Efficient Fine-Tuning of Transformer Models via Compressed Context Vector Caching

김재준, 2025/05/14

1. 서론

- 최근 자연어 처리 분야에서 대규모 언어모델(LLM, Large Language Model)은 다양한 응용 분야에서 혁신적인 발전을 이루고 있음.
  - 수십억에서 수백억 개의 파라미터를 가진 LLM 기반 모델(Foundation Model)을 훈련하거나 특정 도메인에 맞게 파인튜닝(fine-tuning)하는 과정에서 엄청난 계산 비용과 자원을 요구함.
  - 실제로 기반 모델의 훈련에는 수천 개 이상의 GPU 또는 TPU를 병렬로 사용하는 막대한 자원을 필요로 하며, 이는 현실적인 자원 제약으로 인해 많은 기업 및 연구기관에 큰 부담이 되고 있음.

- 이러한 문제를 해결하기 위해 최근 Parameter-Efficient Fine-Tuning(PEFT)이라는 개념이 등장.
  - PEFT는 전체 파라미터를 변경하는 대신 일부 파라미터만 선택적으로 업데이트하는 방법을 통해 학습 비용을 줄이고 효율적으로 모델을 특정 태스크나 도메인에 적응시키는 전략임.
  - 대표적인 PEFT 방법으로는 LoRA(Low-Rank Adaptation), Adapter-Tuni ng, Prefix-Tuning 등이 제안되었으나, 기존 기법들은 특정 모듈에만 적용 가능하여 범용성이 부족하거나, 불필요한 차원까지 연산을 수행하는 비효율성이 존재하는 한계점이 있음.

1. 서론

- 기존 트랜스포머 모델의 어텐션 값(Z)은 각 입력 토큰과 문장 전체와의 관계성을 나타내지만, 매번 Self-Attention 연산을 반복 수행함에 따라 높은 계산 및 메모리 부담이 발생함.

- 본 연구에서는 최초 계산된 어텐션 값(Z)을 마지막 디코더에서 추가적인 Linear Layer를 통해 Context Vector를 생성함.
  - 이 Context Vector는 각 토큰의 상대적 중요도와 관계성을 유지하면서 정보를 압축한 형태의 벡터이며,
    이는 의미적으로 유사한 토큰들을 그룹화하거나 유사도 측정에 사용됨.

  - 캐쉬 알고리즘을 활용하여 반복적으로 나타나는 토큰의 어텐션 값(Z)를 재사용하여 중복된 연산을 최소화하며,
    문맥 변화 탐지 및 캐시 갱신 알고리즘을 통해 캐시된 Context 벡터와 새로운 Context 벡터간의 유사도를 측정하여 사
    전에 설정된 임계값과 비교하여 동적인 갱신여부를 판단하는 알고리즘을 이용해 효율적인 캐시 관리를 기법을 제공함.

2. 본론

2.1 관련 연구

- 대규모 언어모델을 특정 작업 또는 특정 도메인에 적합하게 미세조정하기 위한 Parameter-Effic ient Fine-Tuning(PEFT) 기법은 최근 활발히 연구되고 있음.
  - LoRA (Low-Rank Adaptation): 모델의 가중치 업데이트를 저차원 행렬로 제한하여 효율성을 높이는 기법임.
  - Adapter-Tuning: 추가적인 경량 모듈(Adapter)을 모델 내에 삽입하여, 소규모 파라미터만을 업데이트하는 방식임.
  - Prefix-Tuning: 토큰 앞부분에 추가적인 학습 가능한 벡터를 삽입하여 모델의 성능을 향상시키는 기법임.
  - 이러한 방법들은 전체 파라미터를 갱신하지 않고 일부 파라미터만 업데이트하여 효율성을 높일 수 있으나, 특정 모듈 또는 일부 레이어에만 적용 가능하여 범용성이 제한적임.
  - 또한, 모델 차원의 전체적 활용 최적화가 어렵고, 계산 자원이 불필요한 차원까지 낭비되는 문제점이 발생한다. 토큰 단위의 세밀한 연산 효율화를 위한 최적화는 상대적으로 미흡함.

