

BERT : Architecture, Fine-tuning, and Attention Visualization Method

BERT의 아키텍처와 미세조정, 그리고 어텐션 시각화 기법

이형준 | 동국대학교 컴퓨터공학과

2025.05.30

BackGround(1)



BERT란 Google에서 2018년에 발표한 양방향 트랜스포머 기반의 언어 모델.
당시 다양한 NLP 태스크에서 SOTA(State-Of-The-Art)를 달성하며 큰 주목을 받았다.



Pre-training

- MLM(Masked Language Modeling) : 마스킹된 토큰을 예측
- NSP(Next Sentence Prediction) : 문장의 순서 예측
- 대규모 Corpus에서 활용

“BERT is designed to pre-train deep bidirectional representations from unlabeled text...”



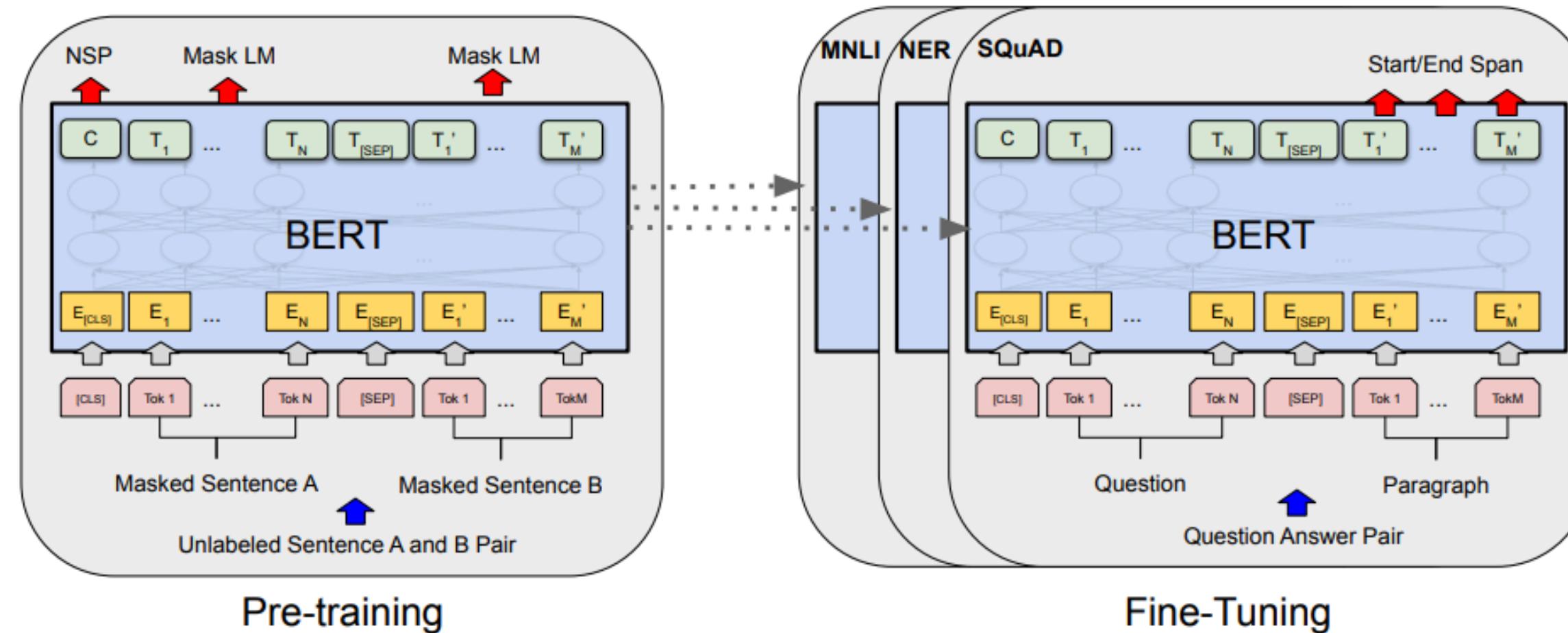
Fine-tuning

- Task-specific Object를 적용, 다양한 Task에 적용
- Pre-training된 가중치를 활용
- 전체 모델, 혹은 일부 레이어의 가중치만 업데이트 가능

“For fine-tuning, all of the parameters are fine-tuned using labeled data from the downstream tasks.”

BackGround(1)

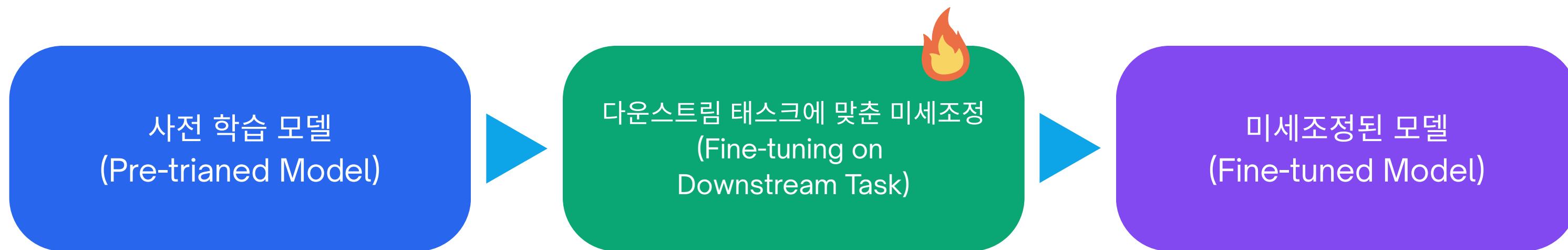
BERT: Bidirectional Encoder Representations from Transformers



BackGround(2)

Fine-tuning

사전 학습(Pre-training)된 모델을 특정 다운스트림 태스크에 적용시키는 작업

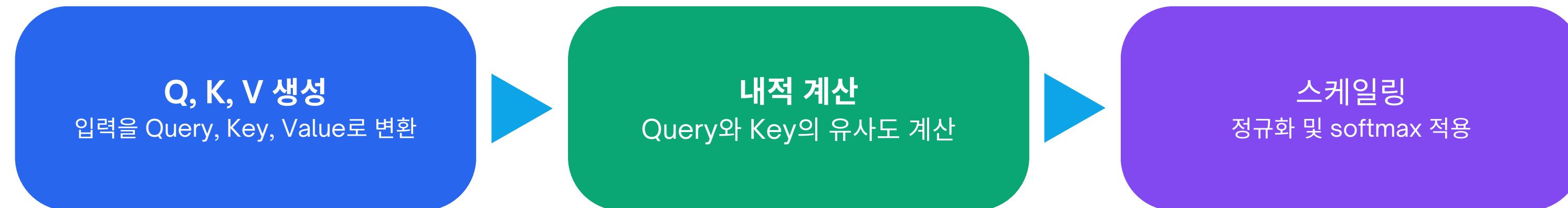


1. 지식 전이(Knowledge Transfer)를 통한 효율적인 학습
2. 일반화(Generalization) 성능 향상
3. 과적합(Overfitting) 방지
4. 안정적인 학습과 빠른 수렴
5. 적은 계산자원 요구
6. 도메인/태스크에 특화된 모델 개발 가능

BackGround(3)

Attention Machanism

모델이 중요한 정보에 집중(Attention)할 수 있는 방법



$$\text{Attention}(Q, K, V) = \text{Attention value}$$

$$Q : \text{Query}$$
$$K : \text{Key}$$
$$V : \text{Value}$$

$$a_t = \sum_{i=1}^N \alpha_i^t h_i$$

$$\text{score}(s_t, h_i) = \frac{s_t^T h_i}{\sqrt{n}}$$

Attention의 문맥적 의미

- 입력의 주요한 부분에 집중
- 가중치를 통한 정보 선별
- 문맥(Context) 이해 능력 향상

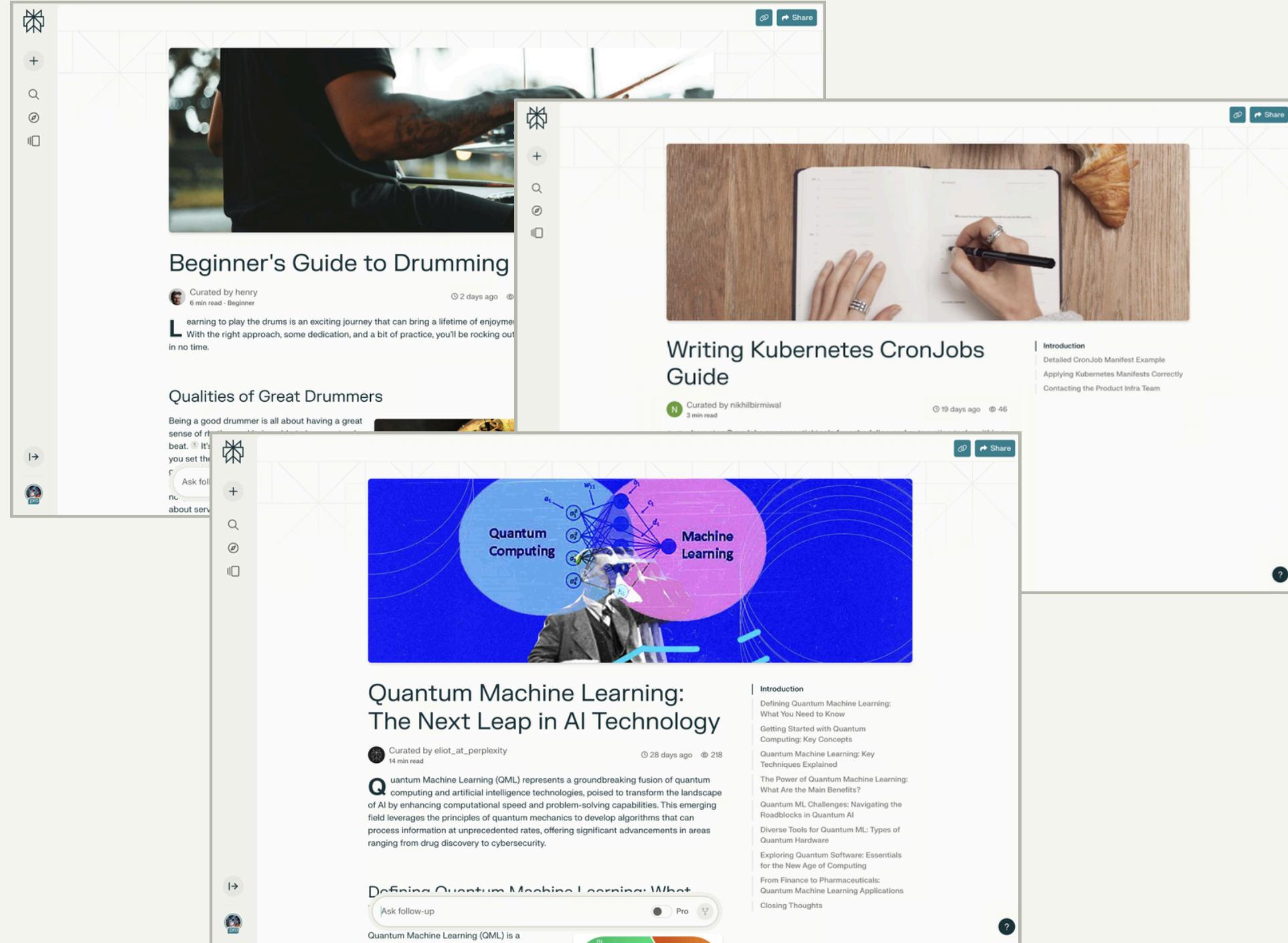
Self-Attention

- 문장 내 단어 간 상호 작용을 학습
- 모든 단어가 서로에게 영향을 미친다
- 병렬 처리 가능

Multi-head Attention

- 여러 Attention을 동시에 수행
- 다양한 관점에서 정보를 병렬 처리
- 더 풍부한 표현과 강력한 학습 능력

Experiment(LLM)



Perplexity Pro

2022년 Perplexity AI가 출시한 AI 기반 지식 검색 엔진.

