HuggingFace

HuggingFace란?

- 허깅 페이스(Hugging Face)는 머신러닝 어플리케이션을 개발하는 미국 기업
  - 2016년에 설립된 허깅 페이스는 처음에는 챗봇 회사로 시작
  - AI를 연구하는 연구자, 개발자 및 회사를 위한 강력한 오픈소스 도구 모음을 구축하는 방향으로 전환
- 자연어 처리 모델을 공통 인터페이스로 이용할 수 있는 Transformers, 이미지 생성 모델을 공통 인터페이스로 이용할 수 있는 diffusers 등의 라이브러리를 제공
  - 허깅페이스 허브 (Hugging Face Hub) : 머신러닝 모델과 데이터 세트를 공유하기 위한 플랫폼 운영

HuggingFace란?

- NLP 분야의 핵심 알고리즘(ex Transformers)등의 코드 및 학습 기술에 대한 표준 제공
- Huggingface Hub를 통해 수십만개의 모델, 학습 데이터셋 등 오픈소스로 제공
  - 버트, GPT-3, T5, 그리고 로베르타 등의 모델이 모두 HuggingFace를 통해 공개됨
- AI 분야의 github과 같은 존재
- 주요 라이브러리
  - Transformers: 사전 학습된 트랜스포머 모델을 불러오고 학습 및 추론
  - Datasets: 대규모 NLP 데이터셋 제공 및 간편한 처리
  - Hugging Face Hub: 모델, 데이터셋, Space 공유 및 관리

NLP란?

- NLP(Natural Language Processing)란 사람의 언어와 관련된 모든 것을 이해하는 데에 중점을 둔 언어학 및 기계 학습(머신 러닝) 분야
  - NLP의 목적은 단순히 하나의 개별 단어를 이해하는 것을 넘어, 해당 단어들의 문맥을 이해하는 것
- NLP 작업 예시
  - **전체 문장 분류**: 리뷰에 드러난 감정 파악하기, 스팸 메일 분류하기, 문장이 문법적으로 올바른지 혹은 문장 쌍이 논리적으로 관련이 있는지 없는지 결정하기
  - **문장 내 단어 분류**: 문장 구성 성분(명사, 동사, 형용사 등) 혹은 개체명(사람, 장소, 기관) 식별하기
  - **텍스트 컨텐츠 생성**: 자동 생성 텍스트로 프롬프트 작성하기, 텍스트 내 마스킹 된 단어의 빈칸 채우기
  - **텍스트 안에서 정답 추출하기** : 지문과 질의가 주어질 때 지문에 주어진 정보를 이용해 질의에 대한 정답 추출하기

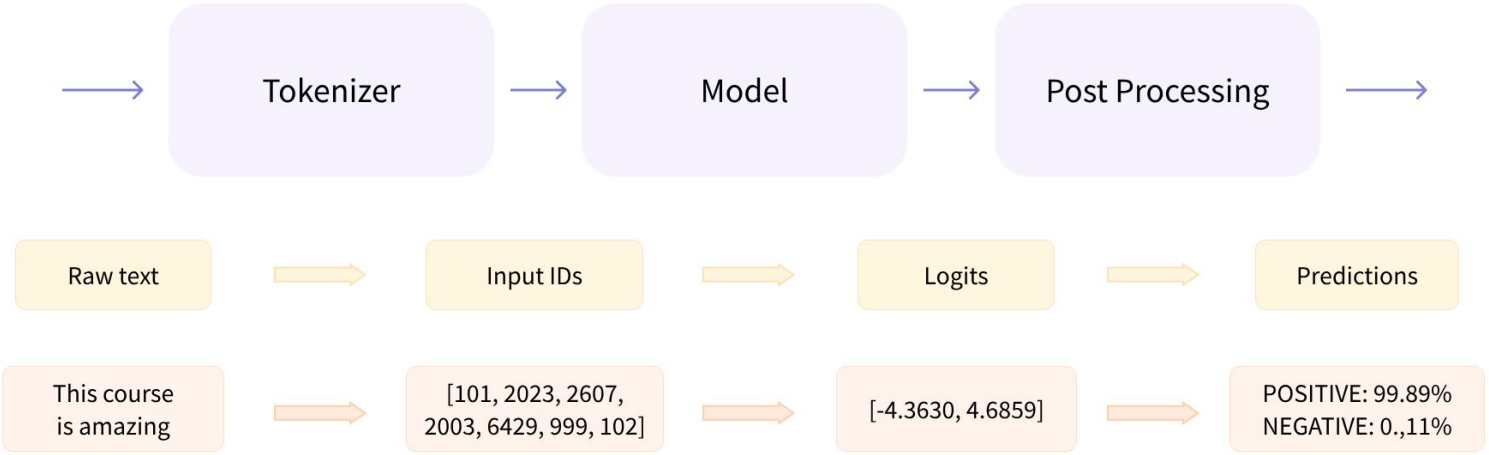
HuggingFace 실습

```
from transformers import pipeline

classifier = pipeline("sentiment-analysis")
classifier(
    [
        "I've been waiting for a HuggingFace course my whole life.",
        "I hate this so much!",
    ]
)
```

```
[{'label': 'POSITIVE', 'score': 0.9598047137260437},
 {'label': 'NEGATIVE', 'score': 0.9994558095932007}]
```

