자연어처리개론 기말프로젝트 최종보고서

과 목 : 자연어처리개론_01

담당교수 : 김광일 교수님

학 과 : 컴퓨터공학전공*, 멀티미디어공학전공**

이 름 : 이재원*, 강병민**, 이소원**

제 출 일 : 2025.06.16.

dongguk UNIVERSITY

1. 서론

Large-scale Pretrained LM(GPT)의 개발은 NLP 연구 및 실무 전체 분야에 걸쳐 패러다임의 변화를 야기하였다. 특히 GPT-2(2019)는 단순한 decoder-only structure와 Large-scale Pretraining만을 사용하여 Translation, QA, Summary 등 여러 downstream task에서 의미 있는 성능을 보였고, 이는 후속 GPT 시리즈 모델의 기반을 마련하였다.

이 프로젝트는 GPT-2를 직접 구현하고, 완성된 모델을 3가지 downstream task에 대해 작동하도록 하며, 최종 성능을 개선하는 것을 목표로 한다. GPT-2 117M open parameter weight를 자체 아키텍처에 불러와, 3가지 downstream task에 대한 fine-tuning을 수행한다.

Sentimental Analysis – SST, cfimdb corpus를 이용하여 Generative LM을 fine-tuning하고, classifier model로 확장한다.

Paraphrase Detection – Quora Question Pairs (QQP) dataset을 이용해, 두 문장이 paraphrase인지 판단하기 위해 "yes/no"를 생성하도록 cloze-style의 task로 바꾸어 fine-tuning을 진행한다.

Sonnet Generation – Shakespeare Sonnet을 이용해, multi-token generation task로 fine-tuning한다.

본 프로젝트의 주요 목적은 단일 Pretrained LM인 GPT-2가 감성 분석, 패러프레이즈 감지, 소네트 생성 등 구조적으로 이질적인 다수의 downstream task 대해 fine-tuning을 통해 얼마나 효과적으로 학습될 수 있는지를 검증하는 것이다. 이를 위해 모델 내부의 self-attention 메커니즘과 Adam 최적화 알고리즘을 직접 구현하였으며, 각 task별 데이터 전처리 및 파라미터 설정을 포함하는 fine-tuning pipeline을 설계·구축하였다. 학습된 모델의 성능은 정확도(accuracy), F1 score, BLEU, rhyme density, CHRF 등 다양한 정량적 지표를 활용하여 다각도로 평가하였다.

본 논문의 구성은 다음과 같다. 제2장에서는 GPT-2 아키텍처 구현 및 감성 분석(Sentimental Analysis) 실험에 대해 기술한다. 제3장에서는 Quora Question Pairs 데이터셋을 활용한 패러프레이즈 감지(Paraphrase Detection)의 구현, 여러 method의 적용 및 pipeline 작성, 그리고 실험 및 결과를 기술한다. 제4장에서는 Shakespeare Sonnet을 이용한 Sonnet Generation task에 대해 여러 method의 적용 및 pipeline 작성, 실험 및 평가 결과를 제시한다. 제5장에서는 본 연구의 결과를 종합적으로 논의하고 연구의 한계 및 향후 연구 방향을 제안한다.

2. GPT-2 아키텍처 구현 및 Sentimental Analysis

본 프로젝트에선 OpenAI GPT-2(117 M) 모델의 모듈을 PyTorch 환경에서 구현하여, 감정 분석 Dataset(SST-2/CF-IMDB)을 이용한 fine-tuning 확인하였다. 구현된 네트워크는 Decoder-only Transformer 이다.

2.1. 모듈별 구현 세부

입력 문장은 word embedding 과 positional embedding 을 합산한 뒤 dropout 을 거쳐 GPT-2 encoder 에 투입된다. 각 GPT-2 층은 'LayerNorm → Causal Self-Attention(하향 삼각+패딩 마스크) → Residual/Dropout → LayerNorm → GELU 기반 Feed-Forward → Residual/Dropout(Pre-LayerNorm) 구조를 따른다. 모든 층을 통과한 후 모델 맨 끝에서 한 번 더 LayerNorm 을 적용한다. LM 모드에서는 입력 임베딩 가중치와 weight-tying 을 통해 token/logit 을 계산하지만, sentimental analysis 에서는 별도의 Linear Classifier Head 를 사용한다.