- 무제한 Pro Search
- 다양한 Ai 모델 선택 가능:
 - Sonar, GPT-4.1, Claude 4.0 Sonnet, Gemini 2.5Pro, Grok 3 Beta
 - R1 1776, o4-mini, Claude 4.0 Sonnet(Reasoning)
- 개수 제한 없는 파일 업로드 가능
- Deep Research 무제한 이용 가능

Experiment(LLM)

Perplexity DeepResearch

Perplexity의 자율적 심층 연구 및 분석 도구

- **Research with reasoning:**
 - 반복적인 검색, 문서 읽기, 연구 계획 개선
- **Report Writing:**
 - 출처 자료들을 모두 평가한 후, agent가 연구 내용을 명확히 하여 종합적인 보고서를 작성
- **Export & Share:**
 - 완성된 보고서를 PDF, 문서 파일로 내보내기 가능

Pre-Market Checklist.pdf
Page 1 of 4

perplexity

Essential Pre-Market Checklist: Key Considerations Before the Market Opens

Before diving into the day's trading activities, market participants must arm themselves with a structured approach to navigate the complexities of pre-market dynamics. The following report synthesizes critical elements from expert sources to create a comprehensive guide for informed decision-making prior to market open.

Global Market Sentiment and Overnight Developments

Assessing International Market Activity

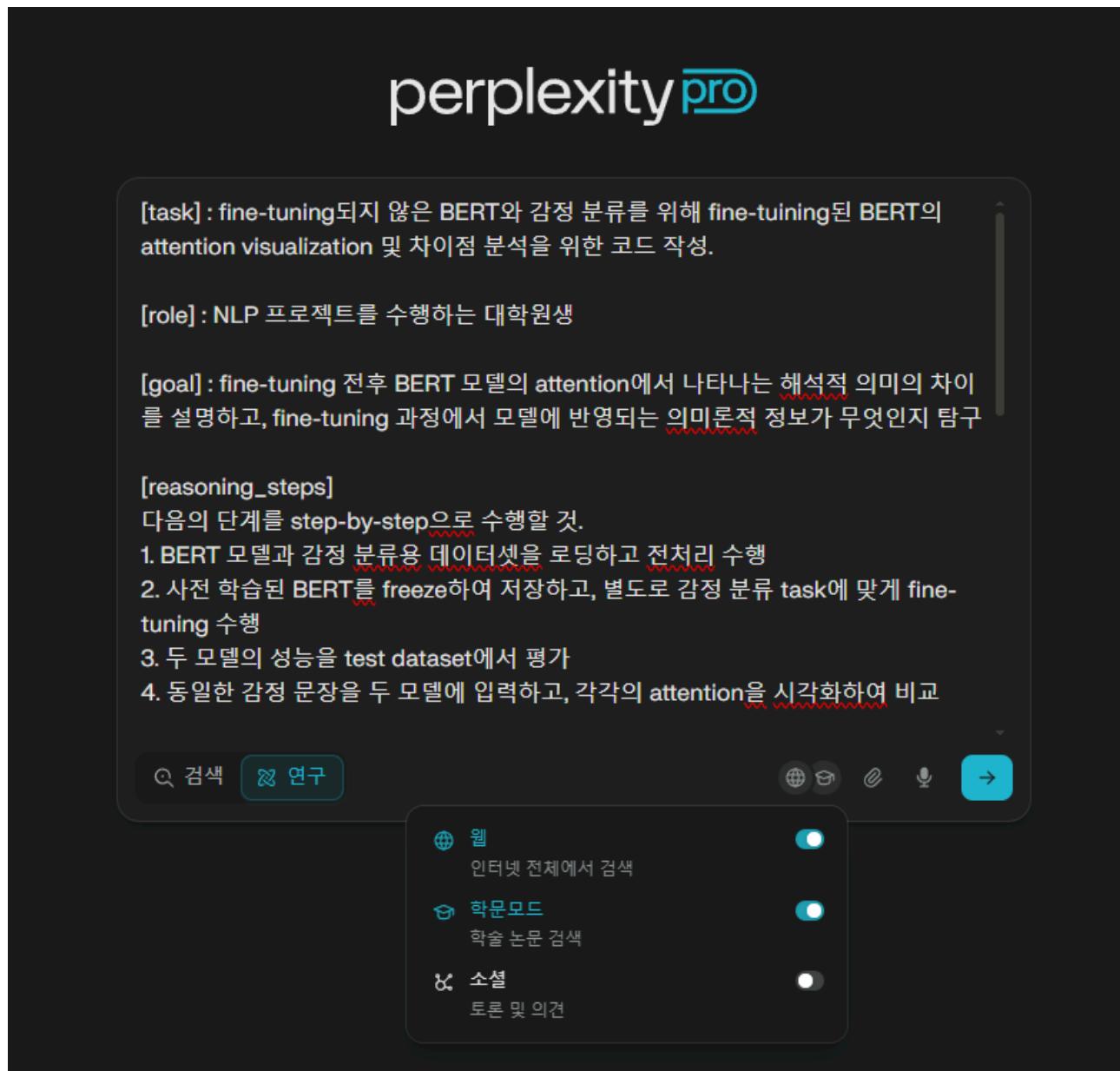
Extended trading sessions, including pre-market hours (4:00 a.m. to 9:30 a.m. ET for U.S. exchanges), allow traders to react to global events that occur outside regular hours^{[1][2][3]}. For instance, geopolitical tensions in Asia or central bank decisions in Europe can ripple through U.S. equity futures, setting the tone for the trading day. Monitoring indices like the Nikkei 225 or DAX performance provides clues about sector-specific momentum—a practice highlighted by institutional traders who analyze overnight gaps in S&P 500 futures to identify support/resistance levels^{[4][5]}.

Macroeconomic Catalysts and News Filters

Reviewing macroeconomic calendars is non-negotiable. The December 2024 Personal Consumption Expenditures (PCE) report, which showed inflation rising to 2.6% year-over-year, exemplifies how pre-market reactions to data releases can dictate opening volatility^[6]. Traders must also filter news for relevance: earnings surprises (e.g., Apple's Q1 2025 revenue hitting \$124.3B despite iPhone slumps^{[6][7]}), mergers, or regulatory changes (e.g., Trump's 2025 tariff announcements^[7]) require immediate analysis to avoid being blindsided by algorithmic price swings^{[4][8]}.

Experiment(LLM)

Mode Setting



Prompt 전문

[task] : fine-tuning되지 않은 BERT와 감정 분류를 위해 fine-tuning된 BERT의 attention visualization 및 차이점 분석을 위한 코드 작성.

Role-Play Prompting
(Kong, et al., 2023)

[role] : NLP 프로젝트를 수행하는 대학원생

[goal] : fine-tuning 전후 BERT 모델의 attention에서 나타나는 해석적 의미의 차이를 설명하고, fine-tuning 과정에서 모델에 반영되는 의미론적 정보가 무엇인지 탐구

[reasoning_steps]

다음의 단계를 step-by-step으로 수행할 것.

- BERT 모델과 감정 분류용 데이터셋을 로딩하고 전처리 수행
- 사전 학습된 BERT를 freeze하여 저장하고, 별도로 감정 분류 task에 맞게 fine-tuning 수행
- 두 모델의 성능을 test dataset에서 평가
- 동일한 감정 문장을 두 모델에 입력하고, 각각의 attention을 시각화하여 비교

Chain-of-Thought
(Wei, et al., 2022)

[instructions] :

- 어텐션 시각화 방법은 (1) BERTViz, (2) exBERT, (3) Language Interpretability Tool (LIT)를 사용.
- fine-tuning 전후 BERT의 attention 차이를 위 방법론을 활용해 시각화.
- 동일 문장에 대한 fine-tuning 전후 모델의 attention 시각화를 나란히(side-by-side) 비교하는 방식으로 구성
- 감정 분류 데이터셋은 자유롭게 선정하되, 링크 및 다운로드 방법을 명시

Recipe, Template
(White, et al., 2023)

[output format]:

- 코드 실행이 단계적으로 가능하도록 여러 개의 Python code block으로 작성.
- 각 단계마다 충분한 설명과 주석 포함.
- 분석 결과는 Markdown 형식으로 정리.

