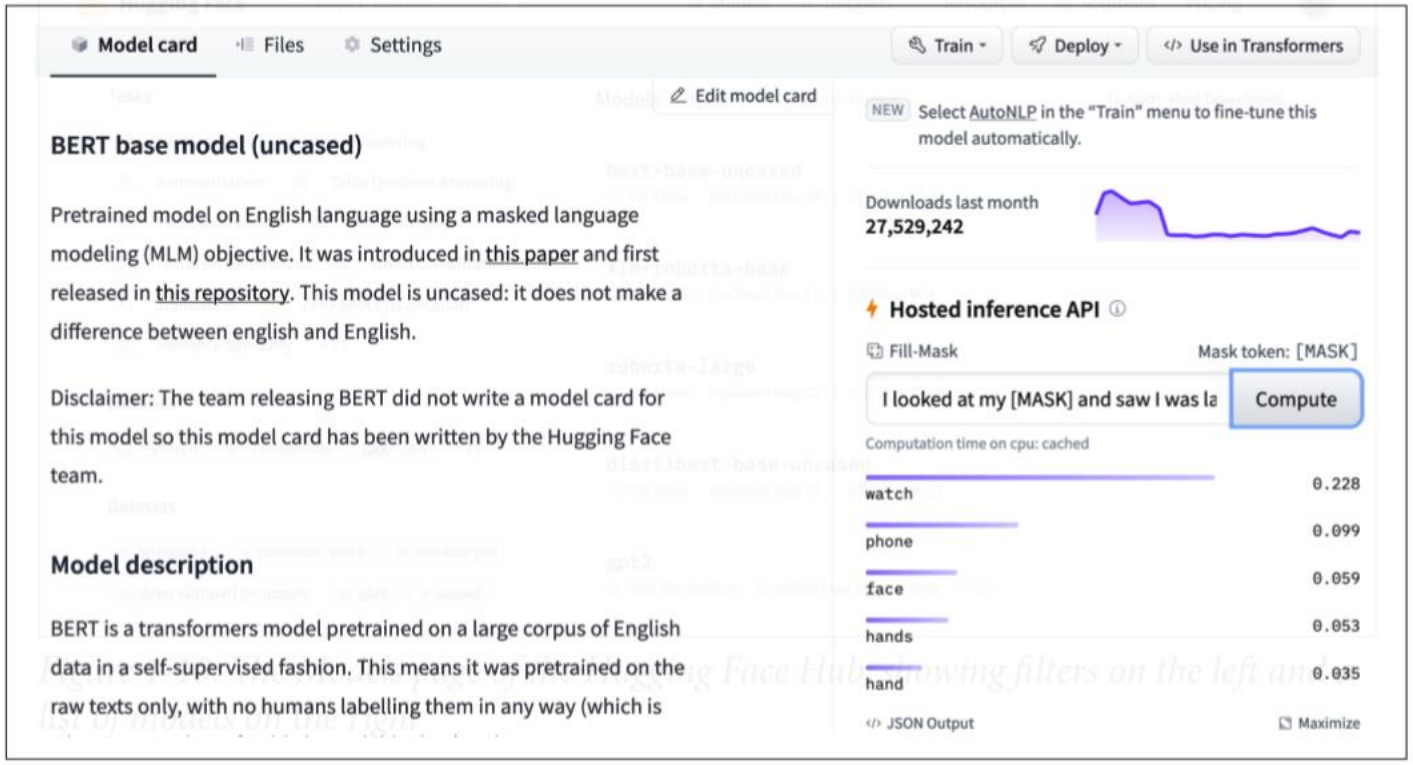


Figure 1-11. An example model card from the Hugging Face Hub: the inference widget, which allows you to interact with the model, is shown on the right

* 모델이나 데이터 세트가 허깅 페이스 허브에서 제공되지 않는 경우 PyTorch 및 TensorFlow의 자체 허브에서 확인 가능

[illegible] 업 모델 사용 가능

Hugging Face 토크나이저

- 토큰화:
  원시 텍스트를 토큰이라는 작은 조각으로 분할
- 토큰화 도구는 다양한 토큰화 전략 제공
- Rust 백엔드 덕에 빠른 토큰화 속도
- 전처리 및 후처리 단계 처리

Hugging Face 데이터셋

- 허브에서 찾을 수 있는 수천 개의 데이터 세트에 대한 표준 인터페이스 제공 -> 프로세스 간소화
- 스마트 캐싱
  - 코드를 실행할 때마다 전처리 다시할 필요 없음
- 메모리 매핑
  - 파일의 내용을 가상 메모리에 저장 -> RAM 제한 회피 가능
- Pandas, NumPy와 상호 운용가능

Hugging Face 가속

- 일반 트레이닝 루프 + 추상화 계측
  - 인프라 변경을 간소화
  - 워크플로우 가속화

Transformer의 도전과제

- 언어
  - 희귀하거나 자원이 부족한 언어
- 데이터 가용성
  - 작업에 필요한 양에 비해 많은 학습 데이터의 양
- 긴 문서 작업
  - 긴 문서에 많은 비용 소모
- 불투명도
  - 특정 예측을 한 이유를 밝히기 어려움
- 편향
  - 데이터에 존재하는 편향이 모델에 각인