거의 다 됐습니다. 지금 가입해서 Notion으로 구축을 시작하세요. 가입 또는 로그인하기

📚 LoRA: Low-Rank Adaptation of Large Language Models

---

1. 서론 (Introduction)

- 최근의 대형 언어모델(GPT-3 등)은 **파라미터 수가 수십~수천억**에 달합니다.
- 이런 모델을 특정 업무(Task)에 맞게 조정하려면 보통 **전체 파라미터를 다시 학습(fine-tuning)** 해야 하는데,
  - 이는 **막대한 계산 자원**, **시간**, **저장공간**이 필요하며,
  - 여러 작업(Task)을 동시에 다루기에는 **비효율적**입니다.
- LoRA(Low-Rank Adaptation)는 이 문제를 해결하기 위한 **경량화된 학습 방법**입니다.

---

2. 문제 정의 (Problem Statement)

- 전통적인 fine-tuning은 **모든 파라미터를 다시 조정**합니다.
  - 이러면 작업마다 하나의 '거대한 사본'이 생기고, **수백 GB~TB급 모델이 여러 개** 생깁니다.
- 목표는 이렇습니다:

> 기존 모델은 그대로 두고, 꼭 필요한 변화만 소량의 파라미터로 표현해 효율적으로 학습하기

---

3. 기존 방식들의 한계 (Are Existing Solutions Good Enough?)

3.1 어댑터 방식 (Adapter Layers)

- Transformer 블록 사이에 작은 추가 레이어를 넣고, 이 부분만 학습

- **단점**: 모델 구조가 바뀌므로
  - 병렬 처리에 불리하고
  - **추론 속도 증가** 및 **메모리 병목**이 발생할 수 있음

3.2 프롬프트 기반 방식 (Prompt/Prefix Tuning)

- 모델 입력에 **학습 가능한 특수 토큰**을 추가해서 조정
- **단점**:
  - 최적화가 어려움 (학습이 잘 안 됨)
  - 입력 길이를 차지하므로 긴 문장에 불리
  - 효과가 데이터셋/작업에 따라 **불안정**

---

4. 제안 방식: LoRA (Our Method)

4.1 핵심 아이디어: 변화는 '저차원'에서 일어난다

- 대형 모델의 변화는 사실상 **일부 중요 방향(패턴)**만 바뀝니다.
- 즉, 모델 변화는 **수학적으로 저차원 구조** (Low-Rank) 라고 볼 수 있습니다.
  - 복잡한 변화도 몇 개의 '방향'만 잘 조절하면 대부분 커버됨

▶️ LoRA는 이렇게 해결합니다:

> 기존 모델 파라미터는 고정하고,
>
> **변화량만 별도의 저차원 행렬로 학습**한다!

- 수식으로는 $\Delta W \approx B \times A$

  (ΔW는 바뀌는 값, A와 B는 작고 얇은 행렬)

- **A는 압축**, **B는 복원** 역할
  - 학습 대상은 A와 B뿐이므로 매우 가볍고 빠름

4.2 어떻게 작동하나?

기존 방식

- 모델은 입력 벡터 x에 대해 `h = Wx` 연산 수행 (W: 가중치 행렬)

LoRA 방식

- `h = Wx + B(Ax)`

  즉, 기존 W는 고정, 변화만 `BAx`로 추가됨

- **W는 건드리지 않음**,

  변화는 **B와 A라는 작고 학습 가능한 행렬**로 따로 표현

🔧 파라미터 크기(r)에 따라...