[constraints]

- GPU를 사용하지 말 것.
- Colab 환경에서 코드를 작성할 것

Experiment(LLM)

Reasoning Steps

```
[task] : fine-tuning되지 않은 BERT와 감정 분류를 위해 fine-tuning된 BERT의 attention visualization 및 차이점 분석을 위한 코드 작성.

[role] : NLP 프로젝트를 수행하는 대학원생

[goal] : fine-tuning 전후 BERT 모델의 attention에서 나타나는 해석적 의미의 차이를 설명하고,
더 표시하기

 연구 ○ 작업 ◆ 출처 - 170

I am searching for resources and explanations related to transformer model explainability, including GitHub repositories, Colab notebooks for BERT interpretability, and research papers on transformer interpretability beyond attention visualization.

검색 중
Q hila-chefer Transformer Explainability GitHub Q BERT_explainability colab
Q Transformer Interpretability Beyond Attention Visualization paper

읽는 중
arxiv arxiv arxiv github hila-chefer.github github github
github

I need to find resources related to explaining transformer models, specifically BERT, to understand how to interpret its decisions. Searching for notebooks and Colab tutorials will help identify practical methods and explanations for BERT explainability.

검색 중
Q Transformer-Explainability BERT_explainability.ipynb Q BERT_explainability colab

읽는 중
arxiv arxiv arxiv arxiv arxiv arxiv arxiv arxiv arxiv

I am retrieving content from these URLs to access information related to transformer explainability and BERT explainability, which are likely relevant to understanding model interpretability techniques.

I am searching for information on methods and research related to understanding and interpreting transformer models beyond just attention visualization, to gain deeper insights into their inner workings.

검색 중
Q Transformer Interpretability Beyond Attention Visualization

읽는 중
arxiv arxiv arxiv arxiv arxiv arxiv arxiv arxiv arxiv

I need to access the content of the provided URL to review the information available on the GitHub page, which may contain relevant details or instructions related to the topic at hand.

I am searching for information related to the "BERT_explainability.ipynb" notebook, specifically focusing on the work by Hila Chefer on transformer explainability, to understand methods and insights into explaining BERT models.

검색 중
Q raw BERT_explainability.ipynb hila-chefer Transformer-Explainability
```

Output

```
[task] : fine-tuning되지 않은 BERT와 감정 분류를 위해 fine-tuning된 BERT의 attention visualization 및 차이점 분석을 위한 코드 작성.

[role] : NLP 프로젝트를 수행하는 대학원생

[goal] : fine-tuning 전후 BERT 모델의 attention에서 나타나는 해석적 의미의 차이를 설명하고,
더 표시하기

 연구 ○ 작업 ◆ 출처 - 170

arxiv Improving Semantic Segmentation in...
arxiv VisBERT: Hidden-State Visualizations for...
arxiv Transformer Interpretability Beyond Attention...
thevf [PDF] Transformer Interpretability Beyond...

Fine-tuning 전후 BERT 모델의 Attention 시각화 및 차이점 분석

본 연구는 사전 학습된 BERT 모델과 감정 분류를 위해 fine-tuning된 BERT 모델의 attention 메커니즘을 시각화하고 비교 분석하여, fine-tuning 과정에서 모델이 학습하는 의미론적 정보의 변화를 탐구합니다. Transformer Explainability Beyond Attention Visualization 방법론을 활용하여 두 모델의 attention 패턴을 정량적으로 비교하고, 감정 분류 태스크에 특화된 학습이 모델의 해석적 특성에 미치는 영향을 분석합니다.

환경 설정 및 라이브러리 설치

먼저 필요한 라이브러리들을 설치하고 import합니다. Transformer Explainability 방법론을 구현하기 위해 해당 GitHub 저장소에서 제공하는 코드를 활용합니다 14 .

python
# 필요한 라이브러리 설치
!pip install transformers datasets torch matplotlib seaborn numpy pandas
scikit-learn
!pip install git+https://github.com/hila-chefer/Transformer-Explainability.git

# 기본 라이브러리 import
import torch
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from transformers import (
    AutoTokenizer, AutoModelForSequenceClassification,
    AutoModel, TrainingArguments, Trainer
)
from datasets import load_dataset, Dataset
```

Experiment

> Python Implementation

```
●●● c++  
1 # 필요한 라이브러리 설치  
2 !pip install transformers datasets torch bertviz lit-nlp matplotlib seaborn pandas numpy  
3 !pip install -U datasets  
4 !pip install -U accelerate  
5  
6 # 기본 라이브러리 import  
7 import torch  
8 import numpy as np  
9 import pandas as pd  
10 import matplotlib.pyplot as plt  
11 import seaborn as sns  
12 from transformers import (  
13     AutoTokenizer,  
14     AutoModelForSequenceClassification,  
15     BertModel,  
16     BertTokenizer,  
17     Trainer,  
18     TrainingArguments,  
19     pipeline  
20 )  
21 from datasets import load_dataset, Dataset  
22 from bertviz import model_view, head_view  
23 import warnings  
24 warnings.filterwarnings('ignore')  
25  
26 # 재현성을 위한 시드 설정  
27 torch.manual_seed(42)  
28 np.random.seed(42)  
29  
30 print("라이브러리 설치 및 import 완료")  
31 print(f"PyTorch version: {torch.__version__}")  
32 print(f"CUDA available: {torch.cuda.is_available()}")  
33
```

BERT 기반의 자연어 처리 프로젝트를 위한 **라이브러리** (transformers, datasets, torch, bertviz, etc...)를 설치하고 환경을 설정

필요한 모듈을 import한 뒤, BERT 토크나이저와 모델, 시각화 도구, 학습 관련 클래스들을 사용할 수 있도록 설정. 시각화에는 **bertviz**, 데이터 분석에는 **pandas**, **seaborn** 등을 활용

난수 시드를 고정해 실험의 **재현성**을 확보하고, PyTorch 버전 및 CUDA 사용 가능 여부를 출력하여 환경 구성을 마무리.

Experiment

> Python Implementation

```
python
● ● ●
1 # IMDb 감정 분석 데이터셋 로딩
2 print("IMDb 데이터셋 로딩 중...")
3 dataset = load_dataset("imdb")
4
5 print("데이터셋 정보:")
6 print(dataset)
7 print(f"\n훈련 데이터 크기: {len(dataset['train'])}")
8 print(f"테스트 데이터 크기: {len(dataset['test'])}")
9
10 # 샘플 데이터 확인
11 print("\n샘플 데이터:")
12 for i in range(2):
13     sample = dataset['train'][i]
14     print(f"텍스트: {sample['text'][:100]}...")
15     print(f"레이블: {sample['label']} ({'긍정' if sample['label'] == 1 else '부정'})")
16     print("-" * 50)
17
18 # 작은 서브셋 생성 (GPU 미사용 환경에서의 빠른 실험을 위함)
19 small_train = dataset['train'].shuffle(seed=42).select(range(800))
20 small_test = dataset['test'].shuffle(seed=42).select(range(200))
21
22 print(f"\n서브셋 크기 - 훈련: {len(small_train)}, 테스트: {len(small_test)}")
```

IMDb 영화 리뷰 감정 분석 데이터셋을 로드.
load_dataset("imdb")를 통해 훈련용과 테스트용
데이터를 로딩

각 리뷰는 0(부정) 또는 1(긍정)으로 라벨링되어 있으며,
첫 두 개의 리뷰를 출력해 텍스트 일부와 감정 분류 결과
를 확인.

GPU 없이도 빠르게 실험할 수 있도록 훈련 데이터에서
800개, 테스트 데이터에서 200개만 랜덤 추출해 작은 서
브셋을 생성

Experiment

> Python Implementation

```
1 # 모델과 토크나이저 초기화
2 model_name = "distilbert-base-uncased"
3 print(f"모델 로딩: {model_name}")
4
5 # 토크나이저 로딩
6 tokenizer = AutoTokenizer.from_pretrained(model_name)
7
8 # 사전 학습된 BERT 모델 (attention 출력을 위해)
9 pretrained_bert = BertModel.from_pretrained(model_name, output_attentions=True)
10 pretrained_bert.eval()
11
12 # Fine-tuning을 위한 분류 모델
13 classification_model = AutoModelForSequenceClassification.from_pretrained(
14     model_name,
15     num_labels=2,
16     output_attentions=True
17 )
18
19 print("모델 로딩 완료")
20 print(f"모델 파라미터 수: {sum(p.numel() for p in classification_model.parameters()):,}")
21
```

사전 학습된 DistilBERT 모델과 토크나이저를 로드 및 초기화. AutoModelForSequenceClassification은 감정 분류를 위한 **fine-tuning**용 모델로 설정

> Python Implementation

```
1 def tokenize_function(examples):
2     """텍스트 토큰화 함수"""
3     return tokenizer(
4         examples["text"],
5         padding="max_length",
6         truncation=True,
7         max_length=512
8     )
9
10 # 데이터셋 토큰화
11 print("데이터셋 토큰화 중...")
12 tokenized_train = small_train.map(tokenize_function, batched=True)
13 tokenized_test = small_test.map(tokenize_function, batched=True)
14
15 # 불필요한 컬럼 제거
16 tokenized_train = tokenized_train.remove_columns(["text"])
17 tokenized_test = tokenized_test.remove_columns(["text"])
18
19 # PyTorch 텐서 형태로 변환
20 tokenized_train.set_format("torch")
21 tokenized_test.set_format("torch")
22
23 print("데이터 전처리 완료")
24
```

텍스트 데이터를 **토큰화**하고 **전처리**하는 과정. 각 텍스트를 최대 512 토큰까지 패딩 및 자르기를 수행한 후, 불필요한 text 제거. 최종적으로 데이터를 **PyTorch 텐서** 형식으로 변환

Experiment

> Python Implementation

```
● ● ● c++  
1 import os  
2 os.environ["WANDB_DISABLED"] = "true"  
3  
4 # 훈련 인자 설정  
5 training_args = TrainingArguments(  
6     output_dir=".results", num_train_epochs=4,  
7     per_device_train_batch_size=128, per_device_eval_batch_size=128,  
8     warmup_steps=100, weight_decay=0.01,  
9     logging_dir=".logs", logging_steps=50,  
10    save_strategy="epoch", no_cuda=True # CPU만 사용  
11 )  
12  
13 # Trainer 초기화  
14 trainer = Trainer(  
15     model=classification_model, args=training_args,  
16     train_dataset=tokenized_train, eval_dataset=tokenized_test,  
17     tokenizer=tokenizer,  
18 )  
19  
20 print("Fine-tuning 시작...")  
21 trainer.train()  
22  
23 print("Fine-tuning 완료!")  
24  
25 # Fine-tuned 모델 저장  
26 trainer.save_model("./fine_tuned_bert")
```

Trainer와 TrainingArguments를 사용해 DistilBERT 모델을 IMDb 감정 분석 데이터에 맞게 파인튜닝. 학습은 총 4 epoch, batch size는 128로 설정하고 CPU 환경에서 진행.

총 학습 시간은 약 44분 소요. 데이터셋을 소형으로 구성하고 DistilBERT처럼 경량 모델을 사용했기 때문에 CPU에서도 학습 시간이 크게 늘어나지 않음.

Fine-tuned 모델은 테스트셋에서 accuracy 82.9%, 파인튜닝 전 Pretrained 모델은 accuracy 47.7%를 기록. 파인튜닝 전후 성능 차이가 뚜렷하게 나타나며, 파인튜닝을 통해 분류 능력이 크게 향상됨을 시사

Experiment

> Python Implementation

```
1 def extract_attention(model, tokenizer, text, model_type="classification"):
2     """
3         모델에서 attention 가중치를 추출하는 함수
4
5     Args:
6         model: BERT 모델
7         tokenizer: 토크나이저
8         text: 입력 텍스트
9         model_type: "classification" 또는 "pretrained"
10
11    Returns:
12        attention weights, tokens
13    """
14    # 텍스트 토큰화
15    inputs = tokenizer(text, return_tensors="pt", padding=True, truncation=True, max_length=512)
16    tokens = tokenizer.convert_ids_to_tokens(inputs['input_ids'][0])
17
18    model.eval()
19    with torch.no_grad():
20        if model_type == "classification":
21            outputs = model(**inputs, output_attentions=True)
22        else:
23            outputs = model(**inputs, output_attentions=True)
24
25    # Attention 가중치 추출 (모든 레이어의 attention)
26    attentions = outputs.attentions # Tuple of attention weights for each layer
27
28    return attentions, tokens, inputs
29
30 # 테스트 문장들 정의
31 test_sentences = [
32     "This movie is absolutely fantastic! I loved every minute of it.",
33     "The film was terrible and boring. I hated it.",
34     "The movie was okay, nothing special but not bad either.",
35 ]
36
37 print("Attention 추출 준비 완료")
38 print("테스트 문장들:")
39 for i, sentence in enumerate(test_sentences):
40     print(f"{i+1}. {sentence}")
41
```

입력된 텍스트에 대해 BERT 기반 모델의 attention 가중치를 추출하는 기능을 수행. 텍스트는 토크나이저를 통해 토큰화되며, 이후 모델에서 각 레이어의 attention 값 을 출력

함수는 attention 가중치와 해당 입력에 대한 토큰 리스트, 모델 입력값을 반환. attention은 각 층별로 다차원 텐서 형태로 제공되어 시각화나 분석에 활용 가능.

