



# Day 1: Fine-tuning

거대 AI를 내 손안으로



**LoRA**

변화량 기록



**QLoRA**

모델 압축



**RAFT**

현실 학습

💡 오늘은 작은 GPU로도 거대 모델을 학습하는 이론을 배웁니다



## 오늘의 여정

### 1. 문제 발견

Fine-tuning은  
왜 어려운가?

메모리 부족  
시간 오래 걸림  
비용 많이 듦

### 2. 해법 1

LoRA

변화량만 학습

QLoRA

모델 압축

### 3. 해법 2

RAFT 데이터

현실적인  
학습 방식

### 4. 실전

학습 과정

Loss 이해

평가 방법

ROUGE

Embedding

💡 문제 인식 → 해법 탐색 → 실전 적용 순서로 진행됩니다



## 학습 방법

### 이 강의 (이론)

왜 이 기술이 필요한가?  
어떤 원리로 작동하는가?  
다른 방법과 무엇이 다른가?

개념과 원리 이해

### 노트북 (실습)

코드로 직접 실행  
데이터 처리부터 평가까지  
결과를 눈으로 확인

Prerequisites 6개 + Main 4개

 강의로 이론을 이해하고, 노트북으로 직접 실습합니다



## 출발: 우리가 가진 것

사전학습된 거대 모델 EXAONE 3.5

### 모델이 아는 것

일반적인 언어 지식  
광범위한 상식

### 우리가 원하는 것

내 도메인 특화 지식  
내 데이터로 맞춤 학습

 거대 모델을 내 데이터에 맞게 조정하고 싶다!



## 문제: Fine-tuning의 벽

전통적인 방법: 모든 파라미터 수정

### 파라미터란?

모델의 기억 = 수십억 개의 숫자

각 숫자가 학습된 지식 조각

비유: 백과사전의 모든 글자

### Full Fine-tuning 문제

❌ 메모리 부족 (GPU 책상 작음)

❌ 시간 오래 걸림

❌ 비싼 GPU 여러 대 필요

💡 거대한 백과사전 전체를 다시 쓰는 것처럼 비효율적



## 핵심 질문

**정말 모든 걸  
바꿔야 할까?**

대부분의 지식은 그대로 두고  
필요한 부분만 수정하면?

원본은 고정 (frozen)  
변화량만 별도 기록

💡 이것이 LoRA 아이디어의 출발점입니다



# LoRA의 발견

변화량을 별도로 기록하자

**원본  $W_0$**

사전학습된 가중치

고정 (frozen)

학습하지 않음

**변화량  $\Delta W$**

추가로 학습할 내용

학습 가능 (trainable)

따로 저장

사용 시:  $W_0 + \Delta W$

💡 비유: 책을 다시 쓰지 말고, 수정 메모만 붙이기



# Low-Rank의 의미

변화량을 압축 저장

## 직접 저장하면

큰 변화량  $\Delta W$  전체 저장  
여전히 메모리 많이 필요  
비유: 거대한 메모판

## 분해하면

$\Delta W = B \times A$  (작은 행렬 2개)  
메모리 대폭 절감  
비유: 작은 메모지 여러 장

💡 Rank = 정보의 본질적 차원. 대부분의 변화는 저차원으로 표현 가능



## 행렬 분해 시각화

예:  $1000 \times 1000$  행렬을  $\text{rank}=64$ 로 분해



**LoRA 저장: 128K 파라미터**  
 $64K + 64K = 128,000$ 개

**직접 저장: 1M 파라미터**  
1,000,000개 (87% 절감!)

💡 행렬 곱셈으로 복원 가능하므로 작은 B, A만 저장하면 됨



## 하이퍼파라미터 $r$ (rank)

### 메모지 크기 결정

#### $r$ 이 크면

- ✓ 표현력 높음
- ✓ 복잡한 변화 가능
- ✗ 메모리 많이 필요

#### $r$ 이 작으면

- ✓ 메모리 효율적
- ✓ 빠른 학습
- ✗ 표현력 제한

💡 균형점 찾기가 중요! 실습에서는  $r=64$  사용



# 하이퍼파라미터 alpha

변화량 조절 나사

$$\text{실제 변화량} = (\text{alpha} / r) \times B \times A$$

**alpha 크면**

변화 폭이 커짐

**alpha 작으면**

안정적, 보수적

💡 보통  $r$ 의 2배 정도로 설정 ( $r=64$ 면  $\text{alpha}=128$ )