2.2 제안 알고리즘의 기본 개념

- 본 연구에서 제안하는 어텐션 값(Z)의 개념은 Transformer 기반의 언어 모델에서 입력된 토큰들의 관계를 정량적으로 나타내는 벡터 값임.
  - Transformer는 입력 문장 내의 각 토큰이 서로에게 얼마나 중요한지(어텐션 가중치)를 계산하여 문맥을 이해하는데, 이때 계산되는 어텐션 값(Z)은 토큰들 간의 상호작용을 담고 있어 문맥 이해와 의미 파악에 핵심적인 역할을 수행함.

  - 특히, 자연어처리(NLP) 모델의 성능과 정확성은 이러한 어텐션 값의 정확한 계산에 크게 의존합니다. 일반적으로 동일한 문맥 내에서 반복적으로 나타나는 토큰은 문맥에서의 역할이나 의미적 중요도가 크게 달라지지 않습니다. 따라서 동일 문맥에서 반복 등장하는 토큰에 대한 어텐션 값(Z)은 최초 계산 이후 거의 변화가 없거나 그 변화 폭이 매우 작다고 가정할 수 있음.

  - Attention 값(Z)을 압축된 형태인 Context Vector로 저장(캐싱)하여, 이후 비슷한 문맥에서 동일한 Attention 계산을 반복하지 않고, 캐쉬된 Context Vector를 재사용하여 계산 비용을 현저히 줄이는 것을 목적으로 함

  - 캐싱 기법을 도입하면 특히 긴 문장이나 문서 단위 입력과 같이 반복적인 토큰이 자주 나타나는 상황에서 계산 비용이 크게 감소하며, 모델의 처리 속도와 효율성이 현저히 개선됩니다. 이를 통해 실시간으로 처리해야 하는 환경에서도 Transformer 모델의 응답 시간을 크게 단축할 수 있음.

2. 본론

2.3 어텐션 값(Z)기반 압축된 문맥 벡터(Compressed Context Vector) 추출 방식

- 본 연구에서는 최초 계산된 어텐션 값(Z)을 추가적인 Linear Layer를 통해 정보 손실을 최소화하는 Context Vector를 재생성함.
  - 이 과정에서 얻어지는 어텐션 값(Z)은 각 토큰의 상대적 중요도와 관계성을 나타내는 밀도 높은 벡터로 표현됩니다. 이러한 Z 값은 문맥에서 토큰이 가지는 의미적 및 문법적 정보를 압축적으로 나타냄.

- 이런 Context 벡터는 다음과 같은 방식으로 활용 가능함.
  - 문맥 벡터로서 의미적으로 유사한 토큰들을 그룹화하거나 유사도 측정에 활용합니다. 문장 혹은 문단 단위의 전체 의미를 파악하는 데 효과적으로 사용됨.
  - 기존 방식에 비해 빠르게 문맥을 반영하여, 다음 토큰을 예측하거나 응답을 생성하는 작업에 적합합니다. Context 벡터를 사용하면 긴 문장이나 긴 문서에서 반복적으로 계산되는 정보를 효율적으로 압축하고 표현할 수 있음.
  - Context 벡터는 토큰 간의 관계를 직접적으로 내포하고 있기 때문에, 후속 작업에서 추가적인 계산 비용을 줄이면서도 정확성을 유지하거나 향상할 수 있음.
  - 특히, 문맥 정보가 중요한 질의응답 시스템이나 실시간 대화 모델과 같은 분야에서 Context 벡터 기반 표현은 더 빠르고 효율적인 성능을 제공할 수 있음.

2.3.1 Linear Layer를 이용한 Compressed Context 벡터 생성

- Compressed Context Vector(이하는 CCV)는 마지막 디코드에서 생성된 Attention Value(Z)를 추가적인 Linear Layer를 통과해서 얻어진 압축된 문맥 벡터이다.

$$CCV = W_c Z + b_c$$

  - 여기서 Z는 기존 Attention 값이며, 크기는 4096 * N차원입니다. N은 시퀀스 길이
  - 새로 제안한 CCV는 2048 * N차원으로, 기존 Attention 값(Z)의 의미적 정보를 유지하면서 더 작게 압축한 벡터입니다.

