LoRA (Low-Rank Adaptation)

-Low-Rank Adaptation of Large Language Models) 2021

논문 방식

기존 모델의 가중치 행렬을 그대로 두고, 해당 위치에 저차원 행렬 A와 B를 추가하여 W+△W = W + BA 형태로 표현함으로써 일부 모듈만들 선택적으로 학습할 수 있도록 한다.

적용 방식

Attention.py, LoRA.py

Class LoRALinear

self.weight = nn.Parameter(torch.randn(out_features, in_features))

self.weight.requires_grad = False

->기존 weight는 학습되지 않도록 동결

self.A = nn.Parameter(torch.randn(r, in_features) * 0.01)

self.B = nn.Parameter(torch.randn(out_features, r) * 0.01)

->논문에서 제안한 저차원 행렬 A,B를 학습 파라미터로 설정

Modules/attention.py 적용 위치 - Query, Key, Value

self.query = LoRALinear(config.hidden_size, self.all_head_size)

self.key = LoRALinear(config.hidden_size, self.all_head_size)

self.value = LoRALinear(config.hidden_size, self.all_head_size)

->기존 Transformer에서 nn.Linear로 구현된 query, key, value대신 LoRALInear로 대체

Adapter

Parameter-Efiicient Transfer Learning,2019

논문 방식

대형 사전학습 모델을 모두 fine-tune하는 방식은 파라미터 수가 너무 많고 비효율적임

이를 해결하기 위한 방식 PETL

- 전체 모델은 동결
- 소수 파라미터만 추가로 학습

대표적인 PETL 기법

- Adapter
- LoRa
- Prefix Tuning, BitFit

즉, 전체 모델은 동결하고, 각 Transforemr 사이에 작은 병목 네트워크를 삽입하여 소수 파라미터만 학습하는 것

적용 방식

Gpt2_layer.py, adapter.py

 # Adapter 정의

self.adapter = Adapter(hidden_size, bottleneck=64)

 # 전체 모델 파라미터 동결

for param in model.parameters():

    param.requires_grad = False

#Adapter만 학습

for name, param in model.named_parameters():

    if 'A' in name or 'B' in name or 'adapter' in name:

        param.requires_grad = True

데이터 증강 기법 (역번역)

Improving neural machine translation models with monolingual data, 2016

논문방식

단일 언어 데이터로 데이터 확장하기 위해 역번역을 사용

번역 모델을 이용해 문장을 타 언어로 번역한 후 다시 원어로 복원

의미는 같지만 표현이 다른 문장을 생성

적용방식

Parphrase_detection.py

Augment_data_with_back_translation()함수

 # 랜덤으로 일부 문장 쌍 선택

selected_indices = random.sample(range(len(train_data)), augment_count)

 # (q1, q2)을 역번역

aug_q1 = back_translate(q1, tokenizer_pivot, model_pivot, tokenizer_back, model_back, device)

aug_q2 = back_translate(q2, tokenizer_pivot, model_pivot, tokenizer_back, model_back, device)

->원래 문장 q1,q2 출력

역번역된 문장 aug_q1, aug_q2

결과 문장이 원문과 같으면 사용하지 않음

```
if result.lower().strip() == text.lower().strip():
    return None
```

Back_translate() 함수 구조

영어 → 프랑스어

```
inputs = tokenizer_pivot([text], ...)
translated = model_pivot.generate(**inputs)
pivot_text = tokenizer_pivot.decode(...)
# 프랑스어 → 영어
inputs_back = tokenizer_back([pivot_text], ...)
back_translated = model_back.generate(**inputs_back)
result = tokenizer_back.decode(...)
```

->Helsinki-NLP/opus-mt-en-fr 모델로 영어→프랑스어

Helsinki-NLP/opus-mt-fr-en 모델로 프랑스어→영어

증강 문장을 원래 학습 데이터에 추가

```
if aug_q1 is not None and aug_q2 is not None:
    augmented_data.append({
        'question1': aug_q1,
        'question2': aug_q2,
        'label': label
    })
```

->모두 성공적으로 역번역된 경우에만 추가

1.역번역 결과가 존재 + NONE x(역번역 결과가 원문과 비슷)

2.역번역 중 에러가 발생하지 x

Top-p Sampling and Temperature Scaling

The Curious Case of Neural Text Degeneration, ACL 2020

논문방식

기존 텍스트 생성 방식은 반복적이고 비자연스러운 텍스트를 생성

->Top-p Sampling: 누적 확률이 상위 p이내인 토큰드만 후보로 삼고, 그 중에서 샘플링

확률이 너무 낮은 단어는 배제, 적절한 다양성과 일관성 유지

적용방식

Sonnet_generation.py

generate() 함수

logits_last_token = logits_sequence[:, -1, :] / temperature

probs = torch.nn.functional.softmax(logits_last_token, dim=-1)

