자연어처리개론 기말 프로젝트

GPT-2 기반 감정 분류 및 패러프레이즈 탐지 응용 시스템 구현과 성능 확장 실험

AI융합학부 2022113591 신재용

AI융합학부 2022113596 오유나

컴퓨터공학전공 2020112036 김상현

2025년 6월 16일

github 저장소 주소 : https://github.com/JAEYONG-shin0117/osss.git

github 페이지 주소 : https://github.com/JAEYONG-shin0117/osss/tree/main

목차

Abstract

본 연구는 GPT-2 언어 모델을 직접 구현하고, 이를 감정 분류, 패러프레이즈 탐지, 소네트 생성 등 다양한 자연어 처리 태스크에 적용하여 그 가능성과 성능을 실험적으로 검증하였다. 모델 구현은 PyTorch 기반으로 attention, layer module, optimizer 등의 핵심 요소를 직접 구성하였으며, Sanity Check 및 Optimizer 테스트를 통해 기능적 정확성을 확보하였다. 확장 실험에서는 LoRA(Low-Rank Adaptation)와 Adapter 기법을 파인튜닝 전략으로 도입하여 파라미터 효율화를 시도하였고, 소네트 생성의 품질 향상을 위해 품사 템플릿과 제약 기반 Beam Search를 적용한 다음 실험을 진행하였다. 본 연구는 구현 중심의 실험을 통해 경량화된 파인튜닝과 생성 품질 개선을 동시에 달성할 수 있는 실질적 방법론을 제시한다.

1. Introduction

최근 대형 언어모델(LLM)의 발전은 자연어 처리(NLP) 전반의 성능을 비약적으로 향상시켰으며, 특히 Transformer 기반의 GPT 계열 모델은 다양한 태스크에서 우수한 성능을 보여주고 있다. 본 연구는 GPT-2 모델을 직접 구축하고, 감정 분류, 패러프레이즈 탐지, 소네트 생성과 같은 downstream task에 적용함으로써, 모델의 전이 학습 성능과 확장 가능성을 검증하는 데 목적이 있다. 기존 대형 모델은 전체 파라미터를 대상으로 하는 full fine-tuning에 따른 자원 소모가 크다는 단점이 있으며, 생성 태스크에서는 형식 제약 미흡으로 인해 일관성 있는 문장 구조 생성에 한계가 있다. 본 연구는 이러한 한계를 개선하기 위해 다양한 파인튜닝의 방법을 이용하여 기능을 확장하였다. 본 연구구에서는 모델의 기본 구현 방식과 확장된 파인튜닝 및 생성 전략의 구체적인 적용 방법을 기술하고, 각 태스크별 실험 결과를 바탕으로 적용 기법의 효과를 정량적/정성적으로 평가한다.

2. Related Work

2.1 LoRA (Low-Rank Adaptation)

[1] 기존 모델의 가중치 행렬을 그대로 두고, 해당 위치에 저차원 행렬 A와 B를 추가하여 $W + \Delta W = W + BA$ 형태로 표현함으로써 일부 모듈만을 선택적으로 학습할 수 있도록 한다. 특히 이 방식은 attention의 Query와 Value 프로젝션 계층에 적용되어 전체 파라미터의 0.1-0.3%만을 업데이트하면서도 full fine-tuning과 유사한 성능을 보였다. LoRA는 별도의 구조 변경 없이 기존 모델에 plug-in 형태로 적용 가능하며, 다양한 downstream task에서도 확장성을 확보할 수 있는 장점이 있다.

2.2 Adapter

[2] Adapter는 사전학습된 모델의 각 Transformer 블록에 작은 병목 구조(bottleneck structure)를 삽입하는 방식으로, 기존 파라미터를 그대로 두고 새로운 소형 모듈만을 학습한다. 입력 벡터 xxx에 대해 Adapter는 다음과 같은 연산을 수행한다:

$$Adapter(x) = W_{up} \cdot ReLU(W_{down} \cdot x)$$

여기서 W_{down}은 hidden size를 bottleneck 크기로 압축하며, W_{up}은 이를 다시 복원하는 역할을 한다. Adapter는 Transformer 내부의 LayerNorm과 FFN 사이 또는 이후에 residual 형태로 삽입되며, 전체 파라미터 중 약 3% 이하만을 학습하는 방식으로도 full fine-tuning에 준하는 성능을 보일 수 있다. 별도 구조 변경 없이 plug-in 형태로 적용 가능하며, 여러 태스크 간 파라미터 재사용이 용이한 점에서 효율성과 확장성을 동시에 확보할 수 있다.

2.3 패러프레이즈 데이터 증강 기법

패러프레이즈 분류 모델은 문장의 의미론적 동등성을 판단하는 과제로, 대부분의 최신 모델은 사전학습된 언어 모델을 기반으로 한 파인튜닝을 통해 높은 성능을 달성한다. 그러나 이러한 모델들은 라벨링된 데이터가 제한될 경우 과적합에 취약하며 실제 도메인에서의 일반화 성능이 저하되는 한계가 있다. 본 연구는 이러한 문제를 극복하기 위해 데이터 증강 기법으로 동의어 치환(Synonym Replacement)과 역번역(Back Translation)을 적용하였다.

2.3.1. 동의어 치환(Synonym Replacement)

동의어 치환은 문장 내 특정 단어를 유의어로 대체함으로써 문장의 의미를 보존하면서도 문장 구조를 다양화할 수 있다는 점에서 간단하면서도 효과적인 연구된 방법이다. Wei and Zou(2019)는 EDA(Easy Data Augmentation) 기법을 제안하며, 랜덤 삽입, 삭제, 교체 등의 간단한 조작을 통해 텍스트 분류 모델의 성능을 유의미하게 향상시킬 수 있음을 보였다. 특히, 그 중 동의어 치환은 WordNet과 같은 어휘 자원을 기반으로 단어를 유의어로 치환하여 문장의 의미를 유지하면서 표현을 다양화할 수 있는 대표적 기법으로 소개되었다.

NLTK의 WordNet API를 이용해 문장 내 1~2개의 단어를 동의어로 무작위 교체하여 훈련 데이터를 확장하는 방식으로 시도되었으나, 문맥에 어색한 단어 교체가 의미를 훼손하거나 GPT-2 모델이 단어 수준보다는 문장 구조의 다양성에 민감하다는 특성 때문에 오히려 정확도가 감소하는 결과를 초래하여 채택되지 않았다.