- CCV Operaton

1. 초기 단계
   - 처음 문장을 입력 받으면 기존 디코더에서 Self-attention 메커니즘으로 Attention값을 계산합니다.
   - 최초 계산 후 마지막 디코더 SA에서 출력된 Attention Value(Z)를 Linear Layer를 통해 CCV를 생성하여 캐싱함. (캐싱 유지 관리 기법은 2.4 절에 따름)
2. 유사한 문맥
   - 캐쉬된 CCV와 input token과 비교하여, 유사한 문맥이면 , 캐시된 CCV를 불러와서 재사용함.
3. 문맥 변화 탐지 및 캐시 갱신 알고리즘
   - 문맥 변화(Context Drift)가 탐지되면 2.5절의 문맥 변화 탐지 및 캐시 갱신 알고리즘을 적용함

2.4 압축된 문맥 벡터 캐싱 메커니즘 설계

- 캐싱된 데이터의 효율적인 관리를 위해서는 데이터의 생명 주기와 접근 빈도를 고려한 관리 전략을 수립해야함.
  - 예를 들어, 반복 빈도가 높은 토큰에 대한 어텐션 값(Z)은 장기간 유지하면서, 접근 빈도가 낮은 어텐션 값은 빠르게 제거하는 방식으로 관리 효율성을 높일 수 있음.
  - 캐싱된 어텐션 값(Z)을 관리할 때는 다음과 같은 효율적인 접근 방식을 적용할 수 있습니다. LRU(Least Recently Used) 캐시 : 가장 오랜 시간 동안 접근되지 않은 데이터를 우선적으로 제거하는 방법임. 최근 자주 사용된 데이터만을 유지하기 때문에 메모리 효율성을 높이고, 자주 사용되는 어텐션 값에 대한 접근 속도를 높임.
  - 메모리 관리 : 제한된 메모리 자원을 효율적으로 사용하기 위해 어텐션 값의 크기, 중요도, 빈도를 분석하여 중요도가 낮거나 자주 참조되지 않는 데이터부터 메모리에서 제거하는 방식을 사용할 수 있음.
  - 캐시 만료 정책 : 시간 기반 또는 사용 빈도 기반으로 데이터의 만료 시점을 설정하여, 불필요한 데이터가 오래 머무르지 않도록 관리함. 이로 인해 메모리 사용 효율이 증가하고 시스템의 성능도 유지시킬 수 있음.

2. 본론

2.4 압축된 문맥 벡터 캐싱 Operation.

- 캐싱 Operation의 효율적인 프로세스는 다음과 같이 진행함.
  1) 캐시 조회(Cache Lookup): 새로운 어텐션 값을 계산하기 전, 캐시에 CCV가 존재하는지 먼저 확인.
  2) 캐시 히트(Cache Hit): 만약 캐시에 CCV가 존재하면, 문맥 변화가 있는지 없는지를 확인하여, 문맥 변화가 없으면 계산 없이 바로 캐싱된 CCV를 사용하고, 문맥 변화가 있으면 문맥 변화 탐지 알고리즘에 의해 캐시된 CCV를 업데이트함.
  3) 캐시 미스(Cache Miss): 캐시에 데이터가 존재하지 않을 경우, 새롭게 어텐션 값을 계산한 후 CCV를 캐시에 저장함.
  4) 캐시 업데이트(Cache Update): 문맥 변화 탐지 알고리즘에 의해 문맥이 변경되었으면 새로 Attention Value(Z)를 생성한 후 CCV를 업데이트함.

2. 본론

2.5 문맥 변화 탐지 및 캐시 갱신 알고리즘

- 문맥 변화(Context Drift)란 모델이 입력받는 텍스트의 문맥이 이전의 텍스트와 현저히 다를 때 발생하는 현상임.
- CCV값은 문맥에 의존적이므로 문맥 변화가 발생하면 이전에 캐시된 CCV값은 부정확한 결과를 초래할 수 있음.
  따라서 문맥 변화를 탐지하고 적절히 캐시를 갱신하는 것이 모델의 성능과 정확도를 유지하는 데 매우 중요함.

- 문맥 변화 탐지 알고리즘은 다음과 같다.
  1. 새로운 입력 토큰이 들어옴
  2. 초기 5~ 10개 디코더 레이어에서 Attention Value(Z) 를 추출
  3. Linear Layer를 통과시켜 CCV와 동일한 2048차원으로 변환(Z' compressed)
  4. Cosine Similarity로 캐시된 CCV와 비교함
     -. 임계값(Threshold)과 비교하여 높으면 캐시된 CCV 재사용함
     -. 임계값보다 낮으면 마지막 디코더에서 CCV를 계산함.
  5. 문맥 변화가 감지되면 새로운 CCV를 생성 및 저장함.