->모델을 출력하고 Softmax 확률 계산

Top-p Sampling 적용

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]

top_p_mask[..., 0] = True

filtered_probs = sorted_probs * top_p_mask

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

->가장 높은 확률의 단어는 포함하고 이후 확률을 다시 정규화

후보 중에서 샘플링

sampled_index = torch.multinomial(filtered_probs, 1)

sampled_token = sorted_indices.gather(dim=-1, index=sampled_index)

->이 과정을 반복해서 전체 시퀀스를 생성

파인튜닝 시도한 방법들

데이터 증강(동의어 치환)

EDA: Easy data augmentation techniques for boosting performance on text classification tasks, 2019

논문 방식

동의어 치환(Synonym Replacement) 방식

1. 단어 선택
   문장에서 stopword(예: is, the, and 등)를 제외한 단어 중 일부를 무작위로 선택합니다.
2. 동의어 검색
   선택된 단어의 동의어(synonym)를 WordNet 등의 사전을 이용해 찾습니다.
3. 치환
   해당 단어를 무작위로 선택된 동의어로 바꿉니다.

이유

NLTK의 WordNet API를 이용해 문장 내 1~2개의 단어를 동의어로 무작위 교체하여 훈련데이터를 확장하는 방식으로 시도되었으나, 문맥에 어색한 단어 교체가 의미를 훼손하거나 GPT-2 모델이 단어 수준보다는 문장 구조의 다양성에 민감하다는 특성

->오히려 정확도가 감소하는 결과를 초래

POS template

The Mechanical Bard: An Interpretable Machine Learning Approach to Shakespearean Sonnet Generation, 2023

논문 방식

1. **POS 템플릿 생성**

- 먼저 기존 셰익스피어 소네트 문장을 **품사 시퀀스(POS sequence)**로 변환합니다.
- 예:
  "The fair lady walks" → [DET, ADJ, NOUN, VERB]
- 이렇게 만든 품사 패턴을 **템플릿(template)**로 저장합니다.

2. **제한된 생성 (Constrained Generation)**

- 모델이 텍스트를 생성할 때,
  각 단어는 **미리 정해진 품사에 맞게** 선택되도록 제한합니다.
- 예: 템플릿이 [ADJ, NOUN, VERB]이면
  첫 단어는 형용사만, 두 번째는 명사만, 세 번째는 동사만 생성 가능.

3. **해석 가능성 확보**

- 이 방식은 단어 선택을 **명시적인 품사 규칙**에 따라 제한하므로,
  사람이 결과를 더 쉽게 해석하고 이해할 수 있습니다.

이유

주어진 데이터 셋에서 POS template을 추출했을 때, 데이터 셋의 크기가 매우 작아서 추상화 단계를 많이 낮춤

->문장 구조가 일치하는 경우 x

Beam search

The Mechanical Bard: An Interpretable Machine Learning Approach to Shakespearean Sonnet Generation., 2023

각 단계마다 고정된 크기의 후보 집합(beam)을 유지하면서 여러 경로를 병렬적으로 탐색하는 것.

논문 방식

1단계 : 초기 단계에서 beam 크기(k)만큼의 후보를 선택

2단계 : 각 후보에서 다음 단계의 가능한 모든 토큰을 확장하여 전체 후보 집합을 만듦

3단계 : 확장된 후보 중 가장 높은 확률을 가지는 상위 k개의 후보만 유지하고 나머지를 제거

4단계 : 이러한 과정을 문장 종료 토큰을 만나거나 최대 길이에 도달할 때까지 반복

이유

1. 결정론적 생성으로 인한 다양성 감소
    - Beam Search는 확률이 가장 높은 시퀀스를 결정론적으로 선택하는 방식이라서, 무작위성이 사라짐.
    - 이 때문에 모델이 유연한 창작을 하지 못하고, 문자열 일치 위주의 답변만 생성해 CHRF 점수가 낮아질 수 있음.
2. 고확률 단어 선택으로 인한 어휘 반복 및 일반화
    - Beam Search는 '안전한' 어휘 조합을 반복해 사용하게 되어, 정형화된 단조로운 텍스트가 생성됨.
    - 이는 참조 텍스트와 달리 어휘적 다양성이 떨어지는 결과를 낳고, 평가 지표에 부정적 영향을 줄 수 있음.
3. CHRF 점수와 인간 평가 사이의 괴리
    - CHRF는 어휘 일치율 기반 지표로, 운율이나 구조적 문학적 요소를 반영하지 않음.
    - Beam Search는 인간이 보기에는 괜찮은 문장을 만들 수 있지만, CHRF 기준에서는 낮은 점수를 받을 수 있음.