모델 테스트를 위해 총 세 개의 문장을 정의. 첫 문장은 **긍정적인 리뷰**, 두 번째는 **부정적인 평가**, 세 번째는 **중립적인 반응**을 나타냄.

Experiment

> Python Implementation

```
1 from bertviz import head_view, model_view
2 from IPython.display import display, HTML
3
4 def visualize_attention_bertviz(model, tokenizer, text, model_name, model_type="classification"):
5     """BertViz를 사용한 attention 시각화"""
6
7     # 입력 준비
8     inputs = tokenizer(text, return_tensors="pt", padding=True, truncation=True, max_length=128)
9
10    model.eval()
11    with torch.no_grad():
12        outputs = model(*inputs, output_attentions=True)
13
14    attention = outputs.attentions
15    tokens = tokenizer.convert_ids_to_tokens(inputs['input_ids'][0])
16
17    print(f"\n{n==={model_name} - Attention Visualization ==}")
18    print(f"텍스트: {text}")
19    print(f"토큰 수: {len(tokens)}")
20
21    # Head View (마지막 레이어의 모든 헤드)
22    try:
23        print(f"\n{model_name} - Head View (Layer 11):")
24        head_view(attention, tokens, layer=5, heads=list(range(12)))
25    except Exception as e:
26        print(f"Head view 시각화 중 오류: {e}")
27
28    # Model View (모든 레이어와 헤드의 캐요)
29    try:
30        print(f"\n{model_name} - Model View:")
31        model_view(attention, tokens)
32    except Exception as e:
33        print(f"Model view 시각화 중 오류: {e}")
34
35 # 사전 학습된 모델과 fine-tuned 모델 비교
36 test_sentence = test_sentences[0] # 긍정적인 문장
37
38 print("==== BERT Attention 시각화 비교 ===")
39
40 # 1. 사전 학습된 BERT
41 print("\n1. 사전 학습된 BERT 모델")
42 visualize_attention_bertviz(pretrained_bert, tokenizer, test_sentence, "Pretrained BERT", "pretrained")
43
44 # 2. Fine-tuned BERT
45 print("\n2. Fine-tuned BERT 모델")
46 visualize_attention_bertviz(classification_model, tokenizer, test_sentence, "Fine-tuned BERT", "classification")
```

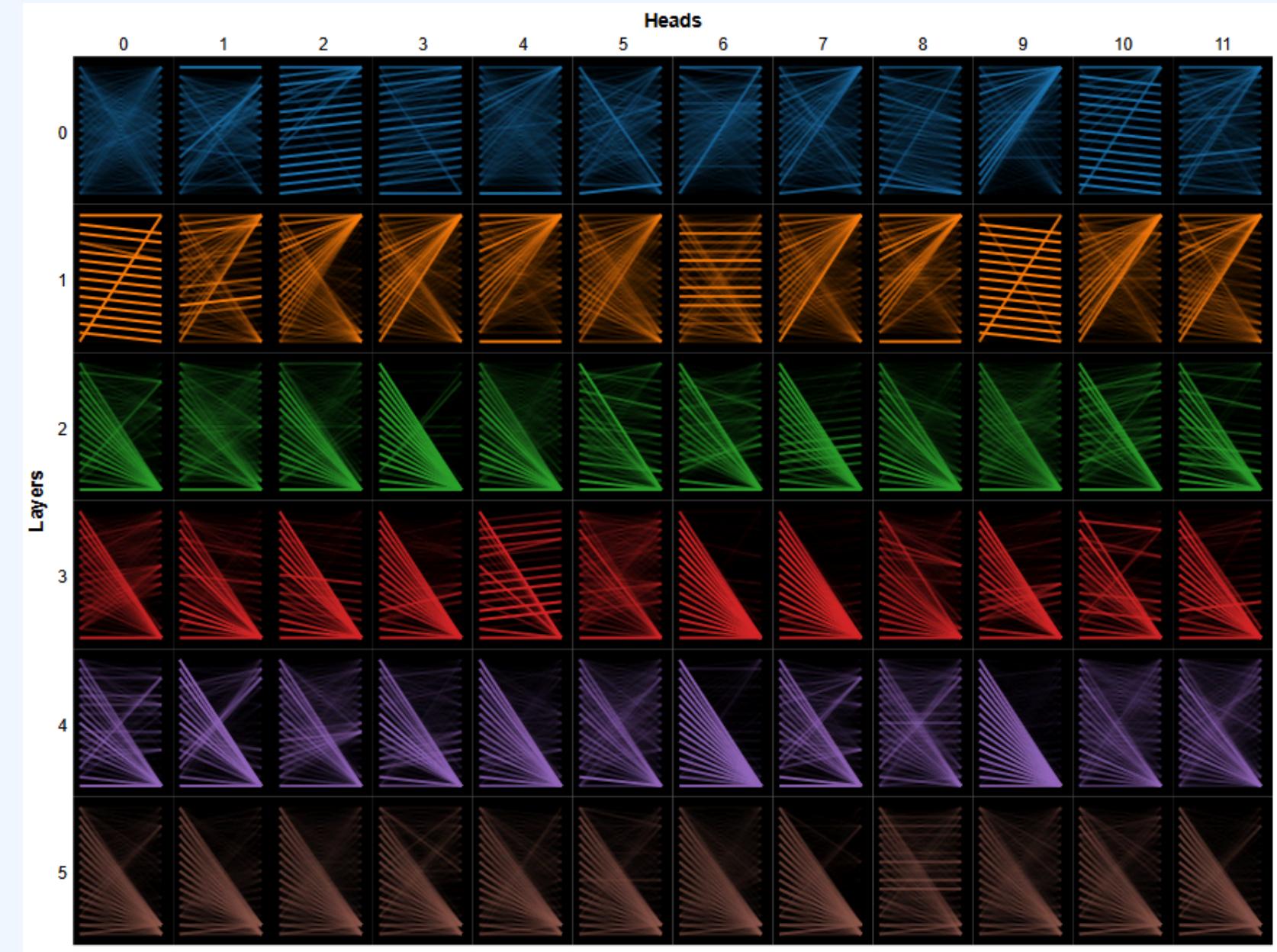
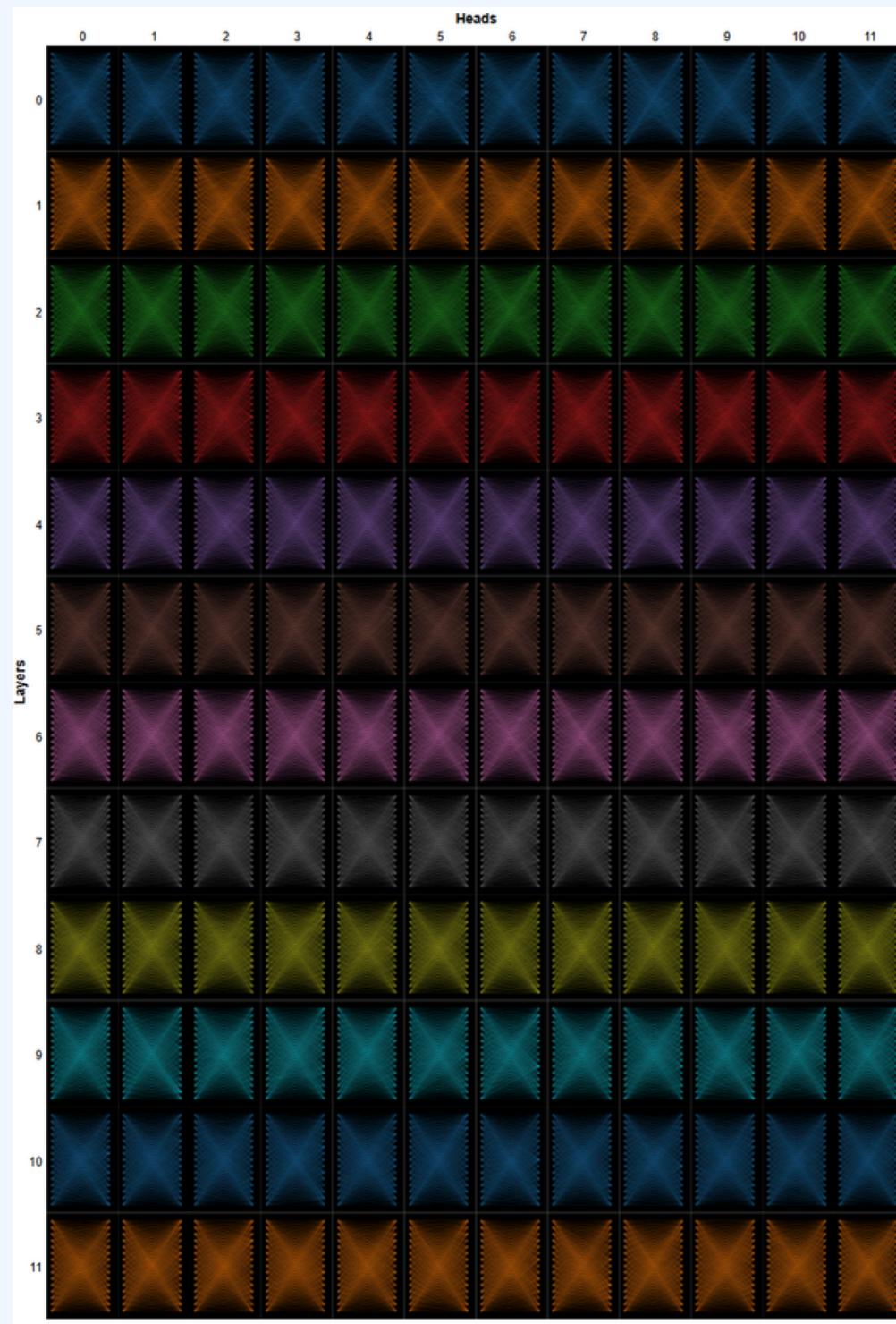


BertViz는 BERT 모델의 attention 메커니즘을 시각적으로 분석하는 도구다. 입력 문장의 각 토큰에 대해 모델이 어디에 집중하는지 시각화한다.