HuggingFace 실습



HuggingFace 실습

- 토크나이저(Tokenizer)를 이용한 전처리
  - Transformer 모델은 원시 텍스트를 바로 처리할 수 없음
  - 파이프라인의 첫 번째 단계는 텍스트 입력을 모델이 이해할 수 있는 숫자로 변환하는 것
- 토크나이저의 역할
  - 입력을 토큰이라고 부르는 단어나 하위 단어, 또는 심볼(예-구두점)로 분할
  - 각 토큰을 하나의 정수에 매핑
  - 모델에 유용할 수 있는 부가적인 입력 추가
- Transformer 모델은 tensor라는 숫자 배열 데이터 형을 받음(딥러닝 표준)

HuggingFace 실습

- 토크나이저 처리 예시 - 공백, 구두점
  - 단어 기반 토크나이저로 언어를 처리할 시, 언어의 각 단어에 대한 식별자가 있어야 하며, 이는 엄청난 양의 토큰을 생성함
  - 영어에 500,000개가 넘는 단어가 있다고 한다면, 각 단어에 입력 ID를 매핑시키기 위해 많은 ID를 추적해야함
  - “dog”와 같은 단어는 “dogs”처럼 다르게 표현되어 모델이 처음에 “dog”와 “dogs”가 유사하다는 것을 알 방법이 없음

Split on spaces

Let’s	do	tokenization!

Split on punctuation

Let	’s	do	tokenization	!

HuggingFace 실습

- 토크나이저 처리 예시 - character(문자) 단위
  - 단어사전이 간결해지고, 모든 단어는 문자로 이루어졌기 때문에 out-of-vocabulary (unknown) 토큰의 수가 훨씬 적음
  - 언어별로 문자 또는 단어 마다 담고 있는 정보량이 다름
  - 모델이 처리해야 하는 매우 많은 양의 토큰이 생기게 됨
    - 단어 기반 토크나이저에서 단어는 단 하나의 토큰인 반면, 문자로 변환하면 10개 이상의 토큰으로 쉽게 바뀔 수 있음

HuggingFace 실습

- 토크나이저 처리 예시 - subword 토큰화
  - 서브워드 토큰화 알고리즘은 자주 사용되는 단어는 더 작은 서브워드로 나누면 안되지만, 희귀한 단어는 의미 있는 서브워드로 나눠야 한다는 규칙에 기반
  - “tokenization”은 의미론적인 정보를 갖는 두 개의 토큰 “token”과 “ization”으로 분할되었고 긴 단어를 두 단어만으로 표현할 수 있어 공간 효율적임
    - 크기가 작은 단어 사전으로도 많은 토큰을 표현할 수 있고 unknown 토큰도 거의 없음