2.2. AdamW Optimizer

학습 안정성을 높이기 위해 AdamW 알고리즘을 직접 구현하였다. 1·2 차 moment 누적, bias-correction, decoupled weight-decay 로직을 모두 포함하며, 첫 step 에서 state tensor 를 초기화한 뒤 매 step 마다 파라미터를 갱신한다.

2.3. Sentiment Classification 헤드 및 fine-tuning 전략

감정 분류를 위해 GPT-2 backbone 위에 dropout(0.3)–linear classifier 로 이루어진 최소 head 를 추가해 GPT2SentimentClassifier 를 정의하였다. 실험은 (a) backbone 을 freeze 하고 head 만 학습하는 last-linear-layer 모드, (b) 모든 파라미터를 미세조정하는 full-model 모드로 수행되었다. 선택된 모드는 실행 인자로 제어되며, 학습 루프에서 CrossEntropyLoss 를 사용해 역전파한다.

2.4. 데이터셋·전처리·학습 설정

훈련에는 SST-5(5-class, ≈11 k sentences)와 CF-IMDB(binary, 25 k sentences)를 사용하였다. 각 문장은 GPT-2 tokenizer 로 최대 128 token 까지 truncate·padding 처리하고, padding 위치를 나타내는 attention_mask 를 동시에 생성하였다. batch-size 는 기본적으로 두 데이터셋 모두 8 로 설정했다. learning-rate 는 1×10^{-3}, epoch 10, weight-decay 0 으로 고정하였다. 학습 과정에서 dev set 의 Accuracy 와 Macro-F1 을 매 epoch 모니터링하고, 최고 dev accuracy 시점의 가중치를 checkpoint 로 저장하였다.

2.5. 사전 결과 요약

실험 결과, full-model fine-tuning 이 last-linear-layer 대비 모든 데이터셋에서 일관된 Macro-F1 향상을 보였다. 또한 자체 구현 AdamW 가 공식 구현과 동일한 성능을 재현해, 향후 실험에서 optimizer 자체가 변수로 작용하지 않는 신뢰 기반을 마련하였다.

3. Paraphrase Detection

3.1. Experiments Setting

Parapharse detection task 에서의 모든 실험은 아래와 같은 환경에서 진행되었다.
- GPU 는 A6000 48gb x 1 을 사용하였다.
- 제공된 quora_train.csv 를 통해 학습을 진행, quora_dev.csv 를 통해 validation/test 를 진행했다.
- 사용한 모델의 크기는 모두 GPT2 기본 모델을 사용하였다.
- 많은 실험에서 기본적으로 batch_size=96 으로 설정하였다. 다른 크기를 사용하였다면 기재했다.
- learning rate 는 실험에 따라 다르게 설정하였으나, 기본적으로 2e-5 를 사용하였다.
- seed 는 재현성을 위해 모두 11711 로 설정하였다.
- 모든 실험에서, 메모리 사용 및 속도 개선을 위해 mixed-precision 을 사용하였다.
(FP16, 16-mixed).
- A6000 48gb GPU 를 사용하였으므로, `torch.set_float32_matmul_precision('high')` 세팅을 활성화한 실험이 많다. 다른 환경에서는 이 코드를 주석처리 할 것을 권한다.
- 많은 실험에서 epoch 당 15-30 분 이내의 시간이 소요되었다.

3.2. Experiments

3.2.1. Baseline

`_250523_default_paraphrase.py` / `_250613_default_BackTranslation.py`

- 기본적으로 구현된, full-finetuning 방식이였으며, baseline 성능이 되어주었다.

- `_250613_default_BackTranslation.py`는 BackTranslation method를 위해 작성하였으나, 기본 baseline 코드와 완전히 동일한 로직을 통해 data augmentation만의 성능을 확인하는 ablation study가 가능하도록 구현하였으므로, baseline으로 채택한다.

3.2.2. SimCSE

`_250522_SimCSE.py` / `_250523_simcse_etc.py`

'SimCSE: Simple Contrastive Learning of Sentence Embeddings' 논문의 Method인 SimCSE(Gao et al., 2021)를 구현하여 적용하였다.