사전 학습된 BERT와 파인튜닝된 BERT 모델에 대해 같은 문장으로 attention 시각화를 수행해, 두 모델의 attention 분포 차이를 비교한다.

Analysis

Visualization Result



Experiment

> Python Implementation

```
●●● python
1 def plot_attention_heatmap(attention_weights, tokens, title, ax):
2     """
3     Attention 가중치 히트맵 시각화
4
5     Args:
6         attention_weights: 평균 attention 가중치 [seq_len, seq_len]
7         tokens: 토큰 리스트
8         title: 그래프 제목
9         ax: matplotlib axis
10    """
11    # 토큰이 너무 많으면 일부만 표시
12    max_tokens = 20
13    if len(tokens) > max_tokens:
14        attention_weights = attention_weights[:max_tokens, :max_tokens]
15        tokens = tokens[:max_tokens]
16
17    # 히트맵 그리기
18    im = ax.imshow(attention_weights, cmap='Blues', aspect='auto')
19
20    # 축 설정
21    ax.set_xticks(range(len(tokens)))
22    ax.set_yticks(range(len(tokens)))
23    ax.set_xticklabels(tokens, rotation=45, ha='right', fontsize=8)
24    ax.set_yticklabels(tokens, fontsize=8)
25
26    ax.set_xlabel('To Token')
27    ax.set_ylabel('From Token')
28    ax.set_title(title, fontsize=12, fontweight='bold')
29
30    # 컬러바 추가
31    plt.colorbar(im, ax=ax, fraction=0.046, pad=0.04)
32
33    return im
```

Attention 가중치를 **히트맵**으로 시각화하는 함수는 토큰 별 attention 강도를 한눈에 파악할 수 있도록 한다. x축과 y축에 토큰을 배치해 어느 토큰에서 어느 토큰으로 집중되는지 쉽게 확인 가능하다.

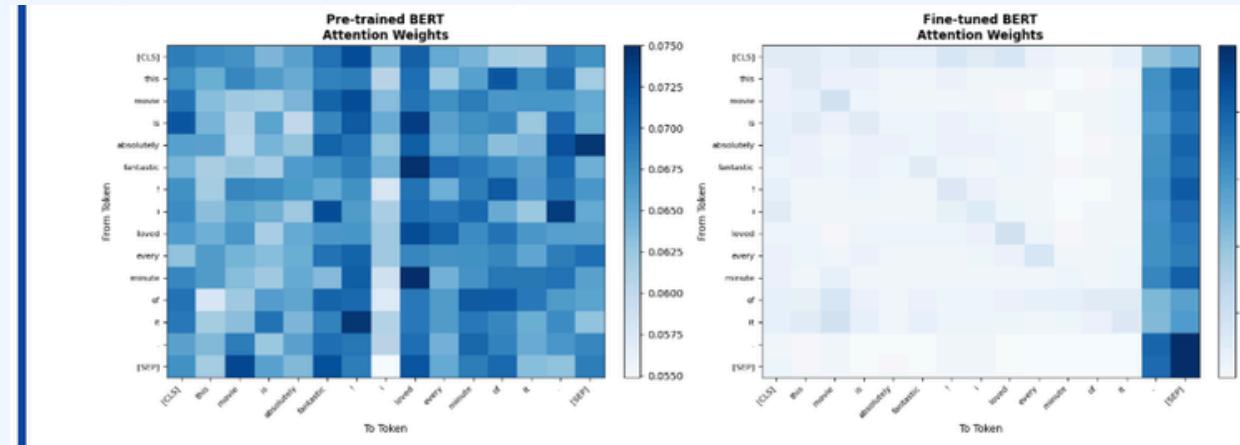
각 테스트 문장에 대해 사전 학습된 모델과 파인튜닝된 모델의 attention 히트맵을 나란히 비교하여 변화된 attention 패턴을 직관적으로 제시.

[SEP], [.]과 같은 토큰들이 Attention score가 지나치게 높게 나왔다. 적은 샘플 데이터와 높은 epoch로 인해 문장을 분리하는 패턴에 과적합 된 것으로 판단되어 해당 토큰들의 Attention Score를 계산에서 제외하였다.

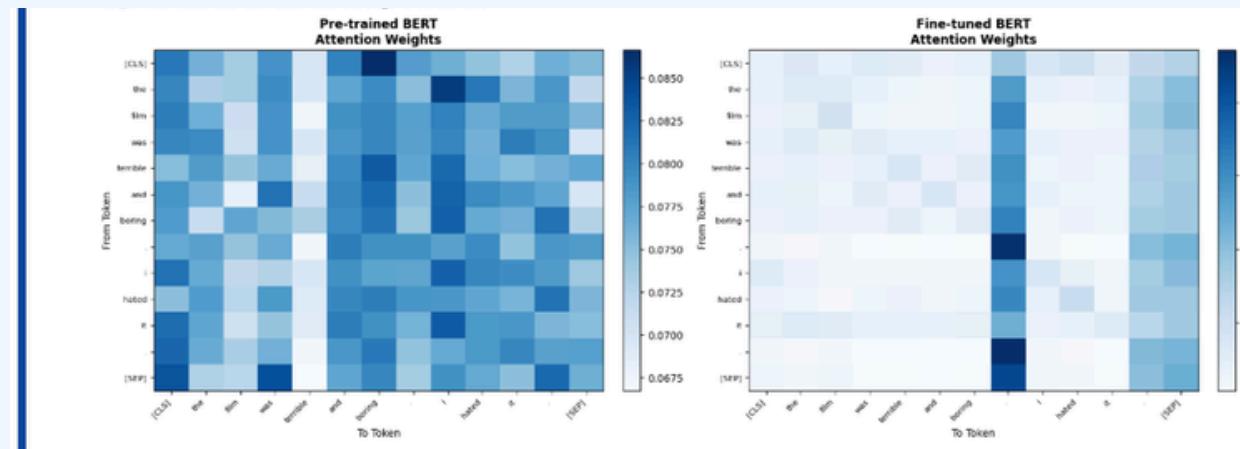
Analysis

Visualization Result

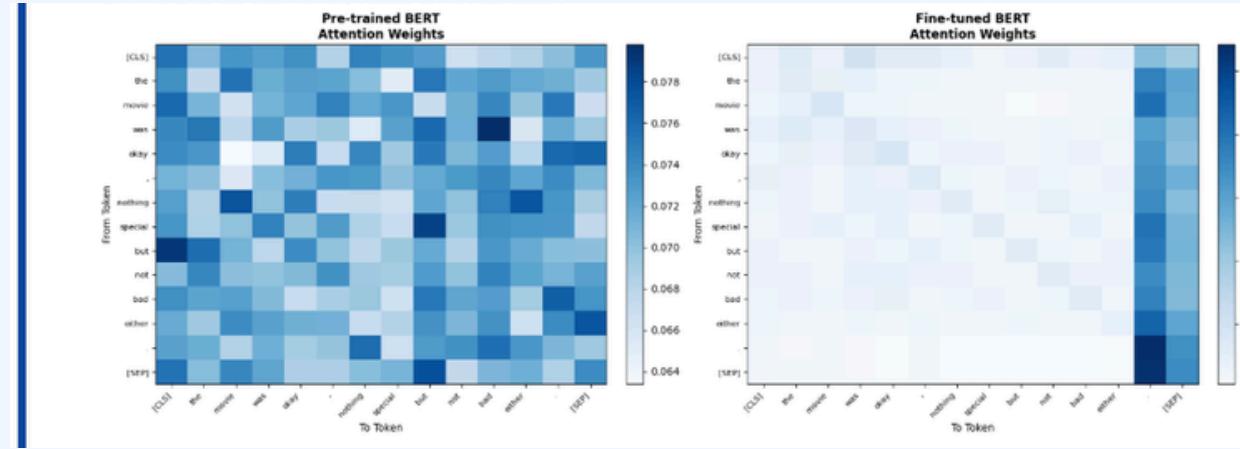
[S1]



[S2]



[S3]



[SEP], 문장 부호
토큰 제거



Experiment

> Python Implementation

```
python
1 def analyze_token_pair_attention(result, token1_idx, token2_idx):
2     """
3         특정 토큰 쌍 간의 attention 변화 분석 (비율 기준, '.'과 '[SEP]' 제외)
4     """
5     pretrained_attention = result['pretrained']['avg_attention']
6     finetuned_attention = result['finetuned']['avg_attention']
7     tokens = result['pretrained']['tokens']
8
9     # 제외 토큰이 포함된 인덱스면 분석 안 함
10    if tokens[token1_idx] in EXCLUDE_TOKENS or tokens[token2_idx] in EXCLUDE_TOKENS:
11        return None
12
13    # 전체 attention 합은 token1_idx row에서 제외 토큰을 제외하고 계산
14    def total_attention_excluding_tokens(att_matrix, tokens, row_idx):
15        valid_indices = [i for i, t in enumerate(tokens) if t not in EXCLUDE_TOKENS]
16        total = sum(att_matrix[row_idx, i] for i in valid_indices)
17        return total
18
19    if token1_idx < len(tokens) and token2_idx < len(tokens):
20        pre_total = total_attention_excluding_tokens(pretrained_attention, tokens, token1_idx)
21        fine_total = total_attention_excluding_tokens(finetuned_attention, tokens, token1_idx)
22
23        # 0 나누기 방지
24        if pre_total == 0 or fine_total == 0:
25            return None
26
27        pre_ratio = pretrained_attention[token1_idx, token2_idx] / pre_total
28        fine_ratio = finetuned_attention[token1_idx, token2_idx] / fine_total
29        change = fine_ratio - pre_ratio
30
31        print(f"토큰 쌍 분석: '{tokens[token1_idx]} → {tokens[token2_idx]}'")
32        print(f" Pre-trained 비율: {pre_ratio:.4%}")
33        print(f" Fine-tuned 비율: {fine_ratio:.4%}")
34        print(f" 변화량: {change:+.4%}p ({'증가' if change > 0 else '감소'})")
35
36        return {
37            'token1': tokens[token1_idx],
38            'token2': tokens[token2_idx],
39            'pre_ratio': pre_ratio,
40            'fine_ratio': fine_ratio,
41            'change': change
42        }
43    return None
```

BERT 모델의 토큰 쌍 간 attention 비율 변화를 분석한다. ‘.’, ‘,’와 ‘[SEP]’ 토큰은 지나치게 높은 점수를 보여 분석에서 제외한다. 불필요한 토큰을 빼고 의미 있는 변화를 집중해서 본다.

