

0924 BERT 내용발표

1. BERT란

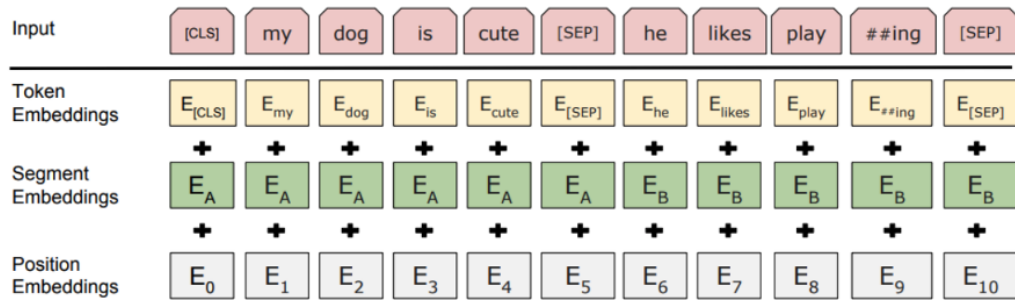
- Bidirectional Encoder Representations from Transformers
- Transformer 기반의 자연어 처리 모델
- 앞의 단어들이 아닌 문장 안의 모든 단어를 양방향으로 고려함으로써 깊이 있는 문맥 이해가 가능함.
- 한 번의 훈련을 통해 여러 작업에 적용 및 미세 조정 가능(pre-training & fine-tuning)

2. BERT의 구조

- Transformer기반 모델로 인코더-디코더 구조이지만, BERT는 인코더만 사용해 양방향 이해
- N개의 인코더 블록으로 구성
 - Self-Attention Layer : 문장 내 단어들의 연관도 계산
 - Feed-Forward Layer : 계산된 정보를 변환해 단어와 문장의 의미 더 복잡하게 표현

3. BERT의 훈련 방식

- 전처리(Preprocessing)
 - 토큰라이저를 통한 단어 분리
 - 데이터에 따라 소문자화 또는 대문자 유지
 - 각 문장의 앞에는 [CLS]를, 뒤에는 [SEP]를 붙임
 - 토큰 길이를 맞추려고 할 때는 [PAD] 추가
- 임베딩(Embedding)

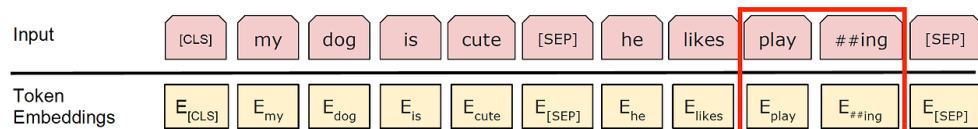


토큰 임베딩(Token Embedding)

Word Piece 임베딩 방식 활용 → 띄어쓰기 이상의 성능

어휘 사전을 기준으로 단어 하위단어 분할 → 명확한 의미 전달, 흔치 않은 단어들에 대한 예측 향상(googling, texting)

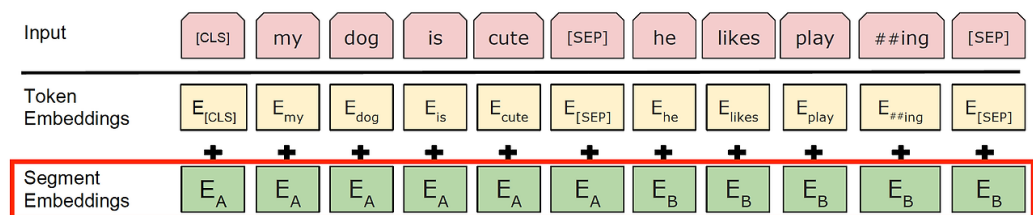
각 단어 또는 하위어에 대해 문자 단위로 고유한 임베딩 벡터



세그먼트 임베딩(Segment Embedding)

