LoRA & Tuning with LoRA modules

장윤나

2023.09.07

Contents

1. LORA: Low-Rank Adaptation of Large Language Models

Edward Hu* **Yelong Shen*** **Phillip Wallis** **Zeyuan Allen-Zhu**
Yuanzhi Li **Shean Wang** **Lu Wang** **Weizhu Chen**
Microsoft Corporation
edward.hu@mila.quebec
{yeshe, phwallis, zeyuana, swang, luw, wzchen}@microsoft.com
yuanzhil@andrew.cmu.edu

1. Introduction

- 기존 LM 활용 방법: one large model → multiple downstream models (task adaptation)
  - 이 때 모든 weight parameters를 재학습하며 많은 시간과 비용이 소요
- Low-Rank Adaptation (LoRA) 은 low-rank를 이용하여 시간, 자원 비용을 줄이고자 한 방법
  - Measuring the Intrinsic Dimension of Objective Landscapes
  - Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-tuning
  - 위 두 개의 논문 등에서 over-parameterized model은 low intrinsic dimension으로 존재하고 있다는 사실에 기반
- 이에 model adaptation 동안의 weight change에도 low intrinsic rank를 가질 것이라는 가정에 기반하여 low-rank방법론을 적용

1. Introduction

- 모델의 pretrained weights는 freeze시키고 몇 개의 dense layer만 학습
- Dense layer의 weight를 low rank로 decompose하는 matrices만을 학습시키는 방법
- W는 freeze, A, B만 학습하고 W에 더해줌
- Pretrained model을 가지고 있는 상태에서 특정 task에 adaptation 학습 된 A, B만 저장하면 되고, 또 다른 태스크에 adaptation하기 위해서는 A', B'만 갈아 끼우면 되기에 storage, task switching면에서 매우 효율적
- Inference latency를 야기시키지 않고, efficiency와 model quality간의 trade-off가 발생할 경향이 낮아짐

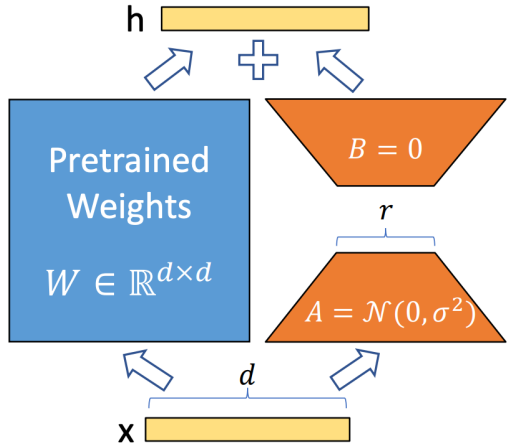


Figure 1: Our reparametrization. We only train A and B.

1. LORA: Low-Rank Adaptation of Large Language Models

2. Problem Statement

- LoRA는 training objective에 agnostic하지만 본 논문에서는 large language model에 맞추어 설명
- Φ의 parameter를 가지는 pretrained language model $P_\Phi(y|x)$은 GPT와 같은 multi-task learner
- 이러한 PLM이 adaptation 될 downstream task는 context-target pair의 데이터 $Z = \{(x_i, y_i)\}_{i=1,\ldots,N}$를 가짐
- 기존 full fine-tuning은 모델의 pre-trained weights Φ_0로 initialized, 아래 objective를 최적화하며 $\Phi_0 + \Delta\Phi$를 업데이트

$$\max_{\Phi} \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{|y|} \log\left(P_\Phi(y_t|x, y_{<t})\right)$$

1. LORA: Low-Rank Adaptation of Large Language Models

2. Problem Statement

- Full fine-tuning은 각 태스크를 위해 $|\Phi_0|$ 와 같은 크기의 $|\Delta\Phi|$ 를 매번 재학습 해야 함
- LoRA는 업데이트해야 하는 파라미터 $\Delta\Phi$를 $\Delta\Phi = \Delta\Phi(\Theta)$와 같이 encode하여 훨씬 작은 사이즈의 파라미터 Θ로 대체해서 학습
- 최적의 $\Delta\Phi$를 찾는 태스크는 Θ를 최적화하는 것으로 대체됨

$$\max_{\Phi} \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{|y|} \log\left(P_{\Phi}(y_t|x, y_{<t})\right)$$

$$\max_{\Theta} \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{|y|} \log\left(p_{\Phi_0+\Delta\Phi(\Theta)}(y_t|x, y_{<t})\right)$$