2.3.2. 역번역(Back Translation)

역번역은 문장 전체를 외국어로 번역한 후 다시 원어로 되돌리는 방식으로 표현을 다양화하면서도 의미는 유지하는 효과를 기대할 수 있으며, 본 연구에서는 MarianMT 기반의 Helsinki-NLP 모델을 활용하여 영어-불어-영어 경로를 통해 데이터의 다양성을 확보하였다. 유럽어 계열 중 불어는 구조적 다양성을 제공하면서도 의미 왜곡 가능성이 낮다고 평가되며, Sennrich 등의 연구에 기반하여 중간 언어로 선정되었다. 전체 훈련 데이터 중 30%를 무작위로 추출해 질문 1과 질문 2를 각각 역번역 처리하였고, 이 과정에서 원문과 지나치게 유사하거나 의미 손실이 의심되는 경우는 필터링하여 최종 훈련셋에 포함되지 않도록 하였다.

2.4 Pre-LayerNorm

기존 GPT-2는 Post-LayerNorm 구조를 채택하였으나, 이 구조는 깊은 네트워크에서 gradient vanishing 또는 explosion 현상에 취약하다는 한계가 제기되었다. Xiong et al.(2020)은 Pre-LayerNorm 구조가 Transformer 학습 과정의 안정성과 수렴 속도를 개선하며, 특히 딥 네트워크에서의 학습 어려움 및 optimization instability를 완화한다고 보고하였다.

Pre-LayerNorm 구조의 핵심은 각 sub-layer(attention, feed-forward) 진입 전에 LayerNorm을 적용함으로써 다음과 같은 효과를 얻는 것이다:

- Gradient Flow 개선: LayerNorm 사전 적용으로 gradient 전파 경로가 안정화됨
- 학습 초기 안정성: 초기 학습 단계에서 발생할 수 있는 gradient 문제 완화
- 수렴 속도 향상: 보다 안정적인 최적화 과정을 통한 빠른 수렴

Pre-LayerNorm 구조의 적용을 위해서는 다음과 같은 기술적 수정이 필요하다:

- Transformer 블록의 LayerNorm 위치를 GPT-2의 기본 구조에서 변경
- transformers.models.gpt2.modeling_gpt2.GPT2Model의 재구현 또는 서브클래싱
- 전체 파인튜닝 파이프라인의 재설계

하지만, 본 프로젝트 환경에는 다음과 같은 제약 조건이 존재한다:

- gpt2.py 내 구조 수정 금지: 교수자 제공 기준 코드의 변경 제한
- 호환성 문제: 기존 학습 코드와의 전면적인 호환성 재설계 필요
- 구현 범위 초과: 단일 증강 실험 목적을 넘어서는 구조적 변경

결과적으로 Pre-LN 구조를 도입한 실험은 구현 상의 제한(수정 금지된 모듈)과, 기존 학습 코드의 전체적인 호환성 문제로 인해 실질적인 적용이 불가능하다는 판단 하에 초기 검토 단계에서 폐기되었다.

향후 조건이 허용된다면 Pre-LN 구조를 포함한 Transformer 레이어 최적화 전략을 재검토할 수 있을 것으로 판단된다.

2.5 POS template

[6]]에 따르면, Sonnet Generation에서 Sonnet은 약강 5보격과 ABAB CDCD EFEF GG의 운율 형식을 엄격하게 지 켜야 하기 때문에, 이러한 시의 형식적 특성을 생성 단계에서 직접적으로 강제 하여 문장 구조의 정확성과 안정성을 보장할 수 있다.

따라서 train 데이터 셋에서 문장 단위로 품사 태깅을 하여 품사 템플릿(POS template)을 생성하고, 문장을 생성할 때 품사 템플릿에서 빈도수가 높은 문장 구조중 하나를 랜덤하게 선택해서 Sonnet 생성 프롬프트에 추가하고, 해당 품사에 맞춰서 Sonnet 문장을 생성하도록 유도를 해서, 형식적으로 적합한 토큰만 허용하여 생성 과정을 구조적으로 엄격히 제한하고자 했다.

하지만, 주어진 데이터 셋에서 POS template을 추출했을 때, 데이터 셋의 크기가 매우 작아서 추상화 단계를 많이 낮췄지만 문장 구조가 일치하는 경우가 거의 없었고, 결과적으로는 Sonnet Generation에서 POS template를 적용하는 방법은 폐기되었다.

2.6 Beam Search

추가적으로 [6]에 따르면, Beam Search를 적용해서 GPT2에서 Sonnet Generation 성능을 향상시킬 수 있다. Beam Search는 텍스트 생성 과정에서 탐색의 효율성과 결과 품질을 향상시키기 위해 널리 활용되는 휴리스틱 기반의 탐색 알고리즘이다. 일반적으로 Transformer 기반 언어 모델, 예를 들어 GPT-2와 같은 모델은 Greedy Search 또는 샘플링 기반 방법으로 다음 토큰을 선택하는데, 이러한 방식은 최적의 전체 문장을 생성하는 데 비효율적일 수 있다.

Beam Search의 핵심 아이디어는 매 단계에서 가장 확률이 높은 토큰 하나만을 선택하는 Greedy Search와 달리, 각 단계마다 고정된 크기의 후보 집합(beam)을 유지하면서 여러 경로를 병렬적으로 탐색하는 것이다. 구체적으로 Beam Search는 다음과 같은 과정으로 동작한다:

1. 초기 단계에서 beam 크기(k)만큼의 후보를 선택한다.
2. 각 후보에서 다음 단계의 가능한 모든 토큰을 확장하여 전체 후보 집합을 만든다.
3. 확장된 후보 중 가장 높은 확률을 가지는 상위 k개의 후보만 유지하고 나머지를 제거한다.
4. 이러한 과정을 문장 종료 토큰(end-of-sentence token)을 만나거나 최대 길이에 도달할 때까지 반복한다.

Beam Search의 주요 장점은 다음과 같다:

1. 탐색 범위 확대: 여러 경로를 동시에 고려하므로 보다 우수한 전체 문장을 생성할 가능성을 높인다.
2. 출력 품질 향상: Greedy Search보다 문법적으로 정확하고 자연스러운 문장 구조를 생성할 수 있다.

3. 활용성 확대: 구조적 제약이 강한 생성 과제, 예컨대 Sonnet Generation처럼 엄격한 형식을 유지해야 하는 상황에서도 효과적이다.

다만 Beam Search는 연산량과 메모리 사용량이 증가하는 단점이 있으며, beam 크기를 크게 설정하면 계산 비용이 급격히 증가할 수 있다. 그리고 내부적으로 테스트를 진행 했을 때, CHRF 점수 결과가 좋지 않아 해당 방법은 폐기되었다. 이는 4-3장에서 자세히 설명하도록 하겠다.