- r = 1~8만 써도 충분히 좋은 성능
- r이 작을수록 압축률 높고 학습 부담 낮음

비유 🍺: 커피 레시피

- W = **기본 커피 레시피**
- LoRA의 BA = **시럽**

  → 커피 자체는 그대로 두고, 시럽만 바꿔서 다양한 맛 제공

  → 원하는 작업마다 다른 시럽만 추가해주면 됨

---

5. 실험 결과 요약 (Empirical Results)

- 다양한 모델(RoBERTa, DeBERTa, GPT-2, GPT-3)에 LoRA 적용

- 결과:
  - **성능은 기존 fine-tuning 수준 유지**
  - 학습 파라미터 수는 최대 **10,000배 감소**
  - 추론 속도 **변화 없음**
  - 저장 용량도 수십 MB로 감소

> GPT-3 175B 기준
>
> 기존: 1.2TB 메모리 사용
>
> LoRA: 350GB로 감소, 속도는 그대로!

---

6. 왜 LoRA는 잘 작동할까? (Why Does LoRA Work So Well?)

6.1 어디에 적용해야 효과적일까?

- Transformer의 **Self-Attention 가중치 중 Wq, Wv**에 적용하는 것이
  파라미터 효율성과 성능 측면에서 최적

6.2 r 값은 얼마나 커야 할까?

- 대부분 작업에서 **r = 1~4**로 충분
- 모델이 변화하는 방향 자체가 **고차원일 필요가 없음**을 의미

6.3 ΔW는 기존 W와 어떻게 다른가?

- ΔW는 기존 W의 **덜 강조되던 유용한 방향성**을 확대함
- 즉, pre-trained 모델이 **이미 갖고 있는 가능성**을
  잘 꺼내 쓰도록 도와주는 역할

---

7. 결론 및 의의 (Conclusion & Significance)

- LoRA는 대규모 언어모델을 다양한 작업에 쉽게 적용할 수 있게 해줌
  - **모델 공유 구조**: W는 공유, 작업마다 A와 B만 교체
  - **저장 효율**, **메모리 절감**, **속도 저하 없음**
- 다양한 학습 기법과도 **혼합 가능**
  - 예: Prefix Tuning + LoRA 조합 → 성능 보완 가능성 높음

📌 핵심 비교 요약

ne-Tuning	LoRA
전체 학습)	0.01% 수준
B~TB	수십 MB
	동일
델 교체	A, B 행렬만 교체
	없음 (기존 W 그대로)
능	동등하거나 우수

🧪 r 값 변화에 따른 성능 변화 분석

- *r (랭크)**는 LoRA의 핵심 하이퍼파라미터로,

  변화량(ΔW)을 얼마나 **세밀하게 표현할 것인지**를 결정합니다.

- 📉 **r이 작을수록**:
  - 학습해야 할 파라미터 수 ↓
  - 계산 비용 ↓
  - 단, 너무 작으면 성능이 저하될 수 있음

- 📈 **r이 클수록**:
  - 표현력은 좋아지지만,
  - LoRA의 경량화 장점은 줄어듦
- ⛳ **논문 결과 요약**:
  - r = 1~4 수준에서 대부분의 자연어처리(Task)에서

    기존 fine-tuning과 **동등한 성능**을 보임
  - 특정 복잡한 작업(예: long-form generation)에서는

    r = 8~16이 약간 더 안정적일 수 있음

> ✅ 실전 팁:
>
> **작업의 복잡도에 따라 r 값을 조정**하면 성능과 효율의 균형을 잡을 수 있음

💡 LoRA + 다른 방법 (Adapters, Prefix 등) 조합 가능성

LoRA는 구조상 **기존 모델 구조를 거의 건드리지 않기 때문에**

다른 경량화 방법들과 **병렬 또는 혼합 적용**이 가능합니다.

🧬 가능한 조합 예시:

1. **LoRA + Adapter Layers**
   - LoRA는 self-attention 부분에 적용
   - Adapter는 feedforward 부분에 적용
   - → 서로 다른 부분을 학습하므로 **보완적 역할** 가능
2. **LoRA + Prefix/Prompt Tuning**
   - Prompt는 입력 쪽, LoRA는 내부 연산에 적용
   - 특히 다중 작업(Multi-task) 환경에서 **유연성 극대화**
3. **LoRA + Quantization**
   - 모델 전체는 양자화(저정밀 연산)로 압축