## 어디에 LoRA를 적용할까?

Attention 메커니즘이 핵심

### Q (Query)

"무엇을 찾을까?"

질문 만들기

### K (Key)

"어디에 있을까?"

위치 찾기

### V (Value)

"무엇을 가져올까?"

정보 추출

### O

출력  
조합

💡 이 4가지를 조정하면 모델이 "주목"하는 방식이 바뀝니다



## LoRA의 장점



### 메모리 절감

작은 GPU에서도 가능  
변화량만 저장



### 빠른 학습

일부만 학습  
시간 단축



### 유연성

여러 어댑터 제작  
상황별 교체 가능



### 작은 배포

어댑터만 공유  
베이스 모델 재사용

💡 이 4가지 장점이 LoRA를 인기 있게 만든 이유입니다



## LoRA의 한계

변화량은 작아졌지만  
원본 모델  $W_0$ 는 여전히 크다

원본 모델을 메모리에  
올리는 것 자체가 부담  
여전히 큰 GPU 필요

해결책은?  
→ 모델 자체를 압축  
QLoRA 등장!

💡 LoRA + 양자화 = QLoRA



## Section 1 정리

### 핵심 개념

- ✓ Full Fine-tuning: 모든 파라미터 수정 → 메모리 부담 큼
- ✓ LoRA 아이디어: 원본 고정 + 변화량만 별도 기록
- ✓ Low-Rank: 변화량을  $B \times A$ 로 분해하여 압축 저장
- ✓ 한계: 원본 모델은 여전히 크다

**다음: QLoRA로 모델 자체를 압축하기**

💡 LoRA = 변화량만 학습하는 효율적인 방법



## 새로운 문제

**LoRA로 학습은 가벼워졌지만  
모델 자체를 메모리에 올리는 게 힘들다**

비유: 가구를 옮기고 싶은데  
문이 좁아서 가구가 들어가지 않는 상황

💡 해결책: 가구를 분해(압축)해서 들여놓기



# 양자화 (Quantization)

정밀도를 포기하고 공간 확보

## 32-bit (원래)

매우 정밀한 소수  
3.141592653589...  
공간 많이 차지

## 4-bit (양자화)

대략적인 값  
3.14  
공간 절약

💡 비유: 소수점 여러 자리 → 반올림해서 정수로



# 어떻게 압축할까?

숫자 표현 범위 줄이기

32-bit: 약 40억 가지 다른 숫자 표현 가능

4-bit: 16가지 숫자만 표현 가능

**문제: 어떤 16가지를 선택할까?**

💡 비유: 수천 가지 색 → 16색 팔레트. 어떤 색을 선택할지가 중요



## NF4: 똑똑한 압축

Normal Float 4-bit

### 균등 양자화

-8, -7, -6, ..., 0, ..., 7

간격이 모두 같음

단순하지만 비효율

### NF4

가중치 분포에 최적화

자주 나오는 값에 집중

정보 손실 최소화

💡 비유: 자주 쓰는 색을 팔레트에 더 많이 넣기



## QLoRA = LoRA + 양자화

### 두 기술의 결합

#### 원본 모델 $W_0$

4-bit로 압축 (NF4)

고정 (frozen)

메모리 75% 절감

#### LoRA 어댑터 $B \times A$

16-bit 유지 (BF16)