2.7 top-p Sampling and Temperature Scaling

[7]Holtzman et al. (2020)은 기존의 greedy decoding이나 top-k sampling 방식이 종종 의미 없는 반복, 단조로운 문장, 또는 확률 분포의 극단적인 편향으로 인해 비자연적인 텍스트를 생성한다고 지적하였다. 이러한 문제를 해결하기 위해 제안된 nucleus sampling (top-p sampling)은 확률 분포 상에서 누적 확률이 p 이하인 최소 토큰 집합만을 대상으로 샘플링을 수행함으로써, 모델의 창의성과 표현 다양성을 보장할 수 있다. 또한, temperature scaling 기법은 softmax 이전의 logits 값을 온도 계수로 나누어 확률 분포의 sharpness를 조절한다. 낮은 온도는 분포를 더욱 뾰족하게 만들어 높은 확률의 토큰만을 선택하도록 유도하며, 높은 온도는 더 평평한 분포를 만들어 다양한 토큰이 선택될 가능성을 높인다. 이러한 조합은 특히 자유도가 높은 생성 태스크에서 매우 효과적인 디코딩 전략으로 평가받고 있다.

3. Methodology

3.1 GPT-2 모델 기본 구현

본 연구에서는 GPT-2 모델의 내부 구조를 PyTorch 기반으로 직접 구현하였다. 모델은 모듈화된 구조로, attention 메커니즘(attention.py), Transformer 레이어(gpt2_layer.py), 전체 모델 아키텍처(gpt2.py)로 나뉘며, 각 구성 요소는 다음과 같이 정의되었다.

3.1.1 Self-Attention 메커니즘

attention.py에 구현된 CausalSelfAttention 클래스는 GPT-2에서 사용하는 causal mask 기반의 multi-head self-attention 구조이다. Query, Key, Value는 각각 독립적인 nn.Linear를 통해 계산되며, einops.rearrange를 활용하여 입력 텐서를 multi-head 구조로 변환한다. Attention score는 다음과 같이 계산된다:

```
attention_scores = torch.matmul(query, key.transpose(-1, -2))
attention_scores = attention_scores / math.sqrt(self.attention_head_size)
attention_scores = attention_scores + attention_mask
attention_probs = nn.functional.softmax(attention_scores, dim=-1)
context_layer = torch.matmul(attention_probs, value)
```

context_layer는 마지막에 head를 concat하여 [batch, seq_len, hidden_size] 형상으로 복원된다.

3.1.2 GPT-2 레이어

gpt2_layer.py에는 Transformer 블록 하나가 구현되어 있으며, 구조는 Pre-LN 기반의 GPT-2 형식이다. 각 블록은 다음과 같이 구성된다:

- 입력 → LayerNorm → CausalSelfAttention → Residual 연결
- Residual 결과 → LayerNorm → FeedForward(GELU) → Residual 연결

Dropout은 attention과 feed-forward 출력에 각각 적용되며, residual 연결은 add() 함수에서 다음과 같이 정의된다:

```
output = input + dropout(dense_layer(output))
```

이를 통해 학습 안정성과 수렴 속도를 높였다.

3.1.3 GPT-2 전체 모델

gpt2.py에 구현된 GPT2Model은 전체 아키텍처를 조립하며 다음 단계를 따른다:

1. TokenEmbedding + PositionEmbedding + Dropout
2. N개의 GPT2Layer 인코더 통과
3. Final LayerNorm 적용
4. 마지막 non-padding 토큰의 hidden state를 추출하여 downstream task에 활용

또한 hidden_state_to_token() 메서드를 통해 출력 hidden state를 임베딩 weight와의 내적으로 로짓(logits)으로 변환하는 가중치 공유(weight tying)를 구현하였다.

3.1.4 감정 분류기

classifier.py에 구현된 GPT2SentimentClassifier는 GPT-2를 백본으로 사용하는 감정 분류기이다. 입력 문장은 GPT-2를 통해 임베딩되며, 출력은 다음 방식으로 분류된다:

```
gpt_output = self.gpt(input_ids=input_ids, attention_mask=attention_mask)
x = gpt_output['last_token']
x = self.dropout(x)
logits = self.classification_head(x)
```

학습 시 F.cross_entropy(logits, labels) 손실을 사용하며, fine-tuning 전략은 last-linear-layer(GPT 파라미터 고정) 또는 full-model(전체 파라미터 업데이트)로 선택 가능하다.

3.1.5 optimizer

optimizer.py에서는 PyTorch 기본 Optimizer를 상속하여 AdamW를 직접 구현하였다. 핵심은 그래디언트 기반의 파라미터 업데이트와 weight decay를 분리 적용하는 것이다. 각 파라미터에 대해 다음 연산을 수행한다:

```
exp_avg.mul_(beta1).add_(grad, alpha=1 - beta1)
exp_avg_sq.mul_(beta2).addcmul_(grad, grad, value=1 - beta2)
...
p.data.mul_(1 - lr * weight_decay)
p.data.addcdiv_(exp_avg, denom, value=-step_size)
```

이를 통해 bias correction과 학습 안정성이 보장되며, optimizer_test.py를 통해 구현 정확성을 검증하였다.

3.1.6 패러프레이즈 탐지 모델

paraphrase_detection.py에서는 GPT-2를 백본으로 한 패러프레이즈 탐지 모델 ParaphraseGPT를 정의하였다. 해당 모델은 입력 시퀀스의 hidden state를 평균 풀링하여 이진 분류를 수행한다. 학습 시에는 cross_entropy 손실을 사용하며, 모델 구조는 다음과 같다:

- GPT-2 모델의 모든 hidden state 출력
- Attention mask를 사용한 masked mean pooling
- Linear layer를 통한 이진 분류 (paraphrase 여부)

```
hidden_states = self.gpt(input_ids, attention_mask)['last_hidden_state']
mask_expanded =
attention_mask.unsqueeze(-1).expand(hidden_states.size()).float()
mean_hidden = torch.sum(hidden_states * mask_expanded, dim=1) /
torch.clamp(torch.sum(mask_expanded, dim=1), min=1e-9)
logits = self.paraphrase_detection_head(mean_hidden)
```