분석 대상은 문장 첫 토큰([CLS])에서 다른 토큰으로 향하는 attention 비율이다. 제외 토큰을 뺀 전체 attention 대비 비율을 사전 학습 모델과 파인튜닝 모델에서 각각 계산해 비교한다.

결과는 문장별로 [CLS] 토큰에서 가장 큰 attention 비율 증가와 감소를 출력한다. 이를 통해 파인튜닝 후 모델이 어떤 단어에 더 집중하거나 덜 집중하는지 쉽게 파악할 수 있다.

Analysis

Visualization Result

```
python
1 문장 1: This movie is absolutely fantastic! I loved every minute of it.
2 -----
3 토큰 쌍 분석: '[CLS]' → 'this'
4 Pre-trained 비율: 8.5142%
5 Fine-tuned 비율: 7.8586%
6 변화량: -0.6555%p (감소)
7 토큰 쌍 분석: '[CLS]' → 'movie'
8 Pre-trained 비율: 7.0473%
9 Fine-tuned 비율: 9.5975%
10 변화량: +2.5502%p (증가)
11 토큰 쌍 분석: '[CLS]' → 'is'
12 Pre-trained 비율: 7.4605%
13 Fine-tuned 비율: 5.7518%
14 변화량: -1.7087%p (감소)
15 토큰 쌍 분석: '[CLS]' → 'absolutely'
16 Pre-trained 비율: 8.1205%
17 Fine-tuned 비율: 3.8837%
18 변화량: -4.2368%p (감소)
19 토큰 쌍 분석: '[CLS]' → 'fantastic'
20 Pre-trained 비율: 6.8464%
21 Fine-tuned 비율: 5.4568%
22 변화량: -1.3896%p (감소)
23 토큰 쌍 분석: '[CLS]' → '!'
24 Pre-trained 비율: 8.0007%
25 Fine-tuned 비율: 15.6342%
26 변화량: +7.6334%p (증가)
27 토큰 쌍 분석: '[CLS]' → 'i'
28 Pre-trained 비율: 7.9272%
29 Fine-tuned 비율: 13.7864%
30 변화량: +5.8592%p (증가)
31 토큰 쌍 분석: '[CLS]' → 'loved'
32 Pre-trained 비율: 8.3299%
33 Fine-tuned 비율: 14.1251%
34 변화량: +5.7951%p (증가)
35 토큰 쌍 분석: '[CLS]' → 'every'
36 Pre-trained 비율: 7.6736%
37 Fine-tuned 비율: 4.2290%
38 변화량: -3.4447%p (감소)
39 토큰 쌍 분석: '[CLS]' → 'minute'
40 Pre-trained 비율: 7.0706%
41 Fine-tuned 비율: 3.1398%
42 변화량: -3.9309%p (감소)
43 토큰 쌍 분석: '[CLS]' → 'of'
44 Pre-trained 비율: 7.5283%
45 Fine-tuned 비율: 3.2539%
46 변화량: -4.2744%p (감소)
47 토큰 쌍 분석: '[CLS]' → 'it'
48 Pre-trained 비율: 7.9128%
49 Fine-tuned 비율: 5.8929%
50 변화량: -2.0199%p (감소)
```

```
python
1 문장 2: The film was terrible and boring. I hated it.
2 -----
3 토큰 쌍 분석: '[CLS]' → 'the'
4 Pre-trained 비율: 10.3732%
5 Fine-tuned 비율: 11.4353%
6 변화량: +1.0621%p (증가)
7 토큰 쌍 분석: '[CLS]' → 'film'
8 Pre-trained 비율: 9.6360%
9 Fine-tuned 비율: 10.7522%
10 변화량: +1.1161%p (증가)
11 토큰 쌍 분석: '[CLS]' → 'was'
12 Pre-trained 비율: 9.7551%
13 Fine-tuned 비율: 9.1981%
14 변화량: -0.5571%p (감소)
15 토큰 쌍 분석: '[CLS]' → 'terrible'
16 Pre-trained 비율: 9.9158%
17 Fine-tuned 비율: 7.7752%
18 변화량: -2.1406%p (감소)
19 토큰 쌍 분석: '[CLS]' → 'and'
20 Pre-trained 비율: 9.5957%
21 Fine-tuned 비율: 4.2221%
22 변화량: -5.3737%p (감소)
23 토큰 쌍 분석: '[CLS]' → 'boring'
24 Pre-trained 비율: 9.9427%
25 Fine-tuned 비율: 5.7248%
26 변화량: -4.2179%p (감소)
27 토큰 쌍 분석: '[CLS]' → 'i'
28 Pre-trained 비율: 10.7147%
29 Fine-tuned 비율: 15.5051%
30 변화량: +4.7904%p (증가)
31 토큰 쌍 분석: '[CLS]' → 'hated'
32 Pre-trained 비율: 9.9526%
33 Fine-tuned 비율: 20.4586%
34 변화량: +10.5059%p (증가)
35 토큰 쌍 분석: '[CLS]' → 'it'
36 Pre-trained 비율: 9.9241%
37 Fine-tuned 비율: 9.4623%
38 변화량: -0.4618%p (감소)
```

```
python
1 문장 3: The movie was okay, nothing special but not bad either.
2 -----
3 토큰 쌍 분석: '[CLS]' → 'the'
4 Pre-trained 비율: 9.2192%
5 Fine-tuned 비율: 14.5103%
6 변화량: +5.2911%p (증가)
7 토큰 쌍 분석: '[CLS]' → 'movie'
8 Pre-trained 비율: 8.5542%
9 Fine-tuned 비율: 11.7827%
10 변화량: +3.2285%p (증가)
11 토큰 쌍 분석: '[CLS]' → 'was'
12 Pre-trained 비율: 8.7560%
13 Fine-tuned 비율: 17.8599%
14 변화량: +9.1038%p (증가)
15 토큰 쌍 분석: '[CLS]' → 'okay'
16 Pre-trained 비율: 8.8303%
17 Fine-tuned 비율: 10.9203%
18 변화량: +2.0900%p (증가)
19 토큰 쌍 분석: '[CLS]' → 'nothing'
20 Pre-trained 비율: 9.1641%
21 Fine-tuned 비율: 5.5097%
22 변화량: -3.6544%p (감소)
23 토큰 쌍 분석: '[CLS]' → 'special'
24 Pre-trained 비율: 9.9312%
25 Fine-tuned 비율: 4.1485%
26 변화량: -5.7827%p (감소)
27 토큰 쌍 분석: '[CLS]' → 'but'
28 Pre-trained 비율: 9.8775%
29 Fine-tuned 비율: 5.3624%
30 변화량: -4.5151%p (감소)
31 토큰 쌍 분석: '[CLS]' → 'not'
32 Pre-trained 비율: 9.2271%
33 Fine-tuned 비율: 8.2189%
34 변화량: -1.0082%p (감소)
35 토큰 쌍 분석: '[CLS]' → 'bad'
36 Pre-trained 비율: 8.7337%
37 Fine-tuned 비율: 5.2565%
38 변화량: -3.4772%p (감소)
39 토큰 쌍 분석: '[CLS]' → 'either'
40 Pre-trained 비율: 8.8664%
41 Fine-tuned 비율: 8.9071%
42 변화량: +0.0407%p (증가)
```

Experiment

➤ Python Implementation

```
python
1 def calculate_overall_attention_distribution(attention_matrix, tokens):
2     """Calculate overall attention distribution excluding special tokens"""
3     valid_indices = get_valid_token_indices(tokens)
4
5     # Total attention each token receives (column sum)
6     incoming_attention = np.zeros(len(valid_indices))
7     # Total attention each token gives (row sum)
8     outgoing_attention = np.zeros(len(valid_indices))
9
10    for i, valid_i in enumerate(valid_indices):
11        for j, valid_j in enumerate(valid_indices):
12            incoming_attention[i] += attention_matrix[valid_j, valid_i]
13            outgoing_attention[i] += attention_matrix[valid_i, valid_j]
14
15    # Normalize
16    total_incoming = incoming_attention.sum()
17    total_outgoing = outgoing_attention.sum()
18
19    if total_incoming > 0:
20        incoming_attention = incoming_attention / total_incoming
21    if total_outgoing > 0:
22        outgoing_attention = outgoing_attention / total_outgoing
23
24    return incoming_attention, outgoing_attention, [tokens[i] for i in valid_indices]
```

특수 토큰을 제외한 토큰들의 incoming과 outgoing attention을 계산하고, 전체 합으로 정규화해 **토큰별 attention 분포**를 반환.