두 문장을 구분하기 위한 임베딩벡터

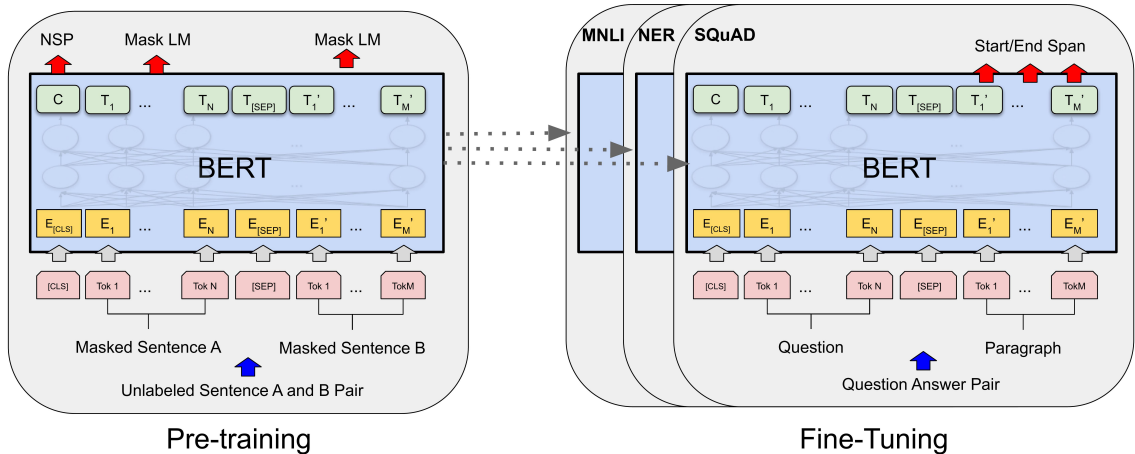
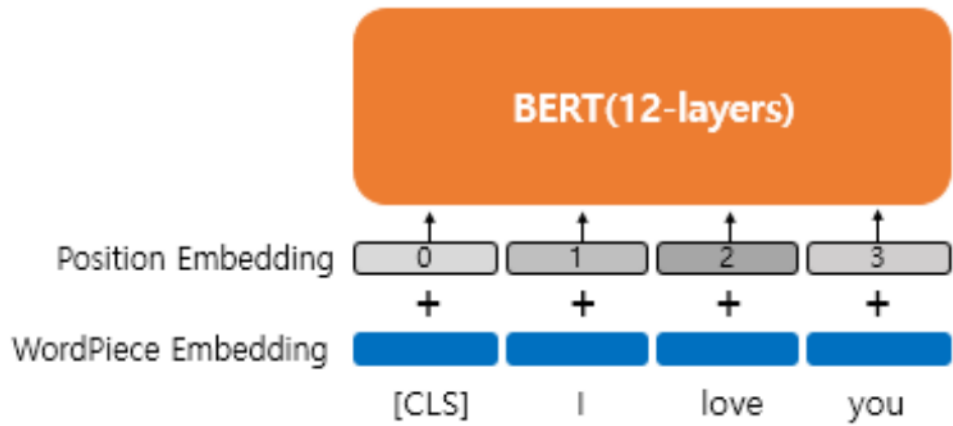
첫번째 문장에는 0을, 두번째 문장에는 1을 더해준다.



위치 임베딩(Position Embedding)

Transformer가 사용하는 Self-Attention모델은 입력 위치 고려X

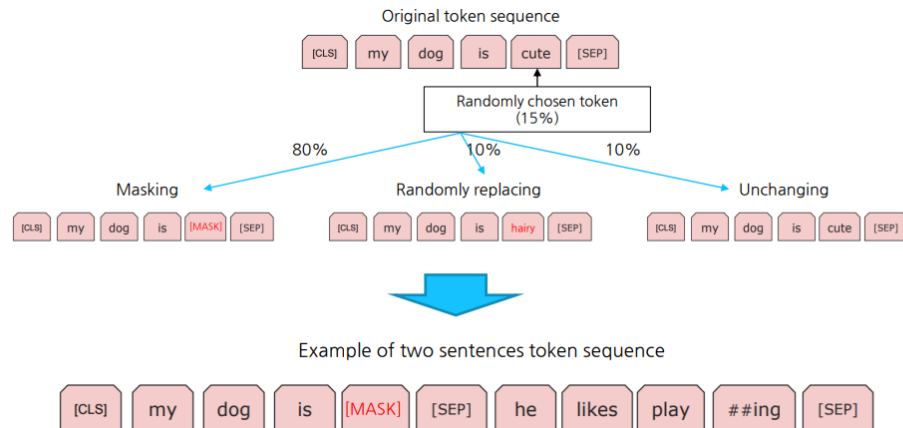
sin,cos함수를 사용하여 토큰의 순서를 인코딩한 임베딩벡터



- 사전 훈련(Pre-training)
 - 대량의 데이터를 통해 미리 학습 → 일반적인 언어 모델 구축
- 미세 조정(Fine-tuning)
 - 사전 훈련을 통해 이미 언어이해 능력이 있어 특정 작업에 따라 추가 학습 진행
 - 소량의 데이터로도 효과적인 미세 조정 가능