3.1.7 소네트 생성 모델

sonnet_generation.py에는 GPT-2 기반의 시 생성 모델 SonnetGPT가 정의되어 있다. 해당 모델은 GPT-2를 활용하여 각 토큰에 대한 다음 토큰의 확률 분포를 학습하며, 학습은 Cross Entropy 기반의 Language Modeling 손실을 사용한다. 생성 시에는 Top-p(Nucleus) 샘플링 및 Softmax Temperature를 조합하여 시퀀스를 생성한다:

```
logits_last_token = logits_sequence[:, -1, :] / temperature
probs = torch.nn.functional.softmax(logits_last_token, dim=-1)
... (top-p filtering 후 샘플링) ...
token_ids = torch.cat([token_ids, sampled_token], dim=1)
훈련 데이터와 별도로 held-out 데이터셋이 존재하며, 시작 줄만 주어진 채 전체
소네트를 생성해야 한다. 생성된 결과는 제출용 텍스트 파일로 저장된다.
```

3.2 LoRA 적용

본 연구에서는 Edward Hu et al. (2021)이 제안한 Low-Rank Adaptation(LoRA) 기법을 기반으로, [1]GPT-2 모델의 attention layer 내 Query 및 Value projection에 저차원 행렬 A ∈ ℝ^(r×d), B ∈ ℝ^(d×r)를 삽입하였다. 여기서 d는 입력 차원, r은 low-rank 차원으로 설정되며, 본 연구에서는 r = 8을 기본값으로 사용하였다. PyTorch 기반 구현에서, 기존 Query 프로젝션 레이어는 nn.Linear 모듈로 정의되어 있으며, 해당 위치에 LoRALinear 클래스를 도입하여 기존 nn.Linear 모듈을 대체하였다.

LoRALinear는 다음 수식을 따른다.

$$W_{eff} = W + BA$$

이때 W는 고정된 기존 weight이고, A (r × d), B (d × r)는 학습 가능한 low-rank 파라미터이며, 전체 연산은 다음과 같다:

```
base = x @ self.weight.T
delta = (x @ self.A.T) @ self.B.T * (alpha / r)
output = base + delta + bias
```

이를 통해 GPT-2의 attention 모듈에 LoRA를 적용했으며, 적은 수의 학습 파라미터로도 downstream task 성능을 유지하였다.

3.3 Adapter 적용

본 연구에서는 GPT-2 기반 모델의 효율적인 파인튜닝을 위해 [Houlsby et al., 2019]에서 제안한 Adapter 기법을 적용하였다. Adapter는 Transformer 블록 내에 삽입되는 작은 병목 구조(bottleneck structure)로, 원래의 사전학습된 파라미터를 동결한 상태에서 비교적 적은 수의 추가 파라미터만 학습함으로써 모델의 미세조정 효율을 극대화한다.본 구현에서는 각 GPT-2 블록의 Feed-Forward 후처리 영역에 Adapter를 삽입하였다. Adapter는 2.2장에서 설명했던 것처럼 다음과 같은 구조를 갖는다:

$$Adapter(x) = W_{up} \cdot ReLU(W_{down} \cdot x)$$

여기서 W_{down} ∈R^d×b, W_{up} ∈R^b×d, b는 bottleneck 크기(본 실험에서는 64)이며, d는 hidden dimension (GPT-2의 경우 768)이다. Adapter는 입력을 bottleneck 크기로 압축한 후, 비선형 활성화를 거쳐 다시 원래 차원으로 복원하는 방식으로 작동한다. 최종 출력은 residual connection으로 본래 hidden state와 합산된다:

$$Output = x + Adapter(x)$$

구현 상으로는 adapter.py에 정의된 Adapter 클래스가 gpt2_layer.py 및 classifier.py에 삽입되어 있으며, 다음과 같이 통합된다:

```
output = self.adapter(output) + output (gpt2_Layer.py)
```

```
x = x + self.adapter(self.adapter_norm(x)) (classifier.py)
```

Adapter는 LoRA와 병렬로 사용되며, 파라미터 수를 크게 증가시키지 않으면서도 downstream task에 대한 모델 적응력을 크게 향상시킨다.

3.4 데이터 증강 적용

3.4.1. 동의어 치환

본 실험에서는 텍스트 패러프레이즈 판별 모델의 학습 성능 향상을 위하여, WordNet 을 기반으로 데이터 증강을 시도하였다. 증강 방식은 문장의 의미를 보존하는 범위 내에서 어휘적 다양성을 도입하고자 했다.

구체적으로는 입력 문장 내에서 동의어가 존재하는 단어들을 식별한 후, 이들 중 최대 n=2개의 단어를 무작위로 선택하여 해당 단어를 동의어로 치환하였다. 각 단어 w에 대해 WordNet에서 동의어 후보 집합 S(w)를 다음과 같이 정의한다:

$$S(w)=\{s \in \text{WordNet}(w) \mid s \neq w\}$$

이후 치환 대상 단어 $w_i \in \{w_1,\ldots,w_k\}$가 선택되면, 그 중 무작위로 하나의 동의어 $s_i \in S(w_i)$를 샘플링하여 원문 내 해당 단어를 대체한다. 이 과정을 통해 생성된 문장은 원문과 동일한 라벨을 부여받아 훈련 데이터에 추가되었다.

단, 실험 결과 초반 학습 단계에서 모델 성능의 유의미한 저하가 관측되었으며, 이는 문맥과 무관한 치환에 따른 의미 왜곡 가능성에 기인한 것으로 분석된다. 이에 따라 동의어 치환 기반 증강은 실험 초기 단계에서 철회되었다.

3.4.2. 역번역

역번역(back-translation)은 원문을 중간 언어(pivot language)로 번역한 후, 다시 원문 언어로 재번역하는 과정을 통해 문장의 의미는 유지하면서 표현을 다양화하는 데이터 증강 기법이다. 본 실험에서는 이 방식을 통해 훈련셋의 문장 다양성을 증가시키고, 모델의 일반화 성능을 향상시키고자 하였다.

역번역을 위해 기본 패러프레이즈 모델에 역번역 전처리 모듈을 삽입하였고 이를 바탕으로 원문 문장 x를 다음과 같은 두 단계로 처리하였다:

1단계: 전방 번역 (Forward Translation)

```
translated = model_pivot.generate(**tokenizer_pivot(text))
pivot_text = tokenizer_pivot.decode(translated[0])
```

2단계: 역번역 (Back Translation)

```
back_translated = model_back.generate(**tokenizer_back(pivot_text))
result = tokenizer_back.decode(back_translated[0])
```

역번역을 통한 데이터 증강은 학습 데이터 로딩 단계에서 조건부로 적용되었다. 먼저, 원문 문장은 최소 길이 3자 이상, 최대 길이 500자 이하인 경우에만 역번역 대상으로 선정되었으며, 과도하게 짧거나 긴 문장은 제외되었다.