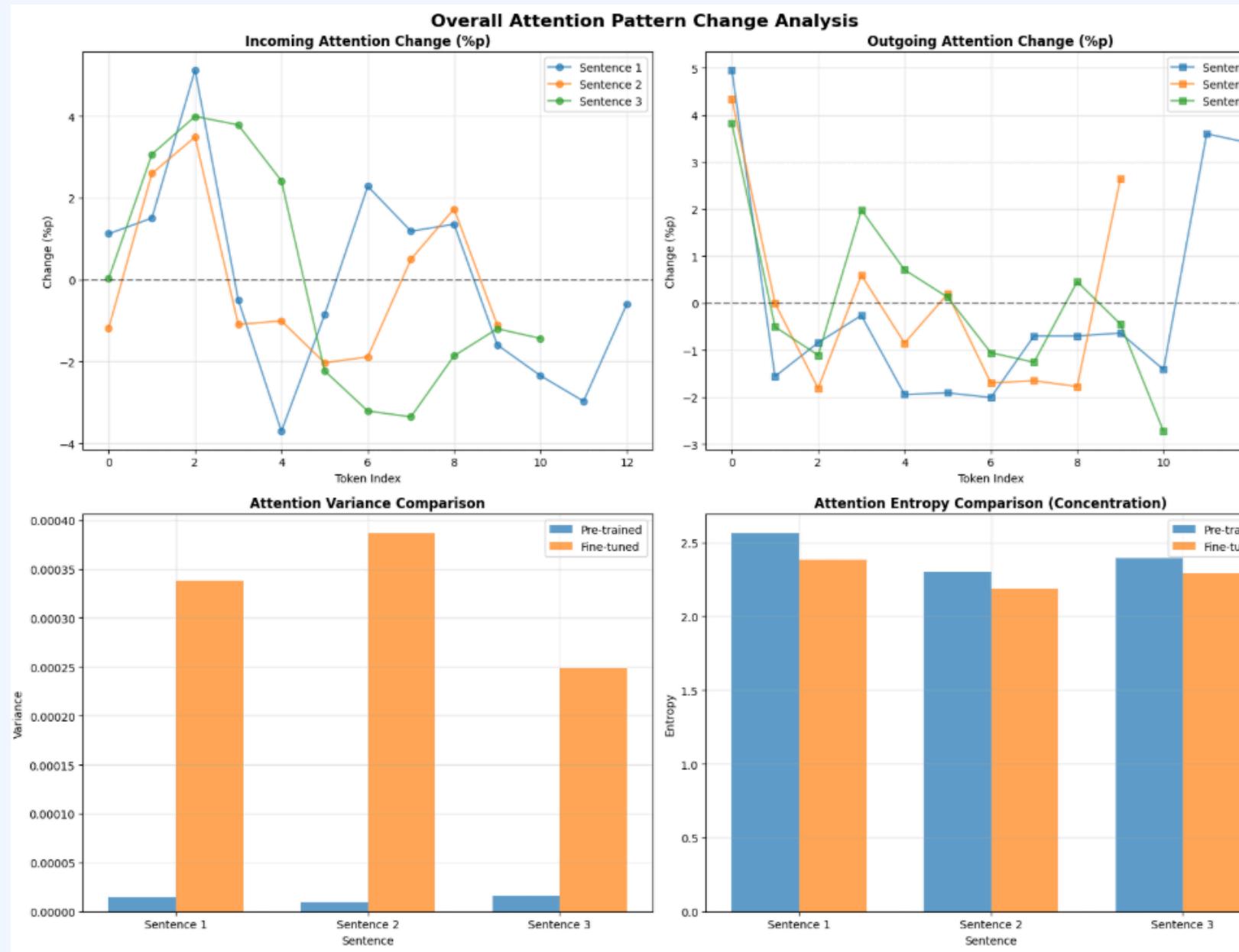
➤ Python Implementation

```
python
1 def extract_all_token_attention_changes(comparison_results):
2     """Extract attention changes for all tokens (treating all as emotion expression words)"""
3     all_changes = []
4
5     for i, result in enumerate(comparison_results):
6         sentence = result['sentence']
7         pre_attention = result['pretrained']['avg_attention']
8         fine_attention = result['finetuned']['avg_attention']
9         tokens = result['pretrained']['tokens']
10
11        # Calculate normalized attention
12        pre_ratio, pre_tokens = normalize_attention_to_ratio(pre_attention, tokens)
13        fine_ratio, fine_tokens = normalize_attention_to_ratio(fine_attention, tokens)
14
15        # Find all tokens (treating all as emotion expression words)
16        for j, token in enumerate(pre_tokens):
17            if j < len(fine_tokens):
18                # Attention from [CLS] token to this token
19                if len(pre_ratio) > 0 and len(fine_ratio) > 0:
20                    pre_cls_attention = pre_ratio[0, j] if pre_ratio.shape[0] > 0 else 0
21                    fine_cls_attention = fine_ratio[0, j] if fine_ratio.shape[0] > 0 else 0
22                    change_pp = (fine_cls_attention - pre_cls_attention) * 100 # %p change
23
24                all_changes.append({
25                    'sentence_idx': i,
26                    'sentence': sentence,
27                    'token': token,
28                    'pre_attention': pre_cls_attention,
29                    'fine_attention': fine_cls_attention,
30                    'change_pp': change_pp
31                })
32
33    return all_changes
34
```

사전학습과 파인튜닝 모델 간 [CLS] 토큰에서 각 토큰으로 향하는 attention 변화량을 계산해, **단어별 attention 증감 추이**를 리스트로 반환

Analysis

Visualization Result

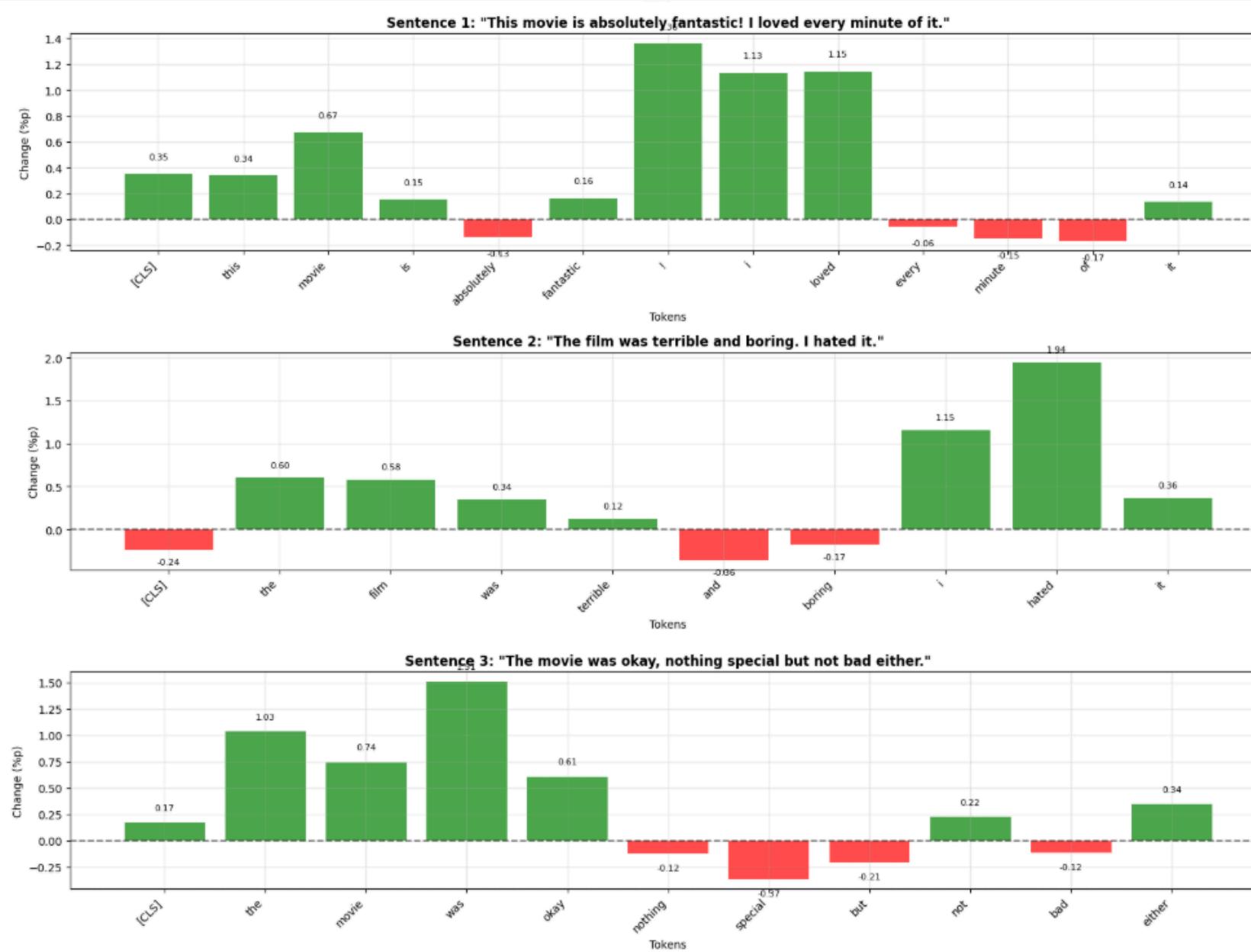


Fine-tuning이 BERT 어텐션에 미치는 영향

- **Incoming attention:** 문장 초반 토큰에 대한 주의가 증가 → 문장 시작에 더 집중.
- **Outgoing attention:** 특정 토큰이 더 많은 정보를 전달하도록 변화 → 정보 흐름의 재구성.
- Fine-tuning 후 **어텐션 분산(variance)**이 전반적으로 증가 → 어텐션 패턴이 더 다양하고 복잡해짐.
- **어텐션 엔트로피** 감소 → 특정 토큰에 대한 집중이 강화됨

Analysis

Visualization Result



감정 유형별 어텐션 변화

-긍정 문장

"fantastic", "loved", 등의 핵심 감정 표현 및 관련 단어들의 어텐션 집중 증가.

*"absolutely"*와 같은 강조 부사는 상대적으로 감소 → 핵심 감정어에 더 집중하도록 학습됨.

-부정 문장

"hated", "I" 등의 주관적 표현에 어텐션 급증.

기능어 "the", "was"의 어텐션 감소 → 의미 중심의 선택적 주의 강화.

-중립 문장

"okay"에 가장 큰 어텐션 증가 → 중립 감정의 핵심 표현으로 학습.

"special", "but" 등 주변 맥락 단어에 대한 어텐션 감소 → 감정 핵심어에 더 집중.

Experiment

> Python Implementation

```
python
1 def analyze_token_pair_attention(result, token1_idx, token2_idx):
2     """
3         특정 토큰 쌍 간의 attention 변화 분석 (비율 기준, '.'과 '[SEP]' 제외)
4     """
5     pretrained_attention = result['pretrained']['avg_attention']
6     finetuned_attention = result['finetuned']['avg_attention']
7     tokens = result['pretrained']['tokens']
8
9     # 제외 토큰이 포함된 인덱스면 분석 안 함
10    if tokens[token1_idx] in EXCLUDE_TOKENS or tokens[token2_idx] in EXCLUDE_TOKENS:
11        return None
12
13    # 전체 attention 합은 token1_idx row에서 제외 토큰을 제외하고 계산
14    def total_attention_excluding_tokens(att_matrix, tokens, row_idx):
15        valid_indices = [i for i, t in enumerate(tokens) if t not in EXCLUDE_TOKENS]
16        total = sum(att_matrix[row_idx, i] for i in valid_indices)
17        return total
18
19    if token1_idx < len(tokens) and token2_idx < len(tokens):
20        pre_total = total_attention_excluding_tokens(pretrained_attention, tokens, token1_idx)
21        fine_total = total_attention_excluding_tokens(finetuned_attention, tokens, token1_idx)
22
23        # 0 나누기 방지
24        if pre_total == 0 or fine_total == 0:
25            return None
26
27        pre_ratio = pretrained_attention[token1_idx, token2_idx] / pre_total
28        fine_ratio = finetuned_attention[token1_idx, token2_idx] / fine_total
29        change = fine_ratio - pre_ratio
30
31        print(f"토큰 쌍 분석: '{tokens[token1_idx]} → {tokens[token2_idx]}'")
32        print(f" Pre-trained 비율: {pre_ratio:.4%}")
33        print(f" Fine-tuned 비율: {fine_ratio:.4%}")
34        print(f" 변화량: {change:+.4%}p ({'증가' if change > 0 else '감소'})")
35
36        return {
37            'token1': tokens[token1_idx],
38            'token2': tokens[token2_idx],
39            'pre_ratio': pre_ratio,
40            'fine_ratio': fine_ratio,
41            'change': change
42        }
43
44    return None
```