1. LORA: Low-Rank Adaptation of Large Language Models

3. Our Method

- LoRA는 adaptation동안 low intrinsic rank를 가진 weight로 update하는 방법
- $W_0 + \Delta W$를 W_0는 frozen, low rank로 decomposition된 $A \in \mathbb{R}^{d \times r}$, $B \in \mathbb{R}^{r \times k}$ 만 학습: $W_0 + AB$로 업데이트
- W_0 와 $\Delta W = AB$ 는 같은 입력 x 에 곱해지며 출력은 coordinate-wise하게 sum

$$h = W_0 x + \Delta W x = W_0 x + BAx$$

- A는 random Gaussian, B는 0 initialized (학습 시작 시 $\Delta W = AB$ 값은 0)
- 이후 $\Delta W x$를 $\frac{\alpha}{r}$로 scale 시킴 ($\alpha = r$, r값을 변화시킬 때 hyper-parameter 재조정 필요성을 줄여 줌)
- Trainable weights를 최소화하기 위하여 Transformer의 attention weights (query, value) 에만 적용

1. LORA: Low-Rank Adaptation of Large Language Models

3. Our Method

```
        # Actual trainable parameters
        if r > 0:
            self.lora_A = nn.Parameter(self.weight.new_zeros((r, num_embeddings)))
            self.lora_B = nn.Parameter(self.weight.new_zeros((embedding_dim, r)))
            self.scaling = self.lora_alpha / self.r
            # Freezing the pre-trained weight matrix
            self.weight.requires_grad = False
        self.reset_parameters()

    def reset_parameters(self):
        nn.Embedding.reset_parameters(self)
        if hasattr(self, 'lora_A'):
            # initialize A the same way as the default for nn.Linear and B to zero
            nn.init.zeros_(self.lora_A)
            nn.init.normal_(self.lora_B)

    def forward(self, x: torch.Tensor):
        if self.r > 0 and not self.merged:
            result = nn.Embedding.forward(self, x)
            after_A = F.embedding(
                x, self.lora_A.transpose(0, 1), self.padding_idx, self.max_norm,
                self.norm_type, self.scale_grad_by_freq, self.sparse
            )
            result += (after_A @ self.lora_B.transpose(0, 1)) * self.scaling
            return result
        else:
            return nn.Embedding.forward(self, x)
```

Decomposition matrix A, B

Set scaling

Freeze pre-trained weights W_0

Initialize A, B

W_0

$W_0 + AB(\frac{\alpha}{r})$

1. LORA: Low-Rank Adaptation of Large Language Models

3. Experiments

- GPT-2 Medium, Large에 대해 실험
  - $\text{Adapter}^{\text{H}}$ adapter를 self-attention과 residual connection 사이에 (original)
  - $\text{Adapter}^{\text{L}}$ MLP와 LayerNorm 다음에 있는 efficient design (Lin et al., (2020))
  - FT^{Top2} 마지막 두 개 레이어만 학습 (Li&Liang, (2021))
  - PreLayer 이전 레이어에서 계산된 prefix embedding 위치의 activation이 trainable parameter로 대체
- 베이스라인의 성능을 비슷하거나 더 적은 수의 파라미터로 넘는 경우를 많이 보임

Model & Method	# Trainable Parameters	E2E NLG Challenge BLEU	NIST	MET	ROUGE-L	CIDEr
GPT-2 M (FT)*	354.92M	68.2	8.62	46.2	71.0	2.47
GPT-2 M ($\text{Adapter}^{\text{L}}$)*	0.37M	66.3	8.41	45.0	69.8	2.40
GPT-2 M ($\text{Adapter}^{\text{L}}$)*	11.09M	68.9	8.71	46.1	71.3	2.47
GPT-2 M ($\text{Adapter}^{\text{H}}$)	11.09M	$67.3_{\pm.6}$	$8.50_{\pm.07}$	$46.0_{\pm.2}$	$70.7_{\pm.2}$	$2.44_{\pm.01}$
GPT-2 M (FT^{Top2})*	25.19M	68.1	8.59	46.0	70.8	2.41
GPT-2 M (PreLayer)*	0.35M	69.7	8.81	46.1	71.4	2.49
GPT-2 M (LoRA)	0.35M	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.85}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{71.8}_{\pm.1}$	$\mathbf{2.53}_{\pm.02}$
GPT-2 L (FT)*	774.03M	68.5	8.78	46.0	69.9	2.45
GPT-2 L ($\text{Adapter}^{\text{L}}$)	0.88M	$69.1_{\pm.1}$	$8.68_{\pm.03}$	$46.3_{\pm.0}$	$71.4_{\pm.2}$	$\mathbf{2.49}_{\pm.0}$
GPT-2 L ($\text{Adapter}^{\text{L}}$)	23.00M	$68.9_{\pm.3}$	$8.70_{\pm.04}$	$46.1_{\pm.1}$	$71.3_{\pm.2}$	$2.45_{\pm.02}$
GPT-2 L (PreLayer)*	0.77M	70.3	8.85	46.2	71.7	2.47
GPT-2 L (LoRA)	0.77M	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.89}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{72.0}_{\pm.2}$	$2.47_{\pm.02}$