Let’s </w>	do</w>	token	ization</w>	!</w>

HuggingFace 실습

```
from transformers import AutoTokenizer

checkpoint = "distilbert-base-uncased-finetuned-sst-2-english"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)
```

```
raw_inputs = [
    "I've been waiting for a HuggingFace course my whole life.",
    "I hate this so much!",
]
inputs = tokenizer(raw_inputs, padding=True, truncation=True, return_tensors="pt")
print(inputs)
```

```
{
    'input_ids': tensor([
        [  101,  1045,  1005,  2310,  2042,  3403,  2005,
        [  101,  1045,  5223,  2023,  2061,  2172,   999,
    ]),
    'attention_mask': tensor([
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]
    ])
}
```

HuggingFace 실습

- 모델(Model) 불러오기
  - 해당 아키텍처는 기본 Transformer 모듈만 포함하고 있음
  - 입력이 주어지면 features라고도 불리는 hidden states를 출력
  - 각 모델의 입력에 대해 Transformer 모델에 의해 수행된 입력의 문맥적 이해로 표현할 수 있는 고차원 벡터 리턴

```
from transformers import AutoModel

checkpoint = "distilbert-base-uncased-finetuned-sst-2-english"
model = AutoModel.from_pretrained(checkpoint)
```

HuggingFace 실습

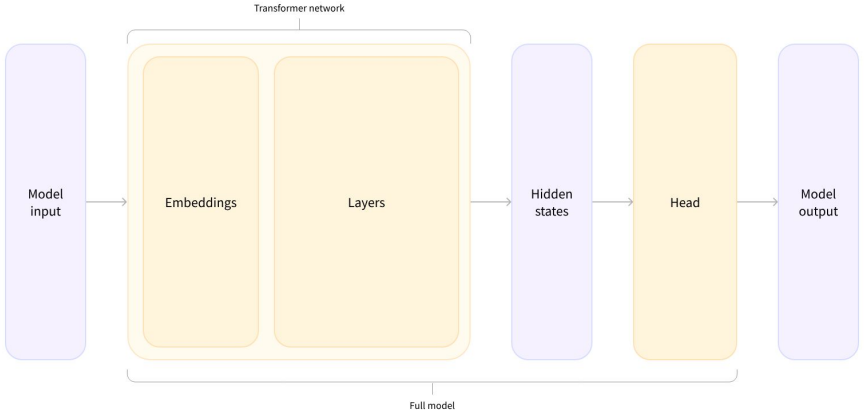
- 고차원 벡터
  - Transformer 모듈에 의한 출력 벡터는 일반적으로 크며 보통 3개의 차원을 가짐
    - Batch size: 한 번에 처리되는 시퀀스의 수 (2)
    - Sequence length: 시퀀스의 숫자 표현 길이 (16)
    - Hidden size: 각 모델 입력 벡터 차원 (768)
  - hidden size는 매우 클 수 있으며, 이로 인해 고차원이라고 불림

```
outputs = model(**inputs)
print(outputs.last_hidden_state.shape)
```

```
torch.Size([2, 16, 768])
```