`_250523_simcse_etc_paraphrase.py`에서는 R-Drop / Label Smoothing + scheduler + gradient checkpointing 등의 기법 또한 추가로 구현하였다.

자세한 구현체는 코드 참고.

3.2.3. Adverserial paraphrasing

`_250611_Adverserial_gem.py`

'Improving Paraphrase Detection with the Adversarial Paraphrasing Task' 논문의 Method인 Adverserial Paraphrasing을 구현하여 적용하였다. 자세한 구현체는 코드 참고.

3.2.4. Multitask Learning

`_250612_Multitask.py` / `_250615_multitask2.py`

'End-to-End Multi-Task Learning with Attention' 논문의 Multitask Learning(Liu et al., 2019) 방식을 여러 방식으로 구현하여 적용해 보았다.

`_250612_Multitask.py`에서는 논문의 Task-Specific Attention Module을 변형하고 기존의 파이프라인에 추가하는 방식으로 구현하였으며,

`_250615_multitask2.py`에서는 실제로 ETPC dataset으로 미리 학습 후, QQP dataset으로 재 fine-tuning을 수행하여 진정한 multitask learning을 구현하였다.

3.2.5. PeFT (LoRA, DoRA)

`_250614_PeFT.py`

LoRA (Hu et al., 2022), DoRA (Liu et al., 2024) 방식을 이용하여 Parameter-Efficient FineTuning을 통해 성능을 확인하였다.

3.2.6. Pooling 방식 변경

`_250614_pooling.py`

sentence embedding 추출 시 기존의 last token을 사용하는 것이 아닌, 모든 token에 대해 mean pooling을 진행하는 방식으로 학습 파이프라인을 수정하여 구현하였다.

3.2.7. BackTranslation

`_250613_default_BackTranslation`

'BET: A BACKTRANSLATION APPROACH FOR EASY DATA AUGMENTATION IN TRANSFORMER-BASED PARAPHRASE IDENTIFICATION CONTEXT' 논문에서의 BackTranslation(Corbeil and Ghadivel, 2020) 방식을 차용하여 구현하였다.

원본 dataset을 6개 언어(ar, de, es, fr, ru, zh)로 번역 후, 다시 역변역하고, 증강된 데이터를 모두 합쳐 하나의 dataset으로 만들어 학습을 진행하였다.

3.3. Results

아래는 위의 각 방식에 따른 fine-tuning 후, dev set에 대한 test Accuracy, F1 score를 정리한다.

Method	Accuracy	F1 Score	others
Baseline(_250523_default_paraphrase)	0.9081	0.9019	epoch=10, lr=1e-5
Baseline(_250614_pooling, last)	0.8919	0.8849	epoch=20, lr=2e-5
Baseline(_250613_default_BackTranslation)	0.8854	0.8781	epoch=10, lr=2e-5
SimCSE(_250522_SimCSE)	0.8946	0.8879	epoch=3, lr=2e-5
SimCSE(_250523_simcse_etc)	0.8682	0.8594	epoch=3, lr=2e-5
Adverserial(_250611_Adverserial_gem)	0.8909	0.8836	epoch=30, lr=2e-5
Multitask(_250612_Multitask)	0.8891	0.8505	epoch=20, lr=2e-5
Multitask(_250615_multitask2)	0.7709	0.7484	epoch=10/20, lr=2e-5
PeFT(_250614_PeFT, LoRA)	0.8707	0.8623	epoch=20, lr=5e-5 r=16, alpha=32, dropout=0.1
PeFT(_250614_PeFT, LoRA)	0.8599	0.8498	epoch=20, lr=5e-5 r=8, alpha=16, dropout=0.1
PeFT(_250614_PeFT, DoRA)	0.8702	0.8616	epoch=20, lr=5e-5 r=16, alpha=32, dropout=0.1
PeFT(_250614_PeFT, DoRA)	0.8601	0.8501	epoch=20, lr=5e-5 r=8, alpha=16, dropout=0.1
Pooling(_250614_pooling, mean)	0.8872	0.88	epoch=20, lr=2e-5
BackTranslation(_250612_BackTranslation)	0.8784	0.871	epoch=20, lr=2e-5

4. Sonnet Generation

4.1. Experiments Setting

Shakespeare 소네트를 생성하는 실험은 다음과 같은 환경에서 수행되었다.