3. Experiments

- GPT-3 모델에서도 다른 방법론에 비해 더 적은 파라미터로 더 좋은 성능을 보임

Model&Method	# Trainable Parameters	WikiSQL Acc. (%)	MNLI-m Acc. (%)	SAMSum R1/R2/RL
GPT-3 (FT)	175,255.8M	**73.8**	89.5	52.0/28.0/44.5
GPT-3 (BitFit)	14.2M	71.3	91.0	51.3/27.4/43.5
GPT-3 (PreEmbed)	3.2M	63.1	88.6	48.3/24.2/40.5
GPT-3 (PreLayer)	20.2M	70.1	89.5	50.8/27.3/43.5
GPT-3 ($Adapter^H$)	7.1M	71.9	89.8	53.0/28.9/44.8
GPT-3 ($Adapter^H$)	40.1M	73.2	**91.5**	53.2/29.0/45.1
GPT-3 (LoRA)	4.7M	73.4	**91.7**	**53.8/29.8/45.9**
GPT-3 (LoRA)	37.7M	**74.0**	**91.6**	53.4/29.2/45.1

1. LORA: Low-Rank Adaptation of Large Language Models

3. Experiments

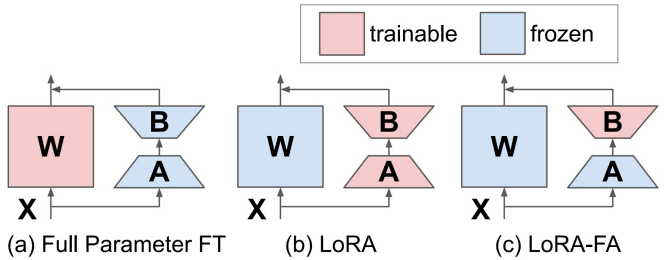
- Optimal rank?
- 매우 작은 r에도 학습을 잘 함, 태스크마다 분석이 필요할 듯

	Weight Type	$r = 1$	$r = 2$	$r = 4$	$r = 8$	$r = 64$
WikiSQL(±0.5%)	W_q	68.8	69.6	70.5	70.4	70.0
	W_q, W_v	73.4	73.3	73.7	73.8	73.5
	W_q, W_k, W_v, W_o	74.1	73.7	74.0	74.0	73.9
MultiNLI (±0.1%)	W_q	90.7	90.9	91.1	90.7	90.7
	W_q, W_v	91.3	91.4	91.3	91.6	91.4
	W_q, W_k, W_v, W_o	91.2	91.7	91.7	91.5	91.4

1. LORA: Low-Rank Adaptation of Large Language Models

Quantization & Memory Efficiency

- LORA-FA: Memory-efficient Low-rank Adaptation for Large Language Models Fine-tuning



- LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale
  - Inference 속도 향상을 위해 Transformer의 feed-forward, attention projection layer의 행렬연산에 Int8 사용
- QLORA: Efficient Finetuning of Quantized LLMs (Guanaco)
  - Pretrained weights을 4bit quantization, 그 후 LoRA weights를 더하는 방식으로 학습
- QuIP: 2-Bit Quantization of Large Language Models With Guarantees
  - 2-bit LLM compression 가능성 보임

2. Stack More Layers Differently: High-Rank Training Through Low-Rank Updates

Vladislav Lialin,* Namrata Shivagunde, Sherin Muckatira, and Anna Rumshisky
University of Massachusetts Lowell