1. MLM(Masked Language Modeling)

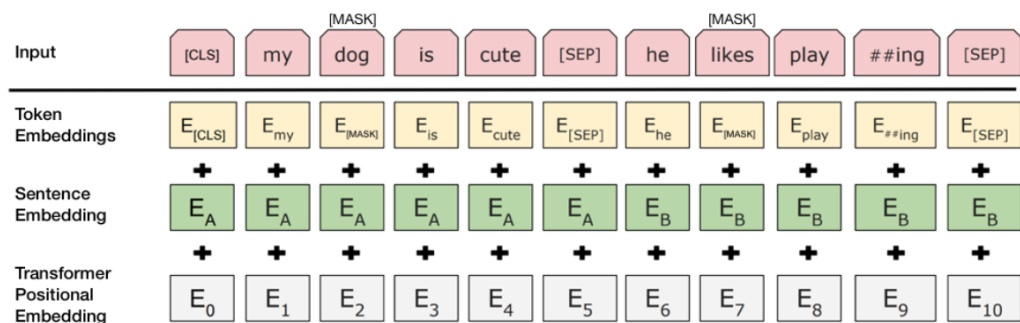
- 일부 단어 선택해 가린 후 [MASK]토큰으로 대체 후 예측
- 앞뒤 문맥을 참고하여 예측하기 때문에 풍부한 의미 이해 가능
- 문장 내 단어들의 상호작용 파악 가능 및 정확한 언어 표현 학습 가능



1. 15%의 토큰 랜덤하게 추출
2. 80% : MASK토큰으로 변경됨
 - 10% : 무작위로 다른 토큰으로 변경됨
 - 10% : 변경되지 않고 유지됨

2. NSP(Next Sentence Prediction)

- 문장의 순서 예측을 통한 흐름 이해
- 문장 간의 논리적 연결 학습으로 문서 분류 및 요약에서 높은 성능



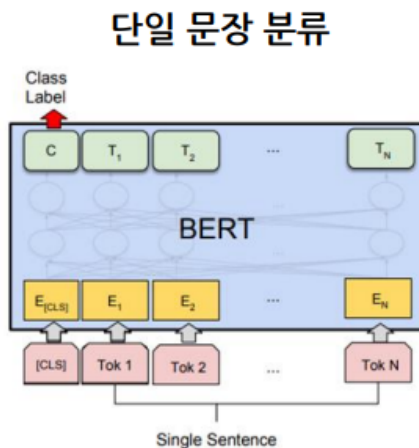
sentence layer가 추가된 이유는 첫번째 문장과 두번째 문장을 구별하기 위해 사용되는 layer

4. BERT의 장단점

- 장점
 - 문맥을 양방향으로 이해해 풍부한 언어 표현을 학습함
 - 사전훈련과 미세조정을 통해 하나의 대형 언어 모델을 조절 가능함
 - 대규모 데이터 학습이 가능하고 한번에 여러 작업 처리 가능
- 단점
 - 모델의 크기가 큰 만큼 시간과 비용 또한 큼
 - 512토큰 이상의 긴 문서나 문장에 대해서는 추가 작업 필요하며 문맥 이해도가 떨어질 수 있고 중요한 문맥 정보가 손실될 수도 있음
 - 외부 지식 및 특정 도메인에서는 낮은 성능
 - 블랙박스 모델로, 이해가 어려우며 해석 가능성이 부족함

5. BERT의 활용예시

a. 단일 문장 분류



- 입력된 문서에 대해 분류를 하는 유형

ex) 감성분석(긍부정)

```
import pandas as pd
import torch
from transformers import pipeline, BertTokenizer, BertForSequenceClassification
```

```

# 1. 문서 요약을 위한 파이프라인 생성
## 'summarization' 모델을 이용하여 문서를 요약하는 파이프라인을
summarizer = pipeline('summarization')

# 2. 감정 분석을 위한 파이프라인 생성
## 'bert-base-uncased'를 파인튜닝한 감정 분석 모델을 이용(5가지)
sentiment_model = BertForSequenceClassification.from_pretrained('nlp-town/bert-base-uncased-sentiment')
tokenizer = BertTokenizer.from_pretrained('nlp-town/bert-base-uncased-sentiment')

# 3. 분석할 문서 입력 (긴 텍스트)
document = """
The immune system is a complex network of cells and proteins that
keeps a record of every germ it has ever defeated so it can recognize
the body again. Abnormalities of the immune system can lead to various
disorders.
"""

# 4. 문서 요약
## 긴 문서를 130자 이내로 요약, 샘플링 없이 단일한 요약 생성
summary = summarizer(document, max_length=130, min_length=30)
print("문서 요약:", summary)

# 5. 요약된 문서에 대한 감정 분석
## 요약된 텍스트를 토큰화하여 감정 분석 모델에 입력
inputs = tokenizer(summary, return_tensors="pt")
outputs = sentiment_model(**inputs)

# 6. 감정 분석 결과 해석
## 모델이 감정에 대한 확률을 반환하는데, 가장 높은 확률을 가진 클래스
logits = outputs.logits
predicted_class = torch.argmax(logits, dim=1)

# 7. 감정 분류 결과 출력
## 0에서 4까지의 점수를 부여하는 모델을 사용
## 1: 매우 부정적, 2: 부정적, 3: 중립적, 4: 긍정적, 5: 매우 긍정적
sentiment_labels = {0: "매우 부정적", 1: "부정적", 2: "중립적", 3: "긍정적", 4: "매우 긍정적"}
print("감정 분석 결과:", sentiment_labels[predicted_class.item()])