학습 (trainable)

rank=64, alpha=128

**추론: 압축된 모델 + 정밀한 어댑터**

💡 원본은 4-bit 압축, 어댑터는 16-bit 정밀도 유지



# Double Quantization

## 압축의 압축

양자화에도 메타데이터가 필요 (스케일, 오프셋 등)

메타데이터도  
양자화해서 저장

추가 메모리  
절감 효과

💡 비유: 압축 프로그램을 다시 압축하기

## QLoRA의 장점

### 대폭 압축

작은 GPU 활용  
누구나 접근 가능

### 성능 유지

약간의 손실  
실용적 수준

### 비용 절감

비싼 GPU 불필요  
전기료 감소

### 민주화

개인도 학습  
연구 활성화

 QLoRA 덕분에 개인 컴퓨터에서도 거대 모델 학습 가능



## Section 2 정리

### 핵심 개념

- ✓ 양자화: 정밀도 포기로 메모리 절감 (32-bit → 4-bit)
- ✓ NF4: 가중치 분포에 최적화된 양자화 방식
- ✓ QLoRA: 압축된 모델(4-bit) + 정밀한 어댑터(32-bit)
- ✓ 작은 GPU에서도 거대 모델 Fine-tuning 가능

**다음: RAFT 데이터로 무엇을 학습할까?**

💡 QLoRA = LoRA + 양자화로 메모리 문제 해결



## 데이터 이야기 시작

모델은 준비됐다 (QLoRA)  
무엇을 가르칠까?

일반적인 Q&A?  
→ 단순 암기  
새로운 질문엔 약함

현실적인 시나리오?  
→ 검색 결과 보고 답하기  
일반화 능력↑

💡 RAFT: Retrieval Augmented Fine-Tuning



# RAG의 한계

## Retrieval-Augmented Generation

### RAG 방식

질문마다 DB 검색  
문서 찾아서 답변  
✗ 느림 (매번 검색)  
✗ 검색 의존

### 아이디어

자주 묻는 내용은  
모델이 기억하면?  
✓ 빠름 (검색 불필요)  
✓ 도메인 특화

💡 RAG 결과를 학습 데이터로 만들어 Fine-tuning → RAFT



# RAFT의 핵심 아이디어

## 검색 시뮬레이션

질문 + 여러 문서 → 정답 찾기 (마치 검색 엔진처럼)

### 학습 데이터

검색 결과처럼 구성

### 학습 과정

문서 보고 답하는 법

💡 이 과정 자체를 학습 데이터로 만든다



# Oracle 문서

정답이 있는 문서

## 특징

- ✓ 질문에 대한 답을 포함
- ✓ 1개만 제공 (golden document)
- ✓ 모델이 이걸 찾아서 답해야 함

💡 비유: 시험 문제집의 정답 페이지



# Distractor 문서

방해꾼 (산만하게 하는 것)

## 특징

- ✓ 관련 없거나 오해를 유발하는 문서
- ✓ 여러 개 섞여 있음
- ✓ 현실의 검색 결과를 모사

💡 비유: 시험에 섞인 낚시 문제, 헛갈리게 만드는 오답 선지



## 왜 Distractor가 필요한가?

### Oracle만 학습하면

- ✗ 항상 좋은 문서만
- ✗ 노이즈 대응 못함
- ✗ 현실과 괴리

과적합 위험

### Distractor 섞으면

- ✓ 노이즈 섞인 상황
- ✓ 좋은 문서 찾는 법
- ✓ 현실적 시나리오

일반화 능력↑

💡 현실의 검색은 완벽하지 않다. 관련 없는 문서도 많이 나온다



## RAFT 학습 과정

### 1. 문서 읽기

Oracle +  
Distractors  
여러 개 제시

### 2. 판단 학습

어떤 문서가  
유용한가?  
어떤 건  
무시해야 하나?