이후 역번역을 통해 생성된 문장이 원문과 어휘적으로 완전히 동일하지 않은 경우에 한해 유효한 증강 데이터로 간주되었다. 반면, 역번역 결과가 원문과 동일하거나 처리 중 오류가 발생한 경우에는 해당 문장을 증강에 활용하지 않고 원문만을 학습 데이터로 사용하였다. 이와 같은 조건을 통해 역번역이 문장 표현의 다양성을 확보하면서도 데이터 품질을 유지할 수 있도록 설계하였다.

3.5 Beam Search 적용

본 연구에서는 생성되는 소넷의 일관성과 품질을 향상시키기 위해 기존의 Top-p 샘플링 방식을 Beam Search 알고리즘으로 대체하였다. Beam Search 텍스트 생성의 각 단계에서 단일 토큰이 아닌, 가장 확률이 높은 k개의 후보 시퀀스(beam)를 유지하며 탐색하는 휴리스틱 탐색 기법이다. 이를 통해 보다 전체적인 문맥을 고려한 텍스트 생성이 가능해진다.

구현된 generate 함수는 다음과 같은 과정을 거친다.

1. 초기화: 주어진 프롬프트 텍스트를 시작으로 하는 단일 Beam(시퀀스와 누적 로그 확률 점수 0.0)으로 탐색을 시작한다.
2. 확장: 현재 각 Beam에 대해, 모델은 다음 토큰의 확률 분포를 예측합니다. 그중 로그 확률이 가장 높은 k개의 토큰을 선택하여 기존 시퀀스에 추가함으로써, 총 k * k개의 새로운 후보 시퀀스를 생성한다.
3. 반복 페널티(Repetition Penalty): 단조로운 반복을 방지하기 위해, 이미 시퀀스에 등장한 토큰의 로짓(logit)에 페널티(기본값 1.2)를 적용하여 해당 토큰이 다시 선택될 확률을 낮췄다.
4. 선택 및 가지치기(Pruning): 생성된 모든 후보 시퀀스를 누적 로그 확률 점수를 기준으로 평가한다. 이때 짧은 시퀀스가 유리해지는 것을 방지하기 위해 길이 정규화(length normalization)를 적용했다. 정규화된 점수가 가장 높은 상위 k개의 시퀀스만을 다음 단계의 빔으로 선택한다.
5. 종료: 모든 빔이 문장 끝(EOS) 토큰에 도달하거나, 설정된 최대 길이에 도달하면 탐색을 종료하고 가장 높은 점수를 가진 Beam의 시퀀스를 최종 결과로 반환한다.

본 실험에서는 Beam의 개수(num_beams)를 5로 설정하여, 생성의 다양성과 품질 사이의 균형을 맞추고자 했다, 하지만 CHRF 점수 결과가 좋지 않아 해당 방법은 폐기되었다. 이는 4-3장에서 자세히 설명하도록 하겠다.

3.6 Decoding Strategy (Top-p + Temperature Sampling)

소네트 생성 시, 모델의 출력 분포로부터 다음 토큰을 결정하기 위해 확률적 샘플링 방식을 사용하였다. 특히 Holtzman et al. (2020)이 제안한 nucleus sampling (top-p sampling) 기법과 temperature scaling을 결합하여 적용하였다.모델 출력 logits에 대해 temperature τ\tauτ를 적용하여 확률 분포의 sharpness를 조절한 후, softmax를 적용해 확률 분포를 계산하였다. 이후, 확률이 높은 순으로 정렬된 토큰들 중 누적 확률이 top-p threshold (기본값 p=0.9)를 넘지 않는 토큰들만을 선택하여 마스크를 적용하였다. 이 마스크는 첫 번째 토큰을 항상 포함하도록

설정하였으며, 선택된 확률 값은 재정규화를 거쳐 multinomial 샘플링을 통해 최종 토큰을 결정하였다. 샘플링 로직은 다음과 같은 순서로 구현되었다:

1. 모델 출력 logits에 temeprature를 적용

   $$logits \leftarrow logits / \tau$$

2. softmax를 통해 확률 분포 계산
3. 확률을 내림차순 정렬한 후 누적합 계산
4. 누적 확률이 top-p를 넘는 지점까지만 마스크를 적용
5. 마스크된 분포를 정규화 후 multinomial 샘플링 수행

코드 내에서는 다음과 같이 구현되었다

```
logits_last_token = logits_sequence[:, -1, :] / temperature
probs = torch.nn.functional.softmax(logits_last_token, dim=-1)

sorted_probs, sorted_indices = torch.sort(probs, descending=True)
cumulative_probs = torch.cumsum(sorted_probs, dim=-1)
top_p_mask = cumulative_probs <= top_p
top_p_mask[..., 1:] = top_p_mask[..., :-1].clone()
top_p_mask[..., 0] = True

filtered_probs = sorted_probs * top_p_mask
filtered_probs /= filtered_probs.sum(dim=-1, keepdim=True)

sampled_index = torch.multinomial(filtered_probs, 1)
sampled_token = sorted_indices.gather(dim=-1, index=sampled_index)
```

4. Experiment

4.1 감정분류기

본 실험에서는 사전학습된 GPT-2 모델에 경량화된 적응 모듈인 LoRA와 Adapter를 적용하여, 감성 분류 과제에 대한 성능을 분석하였다. LoRA와 Adapter를 적용하기 전 모델과 적용 후의 모델을 비교하여 실험을 수행하였다. 실험은 영어 기반 SST(Sentiment Treebank)와 한글 기반 CFIMDB 데이터셋을 대상으로 진행하였다.

- Baseline : 사전학습된 GPT-2를 그대로 사용하고, 분류 헤드만 학습

- LoRA+Adapter : GPT-2 본체는 고정한 채, LoRA와 Adapter 모듈을 추가하여 파인튜닝 수행

SST 실험 결과 비교

설정	최고 에폭	학습 정확도	테스트 정확도
Baseline	7	0.494	0.466
LoRA+Adapter	3	0.623	0.520

LoRA+Adapter를 추가한 모델이 기존 GPT-2 단독 사용 대비 테스트 정확도 기준 약 5.4%% 향상되었고 학습 정확도도 빠르게 상승하는 경향을 보였다.