```

ex) 스팸메일 분류

```
# 1. 스팸 메일과 정상 메일 예시 데이터 생성
data = {
    "email": [
        "Congratulations! You've won a $1,000 Walmart g",
        "Hi, I hope you're doing well. Let's schedule a",
        "Get paid to work from home! Limited time offer",
        "Dear customer, your invoice is attached. Pleas",
        "This is not a scam! You have a chance to win a",
        "Don't forget our appointment tomorrow at 10 AM",
        "Claim your free trial now! Act fast, this offe",
        "Thank you for your purchase! Your order will b",
    ],
    "label": [1, 0, 1, 0, 1, 0, 1, 0] # 1: 스팸, 0: 정상
}

# 데이터프레임 생성
df = pd.DataFrame(data)

# 2. 사전 학습된 BERT 모델과 토큰나이저 로드
model_name = 'bert-base-uncased'
tokenizer = BertTokenizer.from_pretrained(model_name)
model = BertForSequenceClassification.from_pretrained(m

# 3. 각 이메일에 대해 스팸 분류 수행
results = []
for email in df['email']:
    # 이메일을 BERT 입력 형태로 변환
    inputs = tokenizer(email, return_tensors="pt", padd

    # 모델에 입력하여 예측값 생성
    with torch.no_grad():
        outputs = model(**inputs)

    # 출력된 Logits에서 가장 높은 값을 가진 클래스를 선택
    logits = outputs.logits
```

```

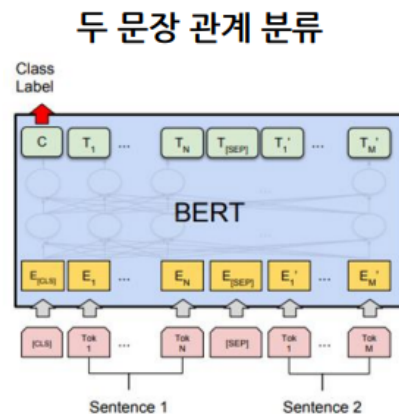
predicted_class = torch.argmax(logits, dim=1)

# 예측 결과 저장
results.append(predicted_class.item())

# 4. 결과를 데이터프레임에 추가 후 출력
df['predicted_label'] = results
print(df)

```

b. 두 문장의 관계 분류 (자연어추론_NLI)



- 문장 간의 논리적인 관계를 분류함
- 모순관계/함의관계/중립관계

ex) 관계예측(연관유무)

```

import torch
from transformers import BertTokenizer, BertForSequenceClassification

# 1. 사전 학습된 BERT 모델과 토크나이저 로드
## 'bert-base-uncased' 모델을 불러오고, 자연어 추론(NLI) 작업에 사용
tokenizer = BertTokenizer.from_pretrained('textattack/bert-base-uncased')
model = BertForSequenceClassification.from_pretrained('textattack/bert-base-uncased')

# 2. 두 문장 입력