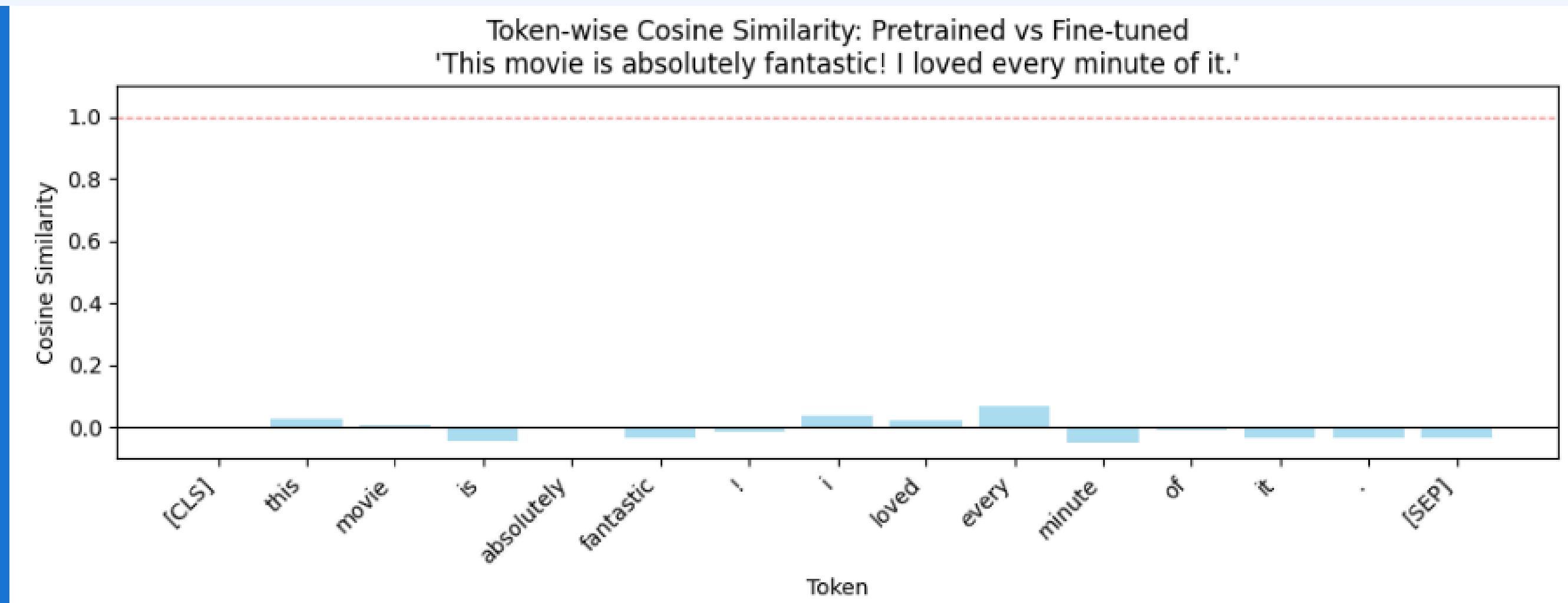
두 개의 언어 모델에서 주어진 문장에 대한 토큰 임베딩(벡터)을 추출하고, 토큰별 임베딩 간 코사인 유사도를 계산하여 두 모델의 임베딩 차이를 시각화하고 분석

코사인 유사도는 두 벡터가 얼마나 비슷한지를 1에 가까울수록 유사하다고 판단하는 지표로, 1은 완전 동일, 0은 완전 무관함을 의미

유사도가 0.9 미만인 토큰은 “크게 변화한 토큰”으로 분류해 별도로 출력하여 두 모델이 임베딩 공간에서 어떻게 차이를 보이는지 직관적으로 파악

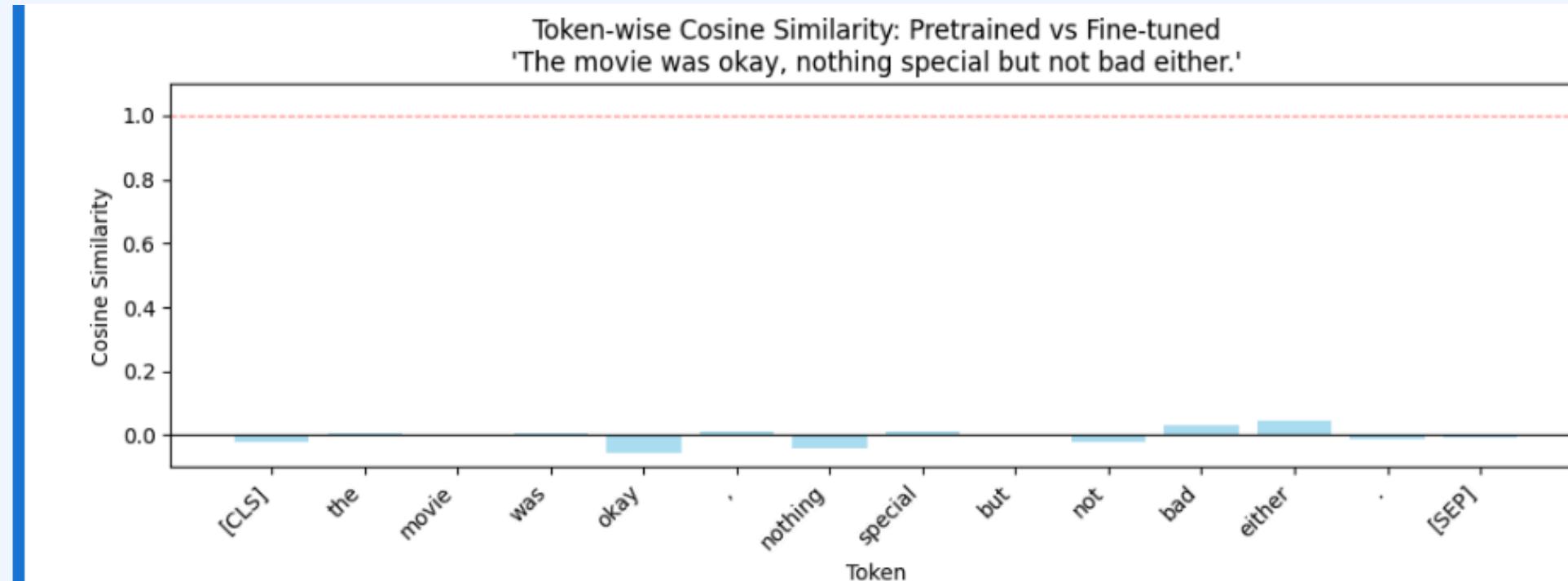
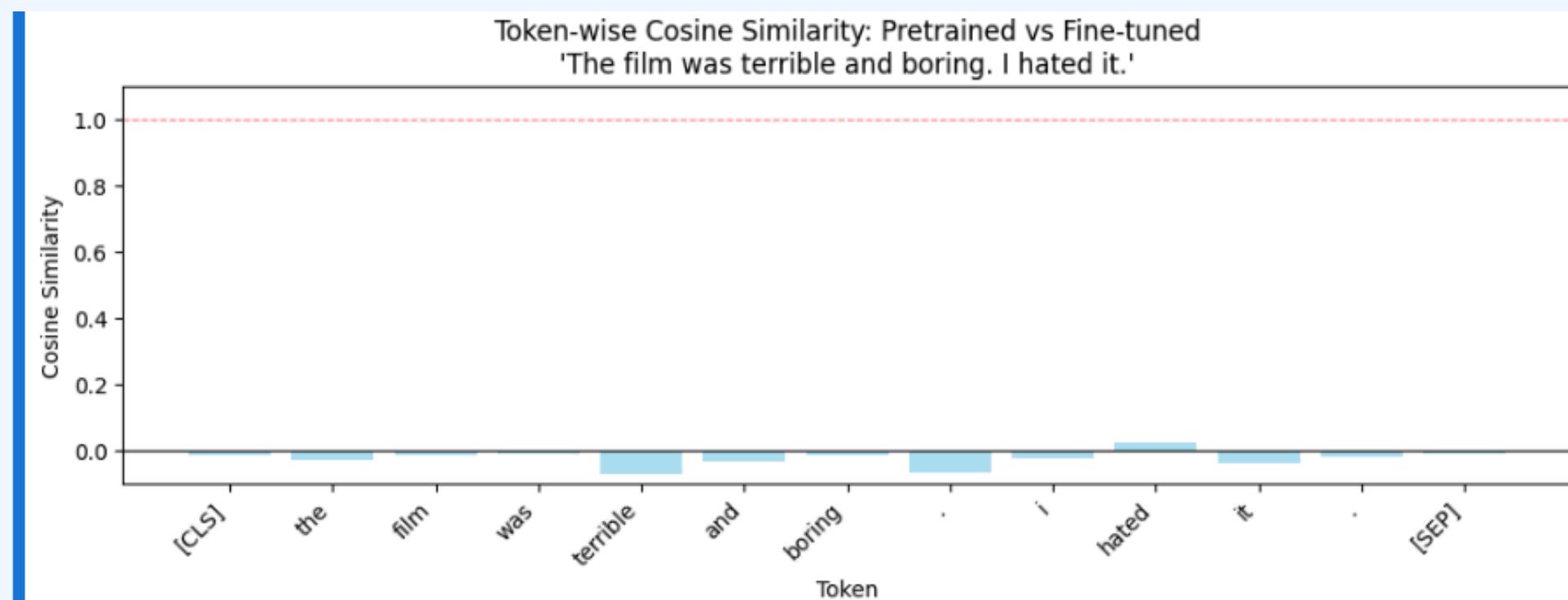
Analysis

Visualization Result



Analysis

Visualization Result



Conclusions

BERT : Architecture, Fine-tuning, and Attention Visualization Method

- Conclusions:
 - BERT 모델의 fine-tuning 과정에서 attention 메커니즘 변화 관찰
→ 감정 분류 task에 맞춰 핵심 감정 토큰에 집중
 - Fine-tuning을 통해 모델의 해석가능성 증대
→ attention 시각화를 통해 변화 직관적 확인 가능
 - 다양한 시각화 도구의 보완적 활용 필요
- Future Direction:
 - 다양한 NLP task별 attention 변화 패턴 비교 분석
 - Attention의 기여도 정량화
→ 성능 향상과의 직접적 연관성 평가
 - Attention 패턴 기반
→ 모델 편향성 탐지 및 학습 품질 평가 방법론 개발

References

- [1]. Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers). 2019.
- [2]. Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).
- [3]. Kong, Aobo, et al. "Better zero-shot reasoning with role-play prompting." arXiv preprint arXiv:2308.07702 (2023)
- [4]. Wei, Jason, et al. "Chain-of-thought prompting elicits reasoning in large language models." Advances in neural information processing systems 35 (2022): 24824-24837.
- [5]. White, Jules, et al. "A prompt pattern catalog to enhance prompt engineering with chatgpt." arXiv preprint arXiv:2302.11382 (2023).
- [6]. Vig, Jesse. "A multiscale visualization of attention in the transformer model." arXiv preprint arXiv:1906.05714 (2019).

github, perplexity



Perplexity

Perplexity is a free AI-powered answer engine that provides accurate, trusted, and real-time answers to any question.

 Perplexity AI