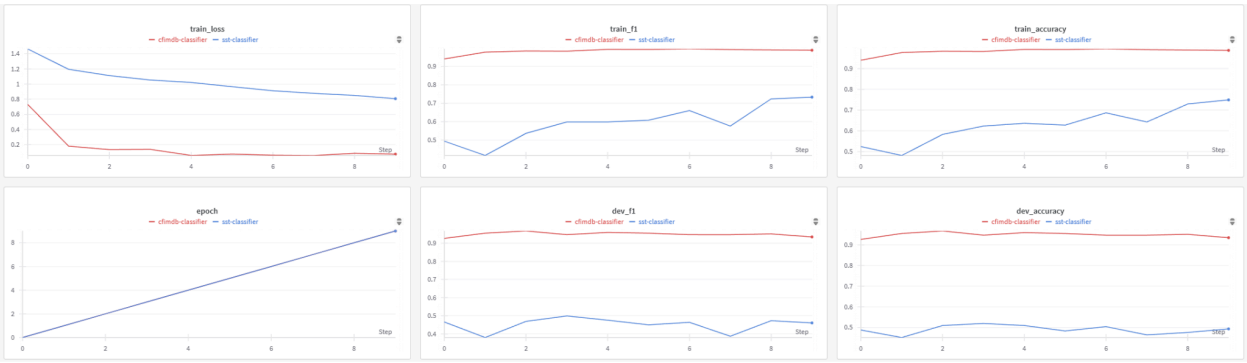
CFIMDB 실험 결과 비교

설정	최고 에폭	학습 정확도	테스트 정확도
Baseline	6	0.868	0.869
LoRA + Adapter	2	0.984	0.967

CFIMDB에서도 LoRA + Adapter 적용 시 약 9.8% 테스트 정확도 향상이 관찰되었고 학습 정확도는 0.984로 수렴하였다.

본 실험을 통해 LoRA + Adapter 기반의 경량 파인튜닝 방식이 GPT-2의 표현력 향상에 영향을 주었고 특히 모델 전체를 업데이트하지 않고도, 적은 수의 추가 파라미터만으로 성능을 안정적으로 개선할 수 있음을 STT와 CFIMDB 두 데이터셋에서 모두 확인하였다.

Baseline 학습 그래프



LoRA + Adapter 학습 그래프

4.2. 패러프레이즈 탐지

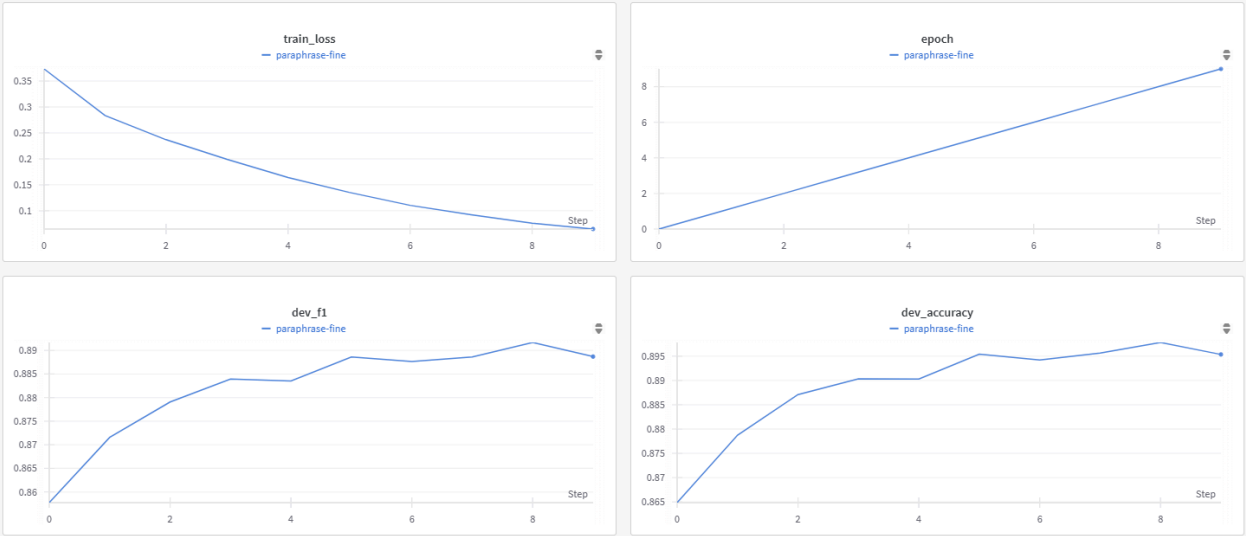
본 실험에서는 사전학습된 패러프레이즈 탐지 모델에 역번역(back-translation) 기반의 데이터 증강 기법을 적용함으로써, 모델 성능에 어느 정도의 향상을 유도할 수 있는지를 분석하였다. 특히 의미적 미묘한 차이가 중요한 Quora Question Pairs(QQP) 데이터셋을 대상으로 역번역 기법이 실제로 도움이 되는지를 정량적으로 평가하는 데 초점을 맞추었다.

역번역은 기존 문장을 중간 언어(예: 영어 → 불어 → 영어)로 번역한 후 되돌리는 방식으로 새로운 문장을 생성하는 데이터 증강 기법이다. 본 실험에서는 원본 QQP 훈련셋의 각 문장에 대해 Google Translate 기반의 역번역을 수행하였고, 생성된 문장이 원문과 비교적 유사한 의미를 유지하는 경우에 한해 기존 쌍의 의미를 변경하지 않는 선에서 문장 구조를 다변화하는 augmentation을 수행하였다. 이를 통해 paraphrase 관계를 유지한 채로 문장 표현이 다른 새로운 데이터 쌍을 대량 확보할 수 있었으며, 총 500쌍을 우선적으로 증강해 실험에 적용하였다.

초기 학습 단계 (Epoch 1~3) 에서는 두 모델 모두 유사한 손실 감소 및 정확도 향상을 보이며, 역번역 데이터가 모델 성능에 크게 기여하는 모습은 관찰되지 않았다. 후반 학습 단계 (Epoch 4~6) 에서는 역번역 모델이 상대적으로 더 높은 dev accuracy를 달성하였으며, 특히 최종 Epoch에서는 기본 모델(0.895) 대비 역번역 모델이 0.899의 정확도를 기록하여 미세한 수준의 성능 향상을 확인할 수 있었다.

설정	최고 에폭	Train Loss	Test Accuracy
Baseline	6	0.136	0.895
LoRA + Back Translation	6	0.138	0.899

Baseline 학습 그래프



LoRA + Back Translation 학습 그래프

4.3. 소네트 생성

설정	CHRF
Baseline	40.942
LoRA + Adapter + temperature	41.538
LoRA + Adapter + Beam Seatch	25.112

Beam Search를 추가로 적용했을 때 CHRF 점수는 25.122가 나왔다. 주요 원인은 다음과 같다.