2.5 문맥 변화 탐지 및 캐시 갱신 알고리즘

- 문맥 변화를 탐지 및 캐싱을 위한 임계값(Threshold) 설정
  - Cosine Similarity기반 문맥 임계값은 다음과 같은 기준을 동적으로 정한다.

1. 공격적인 캐싱 : 실시간 응답이 중요한 경우(예: 챗봇, 검색엔진 등)
   -. Threshold : 0.7 이상, 계산 비용을 크게 절감하여 빠른 응답 가능
2. 균형잡힌 캐싱 : 일반적인 NLP 모델 최적화
   -. Threshold : 0.8 이상
3. 보수적인 캐싱 : 정확성이 중요한 경우(예: 기계 번역, 문서요약등)
   -. Threshold : 0.9 이상

2.6 계산 비용(Computational Cost) 및 메모리 사용량 분석

2.6.1 표기 정의

L_{total} : 전체 디코더 레이어 수 (예: 32)

L_{eff} : CCV 생성에 실제로 사용하는 초기 디코더 레이어 수

$L_{remain} = L_{total} - L_{eff}$: 나머지 레이어 수

N: 입력 시퀀스 길이 , d_1: 기존 어텐션 값 차원(예; 4096), d_2 : 압축된 문맥 벡터(CCV) 차원(예:2048)

p : 캐시 히트 확률, 1-p : 캐시 미스 확률

- 각 연산에 대한 비용을 다음과 같이 가정함.

  T_{sa} : 한 디코더 레이어에서 Self-Attention 연산 비용 $\approx O(N^2 \times d_1)$

  T_l : 마지막 레이어의 어텐션 Z를 선형 변환하여 CCV를 생성하는 비용 $\approx O(N \times d_1 \times d_2)$

  T_{cos} : 캐시 히트 시, 저장된 CCV 와 새 입력간의 유사도 계산 비용 $\approx O(N \times d_2)$

2.6.2 기존 Transformer의 비용

- 전체 디코더 레이어(32)에서 모두 Attention 을 수행하면

  $T_{tran} = L_{total} * T_{sa} = 32 * O(N^2 * d^2)$ 이고, 각 레이어마다 $O(N * d_1)$의 메모리를 사용함.

2.6.3 제안된 CCV 방식의 비용

- 초기 L_{eff} 레이어(예: 5~ 10개) : 이 단계에서는 기존 Self-Attention을 수행하여, 마지막 레이어에서 Z를 계산한 후 선형 변환 T_l 을 통해 CCV를 생성한다. 비용은 $T_{init} = L_{eff} * T_{sa} + T_l$ 임.
- 나머지 L_{remain} 레이어에 대한 처리
  - 캐시 히트(p) : 각 레이어에서 CCV와 새 입력간의 유사도 평가만 수행하므로, 비용은 T_{cos} per layer
  - 케시 미스(1-p) : 해당 레이어에서는 새로 Self-Attention 연산과 CCV 업데이트를위한 선형 변환 T_l까지 수행하므로, 비용은 $T_{sa} + T_l$ 임. 따라서 나머지 레이어에 대한 평균 비용은

    $T_{remain} = L_{remain} * [p * T_{cos} + (1 - p) * (T_{sa} + T_l)]$
- 전체 CCV 방식의 총 비용은

  $T_{ccv} = L_{eff} * T_{sa} + T_l + L_{remain} * [p * T_{cos} + (1 - p) * (T_{sa} + T_l)]$

3. 결론

- 본 논문에서는 Transformer 기반 모델에서 발생하는 계산 비용과 메모리 사용 문제를 해결하기 위해 어텐션 값(Z)을 압축한 CCV(Compressed Context Vector)를 도입하고, 이를 효과적으로 캐싱하는 방법을 제안하였음.
  - 제안 방식은 계산 효율성을 개선하고, 메모리 사용을 최적화하며, 문맥 이해의 정밀도 유지하면서 불필요한 연산을 줄일 수 있었음.
  - 제안한 CCV 캐싱 기법은 파인 튜닝 단계에서도 실시간 응답 및 추론 속도를 향상시키고, 자원 효율성 및 비용을 절감하고, 동적 문맥 변화에 대응하는 할 수 있도록 하였음.
- 향후 연구에서는 AI Agent와의 결합을 통해 실시간 사용자 피드백이나 문맥 변화 상황을 감지하여 캐싱 전략을 동적으로 최적화할 수 연구를 진행할 예정입니다.