```



```

## 첫 번째 문장은 전제(premise), 두 번째 문장은 가설(hypothesis)
premise = "The weather is nice today."
hypothesis = "It is sunny and pleasant outside."

# 3. 두 문장을 BERT 입력 형태로 변환 (BERT는 두 문장을 [SEP] 토큰
## 두 문장을 토크나이저를 이용해 하나의 입력으로 변환
## padding(길이 맞추기) truncation(길이 제한)
inputs = tokenizer(premise, hypothesis, return_tensors="

# 4. 모델에 입력하여 문장 관계 예측
## 두 문장 사이의 관계를 예측하는 logits 출력 (세 관계에 대한 점수
outputs = model(**inputs)

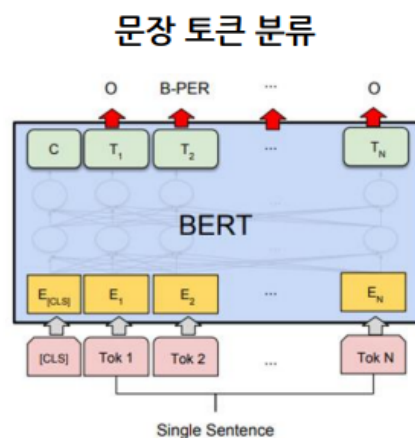
# 5. 출력된 Logits에서 가장 높은 값을 가진 클래스를 선택
## logits는 3개의 값을 가지며, 각각 Entailment(참), Neutral(중립),
logits = outputs.logits
predicted_class = torch.argmax(logits, dim=1)

# 6. 관계 레이블 설정 (0: Contradiction, 1: Neutral, 2: Entailment)
relationship_labels = {0: "Contradiction (거짓)", 1: "Neutral", 2: "Entailment"}

# 7. 예측된 관계 출력
print(f"두 문장 사이의 관계: {relationship_labels[predicted_class]}")

```

c. 문장 토큰 분류



ex) 번역

```
from transformers import MarianMTModel, MarianTokenizer

# 1. 번역할 언어 모델과 토크나이저 로드
## 'Helsinki-NLP/opus-mt-en-ko'는 영어에서 한국어로 번역하는 모델
model_name = 'Helsinki-NLP/opus-mt-en-ko'
tokenizer = MarianTokenizer.from_pretrained(model_name)
model = MarianMTModel.from_pretrained(model_name)

# 2. 번역할 영어 문장 입력
english_sentence = "The weather is nice today."

# 3. 입력 문장을 토크나이저하여 텐서 형태로 변환
## 모델에 입력하기 위해 토큰화 및 인코딩
inputs = tokenizer(english_sentence, return_tensors="pt",

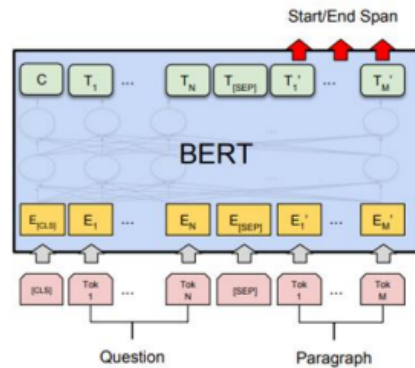
# 4. 모델에 입력하여 번역 생성
## 모델의 forward 메서드를 호출하여 번역 결과를 얻음
with torch.no_grad(): # 그래디언트 계산을 하지 않음
    translated_tokens = model.generate(**inputs)

# 5. 토큰을 디코딩하여 번역된 문장 생성
translated_sentence = tokenizer.decode(translated_tokens[0]

# 6. 번역 결과 출력
print("영어 문장:", english_sentence)
print("번역된 한국어 문장:", translated_sentence)
```

d. 기계 독해 정답 분류 (질의응답)

기계 독해 정답 분류



- 주어진 질문-본문을 보고 본문에서 일부를 추출해 대답함

ex) 질의응답

```
# 라이브러리 설치 및 모델 다운로드
import torch
from transformers import BertForQuestionAnswering, BertTokenizer
model = BertForQuestionAnswering.from_pretrained('bert-large-uncased')
tokenizer = BertTokenizer.from_pretrained('bert-large-uncased')

# 질문과 본문 입력
question = "What is the immune system?"
paragraph = "The immune system is a system of many biological processes that help the body defend itself against disease."

#토큰화
question = '[CLS] ' + question + '[SEP]'
paragraph = paragraph + '[SEP]'
question_tokens = tokenizer.tokenize(question)
paragraph_tokens = tokenizer.tokenize(paragraph)

#토큰 결합
tokens = question_tokens + paragraph_tokens
input_ids = tokenizer.convert_tokens_to_ids(tokens)

#세그먼트 임베딩
segment_ids = [0] * len(question_tokens)
segment_ids += [1] * len(paragraph_tokens)
```

```
#텐서로 변환 후 입력
input_ids = torch.tensor([input_ids])
segment_ids = torch.tensor([segment_ids])
scores = model(input_ids, token_type_ids = segment_ids)

#시작과 끝 인덱스
start_index = torch.argmax(scores.start_logits)
end_index = torch.argmax(scores.end_logits)

#답변 출력
print(' '.join(tokens[start_index:end_index+1]))
```