### 3. 답변 생성

Oracle 기반  
정확한 답변

### 반복

다양한  
상황  
학습

다음: 데이터 품질 확인

💡 모델이 스스로 유용한 정보를 선별하는 법을 배운다



## 데이터 품질의 중요성

Garbage In, Garbage Out

### 나쁜 데이터

- ✗ 결측치 많음
- ✗ 중복/오류
- ✗ 불균형
- 나쁜 모델

### 좋은 데이터

- ✓ 완전한 정보
- ✓ 정제된 품질
- ✓ 적절한 분포
- 좋은 모델

💡 학습 전 데이터 검증은 필수!



# 스키마 검증

## 데이터 구조 확인

### 필수 컬럼

question 존재?  
answer 존재?  
documents 존재?

### 데이터 타입

문자열인가?  
리스트인가?  
타입 일관성

### 구조 일관성

모든 샘플 동일 구조

### 빠른 발견

초기 단계에서 오류 차단

💡 스키마 불일치는 학습 중 에러의 주요 원인



## 결측치 & 중복 처리

### 결측치 (Missing Values)

빈 문자열 ""

None, null

처리: 제거 or 기본값

학습 오류 유발

### 중복 (Duplicates)

같은 질문/답변

과적합 위험

처리: 제거

데이터 편향

💡 실습 노트북 02에서 자동 검사



## 토큰 길이 분석

### Distribution 확인

#### 평균 길이

전체 평균  
너무 길면?  
메모리 부족

#### 최대 길이

이상치 탐지  
잘릴 위험  
정보 손실

#### 분포 확인

히스토그램  
불균형?  
조정 필요

💡 적절한 max\_seq\_length 설정의 기준



# Context Length 제한

`max_seq_length`

## 잘림 (Truncation)

너무 긴 텍스트  
뒷부분 잘림  
정보 손실 위험

## 패딩 (Padding)

너무 짧은 텍스트  
빈 공간 채우기  
메모리 낭비

💡 실습에서는 2048 사용 (EXAONE 기준)



# Train/Validation Split

## 데이터 분리 전략

### 분리 비율

Train 80-90%

Validation 10-20%

적절한 균형

### 데이터 누수 방지

Test 데이터 절대 학습X

### 랜덤 vs 계층화

랜덤: 무작위

계층화: 분포 유지

일반적으로 랜덤

### 재현성

Random seed 고정

## 다음: 학습 이론

💡 Validation으로 과적합 모니터링



## Oracle 없는 경우

모른다고 말하는 법 배우기

### 일부 샘플

Oracle 제거  
Distractor만 제공  
정답 없는 상황

### 학습 목표

"정보 부족" 판단  
"모르겠습니다" 답변  
거짓말보다 낫다

💡 잘못된 답변보다 "모름"을 인정하는 게 더 정직하다



# Label 마스킹 원리

## 답변만 학습하기

### 질문 부분

학습하지 않음 (mask)

-100으로 표시

Loss 계산 제외

### 답변 부분

학습 대상

실제 토큰 ID

Loss 계산 포함

💡 질문 형식을 외우는 게 아니라, 답변 생성에만 집중



# 토큰화 (Tokenization)

텍스트를 숫자로 변환

## 왜 필요한가?

모델은 숫자만 이해할 수 있다

텍스트 → 토큰 → ID (숫자)

예: "안녕" → [12345, 67890]

💡 Tokenizer가 이 변환을 자동으로 처리



# Context 구성

질문 + 문서들

Question + Document 1 + Document 2 + ... + Document N

순서는 랜덤  
(위치 편향 방지)

모델이 내용으로  
판단하도록

💡 Oracle이 항상 첫 번째면 모델이 위치를 외워버린다



## Section 3 정리

### 핵심 개념

- ✓ RAFT: 검색 시뮬레이션을 학습 데이터로
- ✓ Oracle (정답) + Distractors (노이즈) 섞어서 제공
- ✓ 모델이 유용한 문서 선별하는 법 학습
- ✓ Label 마스킹으로 답변 부분만 효율적 학습

다음: 학습 과정은 어떻게 진행되는가?