- GPU: NVIDIA A6000 48GB x 1 사용
- 데이터셋: Shakespeare 소네트 텍스트 파일 기반 학습셋, 검증셋 분리
- 평가 데이터셋: TRUE_sonnets_held_out_dev.txt 사용
- 모델: gpt2 기반 모델 사용
- Batch size: 대부분의 실험에서 기본적으로 8 또는 32 로 설정
- Learning rate: 실험에 따라 1e-6 ~ 5e-5 범위 사용
- Seed: 재현성을 위해 11711 고정
- 혼합 정밀도 학습(Mixed Precision): torch.set_float32_matmul_precision('high') 활성화
- Evaluation metric: sacrebleu.metrics.CHRF 의 CHRF++ 점수 사용
- Decoding 방식: Top-p sampling, Beam Search, Contrastive Search, MBR reranking 등 비교 실험 진행

4.2. Experiments

4.2.1. Baseline

sonnet_generation_base.py

GPT-2 구조를 그대로 사용하여, 전체 hidden state로부터 logit을 생성하는 forward()를 구현하고, top-p sampling을 이용한 generate() 함수를 통해 기본적인 소네트 생성을 수행하였다.

Weight tying을 적용하였으며, 학습 없이 구조만 구현한 상태에서 baseline 성능을 측정하였다.

4.2.2. Prefix-Tuning

sonnet_generation_Prefix.py

Li & Liang (2021)의 Prefix-Tuning 방법을 기반으로 구현하였다.

GPT-2 본체의 파라미터를 모두 freeze하고, 학습 가능한 연속 벡터(prefix embeddings)를 입력 embedding 앞에 concat하는 방식으로 적용하였다. Attention mask 또한 prefix 길이에 맞게 확장하였으며, generate 함수에서도 동일하게 반영되도록 구성하였다.

자세한 구현체는 코드 참고.

4.2.3. Contrastive Search

sonnet_generation_CS.py
Li et al. (2022)의 Contrastive Decoding 방식을 기반으로 구현하였다. top-k 후보군 중에서 hidden state 간 유사도(repulsion score)를 계산하고, 이를 로짓 점수에서 감산하여 대표성이 높은 토큰을 선택하였다.
추가적으로 dynamic_alpha, use_mean_sim, use_repetition_penalty 세 가지 옵션을 조합하여 총 5 가지 설정으로 실험을 진행하였다.
-모든 옵션 비활성 (기본 설정)
-dynamic_alpha 만 활성화
-use_mean_sim 만 활성화
-use_repetition_penalty 만 활성화
-dynamic_alpha, use_mean_sim 만 활성화

4.2.4. Beam Search

sonnet_generation_BS.py
전통적인 Beam Search 알고리즘을 기반으로 구현하였다.
각 단계에서 top-k 토큰 후보를 탐색하여 시퀀스를 확장하며, 길이 정규화(length_penalty)와 n-gram 반복 차단 기능을 포함하였다. EOS 도달 여부에 따라 완료된 후보 시퀀스를 별도로 저장하며, 최종적으로 점수가 가장 높은 후보를 선택하였다.

4.2.5. MBR

sonnet_generation_MBR.py
Minimum Bayes Risk (MBR) reranking 기법을 적용하였다.
Beam Search 를 통해 생성된 상위 n-best 후보들 간 pairwise CHRF 유사도를 계산하고, 평균 유사도가 가장 높은 후보를 최종 출력으로 선택하는 방식이다. 기존 Beam Search 보다 대표성이 높은 시퀀스를 생성하는 것이 목적이다.

4.2.6. Candidate Ensemble + MBR

sonnet_generation_CE.py
Top-p Sampling, Contrastive Search, Beam Search 를 통해 생성된 다양한 후보군을 모두 통합하고, 이들 간의 pairwise CHRF 유사도를 기반으로 MBR reranking 을 수행하였다.
각기 다른 생성 방식의 장점을 혼합하여 가장 일관성 있는 후보를 선택하고자 하는 접근이다.