2. Stack More Layers Differently: High-Rank Training Through Low-Rank Updates

1. Introduction

- Machine learning 분야 연구 추세 - “stack more layers” 많은 파라미터를 가지는 네트워크를 학습
- Large models의 접근성을 좋게 하는 parameter-efficient fine-tuning (PEFT) 방법론의 중요성 증가
- ReLoRA는 high-rank network를 학습시키기 위하여 low-rank 업데이트를 활용하는 방법
- Regular network training과 비슷한 성능 달성
- Multi-billion-parameter network의 efficient training을 가능하도록 함

2. Method

- LoRA는 A, B의 matrices를 학습시켜 학습이 끝난 후 original parameter에 더함

$$\delta W = sW_A W_B$$
$$W_A \in \mathbb{R}^{\text{in} \times r}, W_B \in \mathbb{R}^{r \times \text{out}}$$

- 학습시간동안 전체 업데이트는 각각의 individual matrices보다는 높은 rank를 가질 수 있지만, LoRA는 rank $r = \max_{W_A, W_B} \text{rank}(W_A W_B)$ 로 제한됨
- 만약 LoRA를 restart할 수 있다면, W_A와 W_B를 학습동안 merge하고 matrix value를 reset할 수 있다면 update의 total rank를 늘릴 수 있음

$$\text{rank}(A + B) \leq \text{rank}(A) + \text{rank}(B).$$

2. Stack More Layers Differently: High-Rank Training Through Low-Rank Updates

2. Method

- 이를 반복한다면 전체 네트워크 업데이트는 다음과 같이 됨:

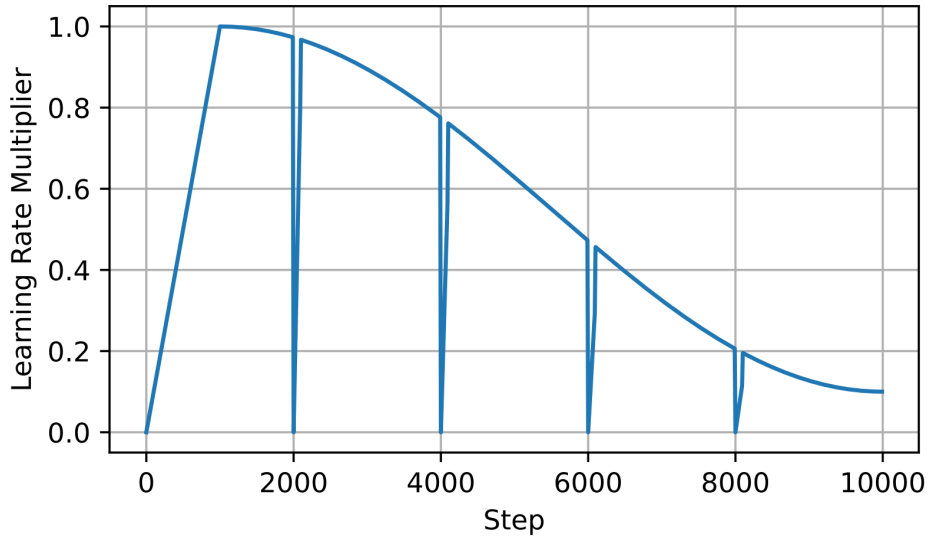
$$\Delta W = \sum_{t=0}^{T_1} \delta W_t + \sum_{t=T_1}^{T_2} \delta W_t + \cdots + \sum_{t=T_{N-1}}^{T_N} \delta W_t = sW_A^1 W_B^1 + sW_A^2 W_B^2 + \cdots + sW_A^N W_B^N \quad (3)$$

where the sums are independent enough, meaning that $\text{rank}(W_A^i W_B^i) + \text{rank}(W_A^j W_B^j) \geq r$.

- 전체 업데이트는 N번의 작은 업데이트의 합, 기존 LoRA rank r보다 더 큰 rank를 가지게 됨
- 하지만 restart를 naïve하게 구현하는 것은 모델이 diverge하도록 만들 수 있기에 쉬운 문제가 아님
- Merge후 restart때 기존의 gradient moments를 사용하면 새로 학습이 아닌 기존과 같은 subspace를 최적화하도록 만들기에 optimizer도 reset시킴

2. Method

- ReLoRA에서 merge-and-reinit을 하는 동안 optimizer state역시 reset을 시키며 learning rate도 0으로 초기화 및 warmup 수행
- ReLoRA는 업데이트에 있어 효과적인 rank를 증가시키고자 하였으며, optimizer reset과 jagged scheduler를 통해 학습 안정화



