프로젝트 중간 보고서

프로젝트명 : GPT-2 구축하기

교과목명	자연어처리개론
담당교수	김광일
팀 원	안하현
	이지현
	전희연

Ⅰ. 프로젝트 개요

1. 연구 배경

최근 자연어처리(NLP) 분야에서는 대규모 사전학습 언어모델(pretrained language models)이 다양한 다운스트림 태스크에서 뛰어난 성능을 보이며 주목받고 있다. 특히 OpenAI의 GPT-2는 텍스트 생성 능력과 문맥 이해력이 결합된 대표적인 decoder-only 구조 언어모델로, 언어 생성뿐 아니라 문장 분류, 의미 판별, 창작 등 다양한 과제에 활용 가능하다.

그러나 GPT-2와 같은 대형 모델을 실제 태스크에 활용하기 위해서는 fine-tuning이 필수적이며, 이 과정에서 모델 파라미터 수, 연산 자원, 태스크 적합성 등의 문제가 발생한다. 이에 따라 최근에는 파라미터 효율적(parameter-efficient)인 fine-tuning 전략이 활발히 연구되고 있으며, 본 프로젝트는 그러한 전략들이 실제 태스크에서 얼마나 효과적으로 성능을 향상시킬 수 있는지를 실험적으로 검증하고자 한다.

2. 연구 목적

본 프로젝트의 목적은 GPT-2 모델을 직접 구현하고, 이를 세 가지 대표적인 자연어처리 태스크(감정 분석, 시 생성, 패러프레이즈 탐지)에 적용하여, 각각의 태스크에 적합한 최신 성능 향상 전략을 실험하고 분석하는 것이다.

각 태스크에 대해 다음을 수행한다:

- 베이스라인 모델을 설정하고, GPT-2를 태스크에 맞게 fine tuning
- 성능 향상을 위한 다양한 방법 도입
- 성능 향상 정도를 정량적으로 비교 평가

Ⅱ. 프로젝트 진행 결과

1. GPT-2 구축하기

Multi-head self-attention, Transformer block, Token Embedding, Positional Encoding, Layer Normalization, Residual Connection, AdamW Optimizer 등 주요 모듈 구현 완료

2. 감정 분석 (Sentiment Analysis)

IMDB 데이터셋은 사전학습된 GPT-2 모델만으로도 검증 정확도(Dev Accuracy) 90% 이상의 매우 높은 성능을 보여주었다. 이는 해당 데이터셋의 문장 구조가 단순하거나 감정 표현이 명확하여, 별도의 Fine-Tuning을 적용하지 않고도 높은 예측 정확도를 나타낸 것으로 분석된

다. 따라서 IMDB 데이터셋은 Fine-Tuning 기법의 성능 향상 효과를 정량적으로 비교·평가하기에 적절하지 않다고 판단하였으며, 본 프로젝트에서는 SST 데이터셋을 중심으로 실험을 진행하였다. 현재 베이스라인 모델 구축 완료되었으며 베이스라인 모델의 실험 세팅은 다음과 같다.

- 데이터셋
  - Stanford Sentiment Treebank (SST-5)
- 입력 포맷
  - `"Review: {문장}" → "Sentiment:"` 형태의 프롬프트 기반 입력을 사용
- 출력 포맷
  - 총 5개 감정 클래스 중 하나 (0: 매우 부정 ~ 4: 매우 긍정)
- 학습 방식
  - Supervised learning
  - 전체 모델 파라미터를 업데이트 (full fine-tuning)
- 베이스라인 모델 구성

  본 프로젝트에서 설정한 감정 분석의 베이스라인 모델은 다음과 같다.
  - Hugging Face의 `GPT2ForSequenceClassification` 구조를 기반으로 구성
  - 입력 문장의 마지막 토큰에 해당하는 hidden state를 추출하여 linear classification head에 연결
  - Softmax 함수를 통해 5개 감정 클래스에 대한 확률 분포를 출력
  - 손실 함수로는 CrossEntropyLoss를 사용하며, AdamW optimizer로 학습
- 하이퍼파라미터 설정 (Baseline 기준)
  - Learning rate: 2e-5
  - Batch size: 64
  - Epochs: 6
  - Optimizer: AdamW
  - Dropout: 0.3
- 베이스라인 실험 결과 (SST 기준)

Epoch	Train Acc	Train F1	Dev Acc	Dev F1
0	31.40%	15.56%	29.52%	14.53%
1	49.20%	41.09%	45.69%	37.68%
2	56.30%	50.75%	48.41%	42.80%
3	59.94%	57.04%	50.05%	47.51%
4	**63.09%**	**59.99%**	**51.77%**	**48.98%**
5	64.77%	61.86%	49.23%	46.38%

- 분석 결과
  - Epoch 4 기준 Dev Set에서 최고 성능: Accuracy: 51.77%, F1 Score: 48.98%
  - Epoch 5에서는 과적합(overfitting) 현상이 일부 나타남.
  - Epoch를 10까지 늘려보았으나 점점 Dev Acc가 낮아지는 현상 발생. 즉, 베이스라인에서는 Epoch을 더이상 늘리지 않는 게 낫다고 판단.