4.3. Results

Method	Accuracy	others
Baseline (Top-p Sampling)	39.7379	기본 generate 함수, top-p=0.9, temp=1.2
Prefix-Tuning	35.0762	연속 prefix embedding
Contrastive Search (all False)	35.6912	기본 contrastive 설정 (3옵션 모두 꺼짐)
Contrastive Search (only dynamic_alpha)	36.3526	dynamic_alpha=True
Contrastive Search (only use_mean_sim)	35.6912	use_mean_sim=True
Contrastive Search (only repetition_penalty)	24.8650	use_repetition_penalty=True
Contrastive Search (dynamic+mean_sim)	37.7542	best setting (두 옵션만 True)
Beam Search	33.4421	beam_size=5, length_penalty=0.6
MBR	37.7483	Beam 후보 기반 reranking
Candidate Ensemble + MBR	37.6524	Top-p, Contrastive, Beam 통합후 rerank

4.4. Fine-Tuning

sonnet_generation_fine.py

가장 안정적이고 높은 성능을 보였던 baseline (Top-p Sampling) 방식을 기반으로 전체 GPT-2 모델 파라미터를 fine-tuning하였다.

학습 시 forward()는 hidden state 전체를 활용해 logit을 생성하였으며, Top-p 샘플링을 통한 generation 방식은 그대로 유지하였다.

학습 구조는 기존 train() 함수를 개선하여 다음과 같은 기법을 새롭게 추가하였다.

- get_linear_schedule_with_warmup()을 통한 학습률 스케줄링 적용
- clip_grad_norm_()을 통한 그래디언트 클리핑 도입 (max norm=1.0)
- 매 epoch마다 Dev 세트 CHRF 평가 및 최고 성능 상위 3개 모델만 저장

- 불필요한 체크포인트 파일은 자동 삭제하여 저장 공간 최적화

실험은 batch size와 learning rate를 달리하여 여러 조합으로 진행되었으며, 아래와 같은 결과를 얻었다.

Batch Size	**Learning Rate**	**Epochs**	**CHRF Score**	**Notes**
32	1e-6	1000	41.8519	Saved at **epoch 594**
32	5e-5	1000	42.6414	Saved at **epoch 469**
32	1e-4	1000	42.4880	Saved at **epoch 188**
8	1e-5	200	**43.2194**	Saved at **epoch 74** (SOTA)

5. Conclusions

Paraphrase Detection 정리

-> Baseline 이 기본적으로 가장 괜찮은 성능, 그러나 여러 Method 에서 hyperparameter 최적화 시 Baseline 보다 더 성능이 개선될 수 있음을 확인하였음. hyperparameter searching 을 통한 최적화를 이후 과제로 남겨둠. Ensemble 기법을 활용하면 더 좋은 성능을 확인할 수 있음으로 사료되나, 확인이 필요함.

Sonnet Generation 정리

-> Baseline (generate) 구조가 기본적으로 가장 안정적인 성능을 보였으며, fine-tuning 을 통해 CHRF 기준 가장 높은 점수를 달성하였다.

Prefix-Tuning, Contrastive Search, Beam Search, MBR 등 다양한 방법론을 적용한 결과, 일부 전략은 비슷한 수준의 성능을 보였으나 완전한 fine-tuning 에는 미치지 못하였다.

추후에는 다양한 decoding 기법에 대한 조합 탐색(Ensemble) 및 prefix 기반 세분화 학습 전략에 대한 추가 실험이 필요하다.

6. References

Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. SimCSE: Simple contrastive learning of sentence embeddings. (EMNLP 2021).

Animesh Nighojkar and John Licato. 2021. Improving Paraphrase Detection with the Adversarial Paraphrasing Task. arXiv:2106.07691.

Shikun Liu, Edward Johns, and Andrew J. Davison. 2019. End-to-End Multi-Task Learning with Attention. (CVPR 2019).

Hu, Edward J., Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-Rank Adaptation of Large Language Models. (ICLR 2022).

Liu, Shih-Yang, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024. DoRA: Weight-Decomposed Low-Rank Adaptation. (ICML 2024).