2. Stack More Layers Differently: High-Rank Training Through Low-Rank Updates

3. Experiments

- LlaMA구조와 비슷한 Transformer 레이어 모델을 C4 데이터셋에 학습
- Rank는 128 사용, effective attention을 위해 bfloat16, Flash attention 사용
- Control은 low-rank 학습때와 같은 파라미터 수를 갖는 full-rank 모델

	60M	130M	250M	350M
Full training	33.81 (60M)	23.65 (130M)	22.39 (250M)	18.66 (350M)
Control	36.52 (43M)	27.30 (72M)	25.43 (99M)	23.65 (130M)
LoRA	47.44 (43M)	34.17 (72M)	36.60 (99M)	57.11 (125M)
LoRA + Warm Start	**34.73** (43M)	**25.46** (72M)	**22.86** (99M)	**19.73** (125M)
ReLoRA	**34.46** (43M)	**25.04** (72M)	**22.48** (99M)	**19.32** (125M)
Training tokens (billions)	1.2	2.6	6.8	6.8

Table 2: Comparing perplexities between baseline methods and ReLoRA (lower is better). Number of trainable parameters for each model in (brackets). Control baseline is full-rank training a model with the same total number of parameters as the number of trainable parameters in low-rank training. Low-rank training is **bold** if it outperforms the Control baseline.

2. Stack More Layers Differently: High-Rank Training Through Low-Rank Updates

3. Experiments

- W_Q, W_K, W_V, W_{down} matrices의 singular value 비교
- LoRA보다 ReLoRA가 full-rank training과 더 비슷함

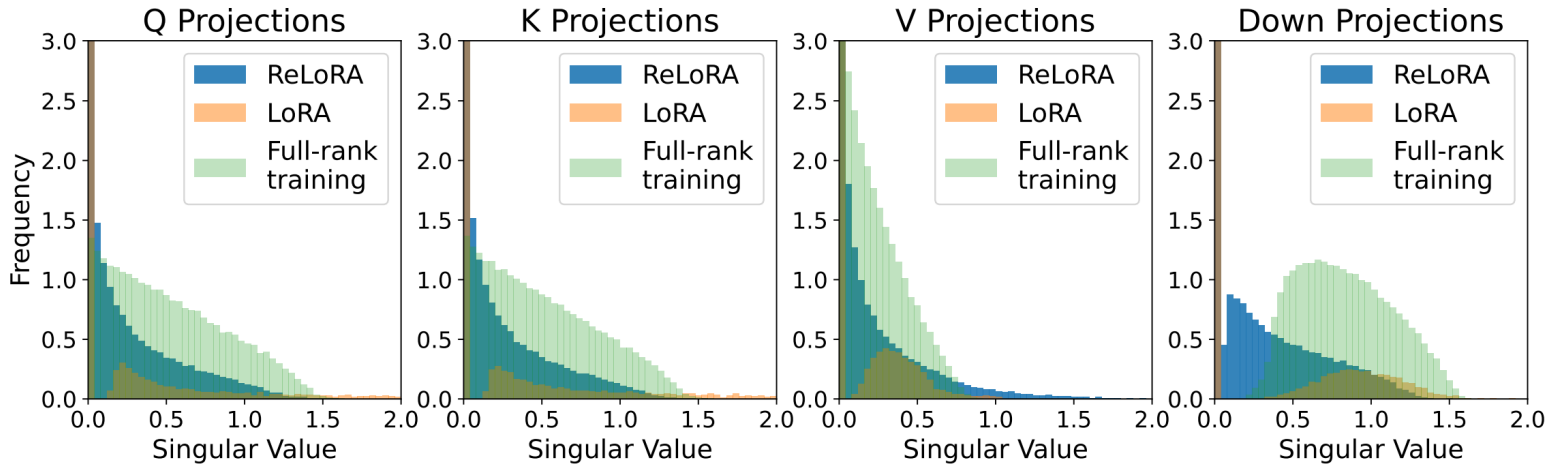


Figure 3: Singular values spectra of the weight difference between ReLoRA and LoRA at 5,000 iterations (warm start) and 20,000 iterations. ReLoRA exhibits a closer resemblance to full-rank training singular values than LoRA, indicating its effectiveness in approximating full-rank behavior.