💡 RAFT = 현실적인 검색 상황을 학습 데이터로



# 학습이란?

틀린 만큼 수정하기

## 1. 예측

모델이  
답을 생성

## 2. 비교

정답과  
비교

## 3. 조정

차이만큼  
파라미터  
수정

## 반복

점점  
정확  
해짐

💡 비유: 과녁 맞추기. 빗나간 만큼 다음에 조정



# Loss Function

얼마나 틀렸나?

## Cross-Entropy Loss

확률 분포의 차이를 측정

정답에 가까울수록 Loss ↓

0에 가까워지도록 학습

💡 Loss = 틀린 정도. 낮을수록 좋다



# Optimizer

어떻게 수정할까?



## Gradient

틀린 방향 계산  
어느 방향으로 수정할지



## Learning Rate

수정 폭 결정  
한 번에 얼마나 바꿀지

💡 Adam: 효율적인 수정 방법 (자동으로 학습률 조절)



# Learning Rate

한 번에 얼마나?

## 크면

- ✓ 빠르게 이동
- ✗ 불안정
- ✗ 최적점 건너뛰

## 작으면

- ✓ 안정적
- ✗ 너무 느림
- ✗ 수렴 안 함

💡 비유: 걸음 크기. 너무 크면 넘어지고, 너무 작으면 못 간다



# Batch

몇 개씩 묶어서?

## Batch 개념

한 번에 여러 샘플 보고 평균내기

안정적인 학습

메모리와 속도의 균형

💡 비유: 시험 문제를 한 문제씩 vs 여러 문제 풀고 평균 점수



# Gradient Accumulation

## 메모리 절약 기법

### 일반적인 방법

큰 배치 한 번에 처리  
→ 메모리 많이 필요

### Accumulation

작은 배치 여러 번  
Gradient 누적 후 업데이트  
→ 효과는 같고 메모리 절약

💡 비유: 큰 집을 한 번에 vs 나눠서 여러 번



# Epoch

전체 데이터 몇 바퀴?

**1 Epoch = 모든 데이터 1회**

여러 epoch 반복

너무 많으면 과적합

너무 적으면 학습 부족

💡 비유: 문제집을 몇 번 반복할까?



# Warmup Steps

서서히 올리기

## 문제: 초기 불안정

처음부터 큰 Learning Rate

→ Gradient 폭발

→ Loss 발산

## 해결: Warmup

0에서 서서히 증가

안정적인 시작

전체 10% 정도

💡 비유: 자동차 워밍업처럼 천천히 시작



# Learning Rate Scheduler

동적으로 조절

## Cosine Annealing

코사인 곡선  
부드럽게 감소  
많이 사용

## Linear Decay

선형 감소  
단순하고 안정  
예측 가능

## Constant

고정  
간단  
덜 효율적

### 왜 필요한가?

후반: 세밀한 조정  
수렴 가속화

### 실습

Cosine 사용

💡 학습 진행에 따라 Learning Rate 줄이기



# Gradient Clipping

폭발 방지

## Gradient Explosion

Gradient가 너무 큼  
파라미터 급변  
NaN Loss 발생

## Clipping

Max Norm 설정 (1.0)  
초과 시 잘라냄  
안정적 학습

💡 비유: 속도 제한기처럼 최대값 제한



# Mixed Precision Training

FP16 / BF16

## FP32 (원래)

32-bit 정밀도  
정확하지만 느림  
메모리 많이 사용

## BF16 (추천)

16-bit  
넓은 범위  
Overflow 적음

## FP16

16-bit 정밀도  
빠르고 가벼움

범위 제한

## 효과

메모리 50% 절감  
속도 2배 향상

다음: 평가 방법

💡 실습에서는 BF16 사용 (Modern GPU)