3. 패러프레이즈 탐지 (Paraphrase Detection)

본 프로젝트의 초기 실험 계획은 세 가지 방법을 중심으로 구성되었다. 첫 번째로 고려한 방법은 LoRA로, 이는 Self-Attention 모듈 내 가중치를 low-rank 행렬로 분해하여 학습하는 기법이다. 기존 가중치는 고정된 상태에서 일부 저차원 행렬만 학습하기 때문에 파라미터 효율성이 높고, 학습 속도가 빠르며 GPU 메모리 사용량이 적다는 장점을 갖는다. 두 번째 방법은 LLRD (Layer-wise Learning Rate Decay)로, Transformer의 각 층에 대해 학습률을 점진적으로 감소시키는 전략이다. 이를 통해 상위층은 빠르게 학습하고 하위층은 일반적인 표현을 유지하도록 조절하여, fine-tuning의 안정성을 높이고 과적합을 방지하는 효과가 있다. 마지막으로 세 번째는 LoRA와 LLRD의 결합 전략으로, LoRA를 통해 파라미터 수를 줄이는 동시에, LLRD를 활용하여 각 층별 학습률을 세밀하게 조절함으로써 paraphrase detection에서의 최적 성능을 달성하고자 하였다.

그러나 이 세 가지 방법에는 몇 가지 구조적 한계점이 존재함을 확인하였다. 먼저, LLRD는 GPT-2 구조에서 그 효과가 검증되지 않았다는 점이 문제였다. LLRD는 주로 BERT와 같은 encoder-only 모델에서 실험적으로 확인된 방식으로, decoder-only 구조인 GPT-2에는 직접적인 적용 근거가 부족하다. 둘째, LoRA의 성능 향상은 주로 생성(generation) 태스크를 대상으로 한 실험에 기반하고 있어, 본 프로젝트의 목표인 분류(classification) 태스크에 그대로 일반화하기엔 무리가 있다. 마지막으로, LoRA와 LLRD의 결합에는 구조적 충돌 가능성이 존재한다. LoRA는 일부 모듈만 학습하고 나머지는 고정(freeze)하는 방식인 반면, LLRD는 모든 파라미터를 학습 대상으로 삼고 학습률만 차등 적용한다. 이로 인해 freeze된 모듈에는 LLRD가 적용되지 않으므로, 두 방식의 단순한 결합은 실질적인 성능 향상을 이끌어내기 어려울 수 있다.

따라서 새로운 방법을 제시한다.

- 성능향상 전략

단계	기법	핵심 아이디어	기대 효과
1	Prompt Tuning[1)]	자연어 프롬프트 포맷을 개선하여 모델이 태스크(Paraphrase Detection)를 명확히 이해하도록 유도	사전학습된 GPT-2의 언어 이해 능력을 효과적으로 활용, 추가 파라미터 없이 정확도 향상 기대
2	[2)]LoRA	GPT-2의 FFN 계층에 저랭크(low-rank) 행렬을 삽입하여 파라미터 수를 줄이고, LoRA 파라미터만 학습	파라미터 효율성 증가, 낮은 GPU 요구량, 빠른 학습 및 파인튜닝 가능
3	Hybrid Fine-Tuning (Prompt Tuning + LoRA)	프롬프트 튜닝 기반 입력 + LoRA 구조를 결합하여 task 이해력과 학습 효율을 동시에 강화	프롬프트 기반 명확한 태스크 정의 + 파라미터 효율성 조합 → paraphrase detection에서 최대 성능 확보

● 이론적 근거

◦ Prompt Tuning

- 자연어 프롬프트 문장을 활용해 모델에게 태스크 목적을 명시적으로 알려주는 방식
- Radford et al. (2019)에서 제안된 “task as text” 프레임워크 기반
- 별도의 파라미터 없이도 task-specific fine-tuning이 가능하며, zero-shot/few-shot 학습 성능 향상 확인됨
- 예: `"Sentence1: {문장1}, Sentence2: {문장2} → Are they paraphrases?"`