1. 생성 방식의 결정론적 특성으로 인한 다양성 감소

Top-p 샘플링 방식은 무작위성을 내포하여, 확률이 낮더라도 정답과 우연히 유사한 문자 시퀀스를 생성할 수 있었다. Beam Search는 매 탐색 단계에서 누적 확률이 가장 높은 시퀀스만을 선택하는 결정론적(deterministic) 방식으로, 이 과정에서 무작위성이 배제되어, 샘플링 방식에서 발생하던 '우연한 문자열 일치'가 사라져 CHRF 점수 하락의 원인이 된다.

2. 고확률(High-Probability) 시퀀스 선택으로 인한 어휘의 일반화

Beam Search는 훈련 데이터 기준, 가장 보편적이고 '안전한' 단어 조합(고확률 시퀀스)을 생성하는 경향이 있다. 이로 인해, 참조 텍스트(정답)에 포함된 독창적이거나 시적 허용이 가미된 저확률 어휘 대신 일반적인 어휘를 선택하게 된다. 생성된 텍스트와 참조 텍스트 간 어휘적 차이가 발생하여, 문자 n-gram 기반의 CHRF 점수가 하락할 수 있다.

3. 평가 지표(CHRF)와 실제 생성 품질 간의 괴리

CHRF는 참조 텍스트와 생성 텍스트 간의 어휘적 유사성(lexical similarity)만을 측정하는 지표로, Sonnert의 핵심 요소인 운율(meter), 압운(rhyme), 구조적 일관성 등 문학적 품질은 평가 범위에 포함되지 않는다. 따라서, Beam Search를 통해 생성된 결과물이 인간이 평가하기에 더 높은 완성도를 가지더라도, 참조 텍스트와 사용된 어휘가 다르면 CHRF 점수는 낮게 측정될 수 있다.

이러한 분석을 통해 알 수 있듯이, Beam Search는 생성 품질 측면에서 일정 부분 유의미한 결과를 제공할 수 있지만, 본 과제에서는 CHRF 점수 향상을 목적으로 진행했으므로 확률적 다양성을 보장하는 top-p + temperature 샘플링 방식이 더 적합하다 판단하였다. 이에 따라, 확률 기반의 샘플링 전략을 채택하여 소네트를 생성하였다. top-p + temperature 샘플링을 적용했을 때의 점수는 41.538점으로 향상하였다.

5. Discussion

5.1 감정 분류기

SST 결과 해석

SST는 감정 레이블이 5단계로 나뉘며, 문장의 길이가 짧고 표현의 뉘앙스가 중요한 과제이다. 모델은 학습이 진행되면서 점진적인 향상을 보였고, 최고 검증 정확도 0.520에 도달하였다. 이는 LoRA + Adapter가 GPT-2의 사전 학습 표현에 태스크 특화된 조정에 영향을 미쳤다는 것을 의미한다.

CFIMDB 결과 해석

CFIMDB는 한글 기반의 이진 감성 분류 데이터셋으로, 리뷰 문장이 비교적 길고 감정이 명확히 드러나는 특징을 갖는다. 해당 데이터셋에서는 학습 초반부터 높은 정확도를 기록했으며, 검증 정확도는 최고 0.967로 수렴하였다. 이는 LoRA와 Adapter가 표현력이 충분한 언어 모델의 특징을 빠르게 조정할 수 있음을 보여준다.

두 데이터셋 간 결과를 비교했을 때, STT는 복잡한 감정 구조로 인해 상대적으로 성능이 낮았고, CFIMDB는 표현이 명확해 빠르게 수렴하였다. 그러나 공통적으로 LoRA와 Adapter의 도입이 baseline에 비해 분류 성능을 향상시켰고 전체 모델을 학습하지 않고도 의미 있는 성능 개선을

달성하였다. 이는 적은 연산 자원과 메모리로도 언어 모델을 도메인에 특화시킬 수 있다는 가능성을 제시하고 다양한 분류 태스크에서의 확정성을 시사한다.

5.2. 패러프레이즈 탐지

본 연구는 의미적 등가성 판단이 중요한 Quora Question Pairs(QQP) 과제를 대상으로, 역번역(back-translation)을 활용한 데이터 증강이 패러프레이즈 탐지 성능에 미치는 영향을 실증적으로 분석하였다. 실험은 사전학습된 언어 모델을 동일하게 사용한 두 조건(기본 모델 vs. 역번역 증강 모델) 간 비교를 통해 수행되었다.

역번역 데이터 증강 적용 모델 결과 해석

역번역 모델은 동일한 모델 구조를 사용하되, 훈련 데이터에 500쌍의 back-translated 문장을 추가하여 학습하였다. 초기 epoch 단계에서는 기본 모델과 성능 차이가 거의 없었지만, 후반부(epoch 4~6)에서 상대적으로 높은 dev accuracy를 보이며 최종적으로 0.899의 정확도를 기록하였다. 이는 단순히 문장 수를 늘리는 것 이상의 효과를 지니며, 문장 구조의 다양성이 모델의 일반화 성능 향상에 기여했을 가능성을 시사한다.

동의어 치환 데이터 증강의 한계점

본 연구에 사용된 QQP 데이터셋은 두 개의 질문이 의미적으로 동일한지 여부를 판단하는 이진 분류 과제를 다룬다. 해당 데이터는 표현의 유사성보다는 의미적 등가성(semantic equivalence)에 기반하여 라벨링되어 있으며, 문장 간의 미세한 의미 차이까지 구별 가능한 정밀한 판별 능력이 요구된다.

이러한 특성으로 인해, 단순한 텍스트 증강 기법이 오히려 성능 저하를 초래할 수 있는 조건이 내재되어 있다. 예를 들어, WordNet 기반의 동의어 치환은 문맥을 고려하지 않은 정적인 어휘 대체를 수행하기 때문에, ‘get’을 ‘take’로, ‘great’을 ‘tremendous’로 치환하는 등의 경우처럼 문장의 의미가 왜곡되거나 어색한 표현이 생성될 수 있다(Wei & Zou, 2019). 이로 인해 원래는 의미가 일치하는 문장쌍이 비문처럼 보이게 되거나, 반대로 의미가 다른 문장이 표면적으로 유사하게 변형되어 레이블 오염이 발생할 가능성이 존재한다.

따라서 QQP와 같이 의미 보존의 정밀성이 핵심인 데이터셋에서는 동의어 치환 텍스트 증강 기법이 모델의 일반화 성능을 오히려 저하시킬 수 있으며, 신중한 품질 관리 및 보완 절차가 수반되지 않는 한 직접적인 적용에는 한계가 존재한다.