# 과적합 (Overfitting)

문제집만 외우기


## 증상

Training Loss ↓ (좋음)  
Validation Loss ↑ (나쁨)  
새 데이터에 약함

## 해결

Dropout (일부 끄기)  
Early Stopping  
더 많은 데이터

💡 비유: 문제집 답을 외웠지만, 새 문제는 못 푸는 상황



# 학습 곡선 읽기

## 진행 상황 체크

### 정상 학습

Train Loss ↓  
Val Loss ↓  
계속 진행

### 과적합

Train Loss ↓  
Val Loss ↑  
주의 필요

### 실패

Loss 발산  
NaN 발생  
재시작

💡 두 Loss를 함께 모니터링해야 한다



## Section 4 정리

### 핵심 개념

- ✓ 학습: 예측 → 비교 → 조정 반복
- ✓ Loss: 틀린 정도 (0에 가깝게)
- ✓ Learning Rate: 수정 폭 (균형 중요)
- ✓ 과적합: Training vs Validation 곡선으로 감지

### 다음: 고급 학습 기법

💡 Loss를 줄이는 반복 과정이 학습



## 평가의 필요성

### Before vs After

#### 주관적 느낌

"더 좋아진 것 같다"


✗ 불명확

#### 정량적 지표

숫자로 측정

✓ 명확한 비교

💡 학습 전/후를 객관적으로 비교해야 한다



# ROUGE 원리

## 단어 겹침 비율

### 개념

정답과 예측의 공통 단어

Recall 중심 (정답 단어를 얼마나 포함?)

요약/생성 태스크 표준

💡 ROUGE = Recall-Oriented Understudy for Gisting Evaluation



## ROUGE의 한계

형태만 본다

### 문제점

❌ "절감" ≠ "줄임" (다른 단어로 인식)

❌ 동의어 무시

❌ 의미는 고려하지 않음

💡 단어 매칭만으로는 의미적 유사성을 놓친다



# Embedding Similarity

의미 거리 측정

## 1. 임베딩

문장 →  
벡터

## 2. 거리 계산

Cosine  
Similarity

## 3. 장점

동의어  
인식  
"절감" ≈ "줄임"

💡 의미가 비슷하면 벡터도 가까워진다



## Section 5 정리

### 핵심 개념

- ✓ 객관적 평가 지표 필요
- ✓ ROUGE: 단어 겹침 (빠르고 간단)
- ✓ Embedding Similarity: 의미 유사도 (정확)
- ✓ 두 지표를 함께 사용하여 다각도 평가

### 다음: 실전 & 배포

💡 ROUGE + Embedding Similarity 조합이 최선



## OOM & Checkpoint 전략

### OOM 대처법

- ✓ Batch size 줄이기
- ✓ Gradient Accumulation
- ✓ Sequence 길이 줄이기
- ✓ Mixed Precision

### Checkpoint 저장

- ✓ 주기적 저장
- ✓ Best model 보관
- ✓ 학습 재개 가능
- ✓ 실험 재현성

💡 중단되어도 다시 시작할 수 있게 준비



## 전체 여정 정리

### 오늘 배운 이론

- 1 문제: Fine-tuning은 메모리가 많이 필요
- 2 LoRA: 변화량만 기록 ( $B \times A$  분해)
- 3 QLoRA: 모델 압축 (4-bit 양자화)
- 4 RAFT: 검색 시뮬레이션 학습 (Oracle + Distractors)
- 5 데이터: 품질 검증, 토큰 분석, Train/Val Split
- 6 학습: Loss, Optimizer, Warmup, Scheduler
- 7 평가: ROUGE + Embedding Similarity
- 8 실전: OOM 대처, Checkpoint, Mixed Precision

**이제 실습 노트북으로 직접 해봅시다!**

 이론 학습 완료! 8개 노트북으로 직접 실행해보세요