◦ LoRA (Low-Rank Adaptation)

- Attention 또는 FFN weight matrix를 $W \approx W + A \cdot B$ 형태로 분해해 학습
- 기존 가중치 `W`는 고정, 작은 크기의 `A`와 `B`만 학습 → 전체 파라미터 수 대폭 감소
- 전체 fine-tuning에 비해 최대 10,000배 적은 파라미터 수로도 유사하거나 더 나은 성능 달성
- GPT-2 기반 NLG 실험(WebNLG, DART 등)에서 BLEU, METEOR, ROUGE 지표가 향상됨

◦ Hybrid Fine-Tuning

- Prompt Tuning을 통해 task-specific 질의를 자연어로 입력하여 인식력 강화
- LoRA 구조를 결합해 효율적 파라미터 업데이트 가능
- 프롬프트로 태스크를 명시하고, LoRA로 미세 조정 효율을 극대화하는 접근
- 최근 연구 다수의 언어 태스크에서 성능 및 메모리 효율을 동시에 확보한 사례 확인됨

● 실험 계획

◦ 데이터 전처리 및 베이스라인 구축

- Quora Paraphrase Detection 데이터셋 사용
- 전처리 후 다음과 같은 프롬프트 형태로 입력 구성: `"Sentence1: {문장1}, Sentence2: {문장2} → Are they paraphrases?"`
- GPT2ForSequenceClassification 구조 기반의 베이스라인 구현 및 학습 수행

◦ 실험 A: Prompt Tuning 적용

- `datasets.py`에서 입력 포맷을 자연어 프롬프트 형태로 변경
- 기존 구조와 파라미터 동일하게 유지하며 성능(Accuracy, F1)을 비교
- 입력 포맷 변화만으로 성능이 얼마나 향상되는지 분석

◦ 실험 B: LoRA 적용

- GPT-2 FFN 계층 또는 classification head 이전 layer에 LoRA 삽입 (`gpt2_layer.py`)
- 전체 파라미터 중 LoRA 모듈만 학습하도록 optimizer 설정
- 파라미터 수, 학습 속도, 성능(Accuracy, F1)을 full fine-tuning과 비교

◦ 실험 C: Prompt Tuning + LoRA 결합

- 프롬프트 입력 포맷(`datasets.py`)과 LoRA 구조(`gpt2_layer.py`)를 함께 적용
- 모델 입력 인식력 + 파라미터 효율성을 동시에 확보

---

1) Xiao Liu et al., P-Tuning v2: Prompt Tuning Can Be Comparable to Fine-tuning Universally Across Scales and Tasks, arXiv preprint arXiv:2110.07602, 2021.

2) Hu, Edward J., et al. "LoRA: Low-rank adaptation of large language models." ICLR (2022)

- 최종 성능, 학습 안정성, GPU 메모리 사용량 등을 종합적으로 비교 평가

Ⅲ. 향후 계획

1. 감정 분석 (Sentiment Analysis)

다음과 같은 Task들의 성능을 베이스라인과 비교하여 가장 최적의 Task를 고르고자 한다. 현재 모든 Task들의 코드 구현은 완성시켰으나 학과 GPU 서버 발급 대기 중이라 서버를 발급받고 나서 다음 Task들을 실행시키고자 한다.

- ULMFiT 전략 적용 실험 (Task A)
- LoRA 적용 실험 (Task B)
- Hybrid Fine-Tuning (Task C)

2. 패러프레이즈 탐지 (Paraphrase Detection)

현재 베이스라인 및 Task A, Task B, Task C에 대한 코드를 모두 구현한 상태이며, 우선 베이스라인의 성능을 확인한 뒤 각 태스크의 성능과 비교하여 가장 최적의 태스크를 선정할 예정이다.

- Prompt Tuning 적용 실험 (Task A)
- LoRA 적용 실험 (Task B)
- Hybrid Fine-Tuning (Prompt Tuning + LoRA) 적용 실험 (Task C)

3. 시 생성 (Poem Generation)

베이스라인 및 Task A, Task B, Task C에 대한 코드를 구현한 후, 베이스라인의 성능을 확인한 뒤 각 태스크의 성능과 비교하여 가장 최적의 태스크를 선정할 예정이다.

- Unlikelihood Loss 적용 실험 (Task A)
- Prefix-Tuning 적용 실험 (Task B)
- Hybrid Fine-Tuning 적용 실험 (Task C)