5.3. 소네트 생성

본 연구에서는 GPT-2 기반의 언어 생성 모델에 대해 다양한 디코딩 전략을 실험하고, 그 효과를 문자 단위 정합성 지표인 CHRF를 통해 분석하였다. 특히 Top-p(nucleus) sampling과 temperature scaling 기법을 적용한 결과, 기존 baseline보다 CHRF 점수가 약 1.5점 향상되었으며, 이는 모델이 더 유창하고 다양성 있는 표현을 생성하도록 유도되었음을 시사한다.

한편, Beam Search와 같은 결정론적 탐색 방식은 높은 확률의 안정적인 단어 조합을 생성하는 데는 효과적이지만, 생성 문장의 다양성이 제한되고 우연한 문자 일치 가능성이 줄어들기 때문에 CHRF 점수에서는 불리하게 작용하였다. 실제로 Beam Search 적용 시 CHRF 점수는 25.122로, Top-p 방식에 비해 크게 하락하였다.

또한, CHRF와 같은 정량적 평가지표는 참조 텍스트와의 어휘적 유사성만을 반영하기 때문에, 운율(meter), 압운(rhyme), 구문 구조 등 문학적 품질 요소는 평가 대상에 포함되지 않는다. 이는 Beam Search로 생성된 문장이 문학적으로 더 자연스러움에도 불구하고 낮은 점수를 받는 현상의 원인이 된다.

이러한 점을 고려하여 본 연구에서는 CHRF 점수 향상을 중심 목표로 삼았고, 이에 가장 효과적인 Top-p + Temperature 기반 샘플링 전략을 최종 선택하였다. 향후 연구에서는 BLEU, ROUGE 외에도 Rhyme Density, Meter Score 등 문학적 요소를 반영한 다중 평가 지표를 함께 도입하여, 보다 정성적인 평가가 가능하도록 확장할 수 있을 것이다.

6. Conclusion

본 연구에서는 사전학습된 GPT-2 언어모델을 기반으로 다양한 자연어 처리 과제에 대해 성능 향상을 위한 경량화 기법과 디코딩 전략을 적용하고 그 효과를 정량적으로 분석하였다. 실험은 총 세 가지 태스크—감정 분류(SST, CFIMDB), 패러프레이즈 탐지(QQP), 그리고 소네트 생성—에 걸쳐 진행되었다.

감정 분류 과제에서는 GPT-2 모델에 LoRA(Low-Rank Adaptation)와 Adapter를 결합한 경량 파인튜닝 방식을 적용하였다. 전체 파라미터를 학습하지 않고도 의미 있는 성능 향상이 관찰되었으며, 특히 한글 기반 CFIMDB 데이터셋에서는 테스트 정확도가 약 9.8% 향상되었다. 이는 LoRA와 Adapter가 도메인 특화 태스크에서의 표현력 개선에 기여함을 시사한다.

패러프레이즈 탐지 실험에서는 의미적 등가성을 판단하는 QQP 데이터셋을 기반으로, 역번역(Back-Translation) 기법을 활용한 데이터 증강을 적용하였다. 증강된 데이터는 모델의 일반화 성능을 개선하며 baseline 대비 높은 정확도를 달성하였다. 다만 의미 변형에 민감한 데이터셋 특성상, 정제되지 않은 증강 데이터는 오히려 성능을 저하시킬 수 있다는 점에서 품질 관리의 중요성이 확인되었다.

소네트 생성 과제에서는 텍스트 생성 품질을 높이기 위해 다양한 디코딩 전략을 실험하였다. Beam Search는 결정론적 특성으로 인해 생성 다양성이 감소하고 CHRF 점수가

하락(25.122)하는 경향을 보였으며, 반면 확률 기반의 Top-p + Temperature 샘플링 기법은 CHRF 점수를 41.538까지 향상시키는 데 기여하였다. 이 결과는 문학적 텍스트 생성에 있어 확률적 디코딩 전략이 효과적일 수 있음을 보여준다.

전반적으로 본 연구는 LoRA + Adapter 기반의 경량 파인튜닝, 데이터 증강 기법, 그리고 디코딩 전략의 선택이 언어 모델의 downstream task 성능에 실질적인 영향을 줄 수 있음을 실험적으로 검증하였다. 향후 연구에서는 BLEU, ROUGE, Rhyme Density, Meter Score 등 다양한 정성/정량 평가 지표를 함께 활용하여 모델 성능을 다각도로 평가하고, 보다 창의적이고 목적 지향적인 텍스트 생성 모델을 개발하는 방향으로 확장할 수 있을 것이다.

7. 참고문헌

[1] E. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Wang, L. Wang, and W. Chen, “LoRA: Low-Rank Adaptation of Large Language Models,” *arXiv preprint*, arXiv:2106.09685, 2021.

[2] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. de Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly, “Parameter-efficient transfer learning for NLP,” in *Proc. 36th Int. Conf. Mach. Learn. (ICML)*, Long Beach, CA, USA, Jun. 2019, pp. 2790–2799.

[3] R. Sennrich, B. Haddow, and A. Birch, “Improving neural machine translation models with monolingual data,” in *Proc. 54th Annu. Meeting Assoc. Comput. Linguistics (ACL)*, Berlin, Germany, Aug. 2016, pp. 86–96.

[4] R. Xiong, Y. Yang, D. He, K. Zheng, S. Zheng, H. Xing, H. Zhang, and T.-Y. Liu, “On layer normalization in the transformer architecture,” in *Proc. 37th Int. Conf. Mach. Learn. (ICML)*, Vienna, Austria, Jul. 2020, pp. 10524–10533.

[5] J. Wei and K. Zou, “EDA: Easy data augmentation techniques for boosting performance on text classification tasks,” *arXiv preprint*, arXiv:1901.11196, 2019. [Online]. Available: https://arxiv.org/abs/1901.11196

[6] Agnew, E., Qiu, M., Zhu, L., Wiseman, S., & Rudin, C. (2023). The Mechanical Bard: An Interpretable Machine Learning Approach to Shakespearean Sonnet Generation. In Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers) (pp. 1627–1638). Toronto, Canada: Association for Computational Linguistics. https://doi.org/10.18653/v1/2023.acl-short.140

[**7**] Holtzman, A., Buys, J., Du, L., Forbes, M., & Choi, Y. (2020). *The Curious Case of Neural Text Degeneration*. Proceedings of ACL.
https://arxiv.org/abs/1904.09751