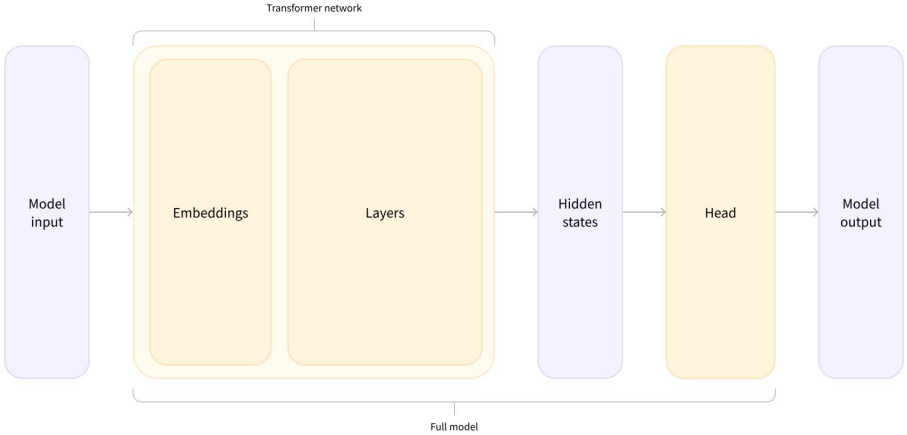
HuggingFace 실습

- 모델 헤드(Model Head)
  - 모델 헤드는 hidden state의 고차원 벡터를 입력으로 받아 다른 차원으로 투영함
  - 모델 헤드는 보통 하나 이상의 선형 레이어(MLP)로 이루어져 있음



HuggingFace 실습

- 임베딩: 토큰화된 각각의 입력 ID를 연관된 토큰을 나타내는 벡터로 변환
- 후속 레이어:문장의 최종 표현을 만들기 위해 어텐션 메커니즘을 이용해 이 벡터들을 처리



HuggingFace 실습

- Tansformers의 다양한 아키텍쳐

- `*Model` (retrieve the hidden states)
- `*ForCausalLM`
- `*ForMaskedLM`
- `*ForMultipleChoice`
- `*ForQuestionAnswering`
- `*ForSequenceClassification`
- `*ForTokenClassification`

HuggingFace 실습

- 예제) 문장을 긍정 또는 부정으로 분류할 수 있게 하는 시퀀스 분류 헤드를 가진 모델
  - 모델 헤드는 이전에 봤던 고차원 벡터를 입력으로 받아 2개의 값(레이블 당 하나)으로 이루어진 벡터를 출력
  - 2개의 문장과 2개의 레이블만 있기 때문에 모델로부터 얻은 출력 형태는 2 x 2임

```
from transformers import AutoModelForSequenceClassification

checkpoint = "distilbert-base-uncased-finetuned-sst-2-english"
model = AutoModelForSequenceClassification.from_pretrained(checkpoint)
outputs = model(**inputs)
```

```
print(outputs.logits.shape)
```

```
torch.Size([2, 2])
```

HuggingFace 실습

- 출력 후처리
  - 모델은 첫 번째 문장에 대해 [-1.5607, 1.6123]으로 예측했고 두 번째 문장에 대해 [4.1692, -3.3464]으로 예측
  - 이는 확률이 아니라 모델의 마지막 층에 의해 출력된 정규화되지 않은 점수인 logits값
  - 확률로 변환되기 위해 logits은 SoftMax 층을 거쳐야 함
    - 학습을 위한 손실 함수가 일반적으로 SoftMax와 같은 마지막 활성함수와 교차 엔트로피와 같은 실제 손실 함수를 모두 사용하기 때문

```
print(outputs.logits)
```

```
tensor([[-1.5607,  1.6123],
        [ 4.1692, -3.3464]], grad_fn=<AddmmBackward>)
```

HuggingFace 실습

- 출력 후처리
  - 첫 번째 문장에 대해 [0.0402, 0.9598]로 예측했고 두 번째 모델에 대해 [0.9995, 0.0005]로 예측했으며, softmax 층을 통과한 확률 값임

```
import torch

predictions = torch.nn.functional.softmax(outputs.logits, dim=-1)
print(predictions)
```

```
tensor([[4.0195e-02, 9.5980e-01],
        [9.9946e-01, 5.4418e-04]], grad_fn=<SoftmaxBackward>)
```

HuggingFace 실습

- 출력 후처리
  - 모델 config의 id2label 속성: 각 출력값 위치에 해당하는 레이블
- 모델 예측 결과를 아래와 같이 결론 지을 수 있음
  - 첫 번째 문장: NEGATIVE: 0.0402, POSITIVE: 0.9598
  - 두 번째 문장: NEGATIVE: 0.9995, POSITIVE: 0.0005

```
model.config.id2label
```

```
{0: 'NEGATIVE', 1: 'POSITIVE'}
```