2. Stack More Layers Differently: High-Rank Training Through Low-Rank Updates

3. Experiments

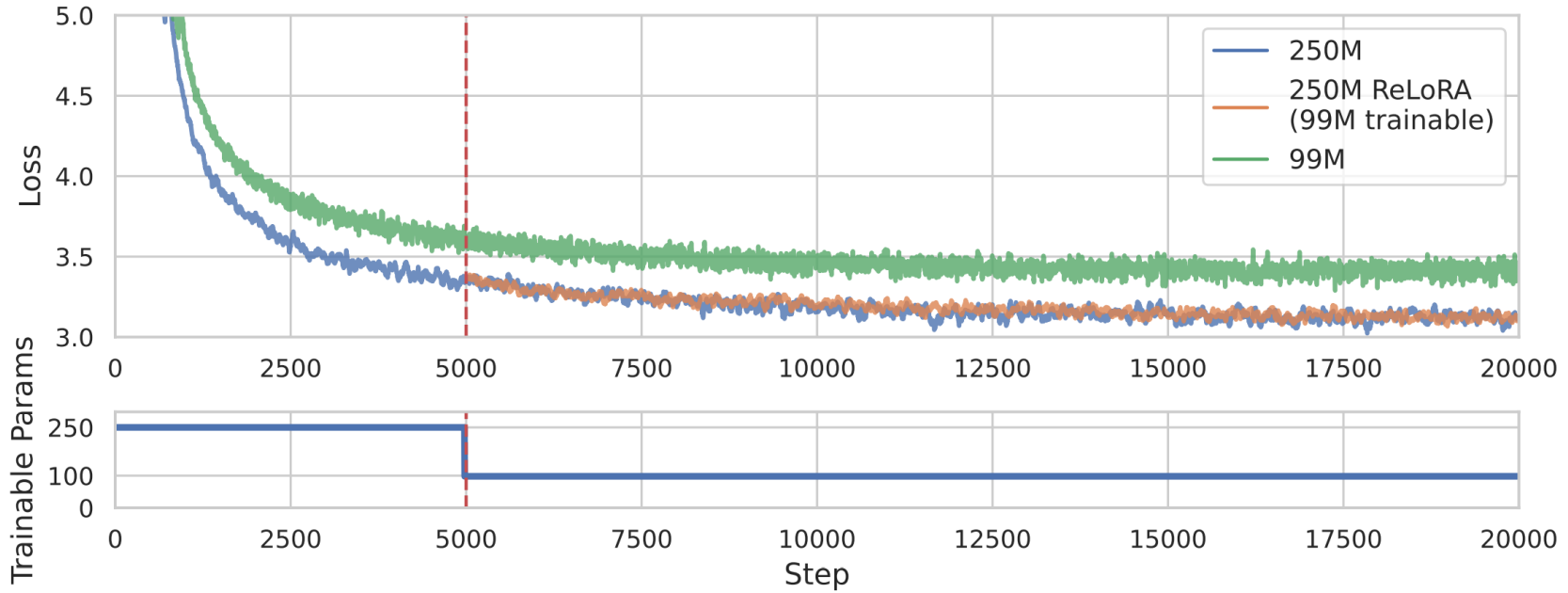


Figure 1: ReLoRA learns a high-rank network through a sequence of low-rank updates. It outperforms networks with the same trainable parameter count and achieves similar performance to training a full network at 100M+ scale. The efficiency of ReLoRA increases with the model size, making it a viable candidate for multi-billion-parameter training.

3. LoraHub: Efficient Cross-Task Generalization via Dynamic Lora Composition

Chengsong Huang[†§*], Qian Liu[†*], Bill Yuchen Lin[◇*], Tianyu Pang[†], Chao Du[†], Min Lin[†]
[†]Sea AI Lab, Singapore
[§]Washington University in St. Louis, MO, USA
[◇]Allen Institute for AI, Seattle, WA, USA

3. LoraHub: Efficient Cross-Task Generalization via Dynamic Lora Composition

1. Introduction

- Large language models의 큰 파라미터 때문에 fine-tuning에서의 computational efficiency와 memory usage를 줄이기 위해 LoRA 계열 연구가 많이 진행됨
- 기존 LoRA 연구에서는 각 태스크 혹은 도메인에 specialize되도록 학습
- 본 논문에서는 LoRA 모듈이 unseen task에 대해 효율적으로 일반화하도록 구성할 수 있을지에 대해 연구
- LoRA 모듈이 자동으로 assemble 되도록 하여 human expertise에 대한 의존을 낮춤
- 다양한 LoRA 모듈이 특정 학습을 위해 사용될 수 있기에 LoraHub learning 방법론이라고 칭함

3. LoraHub: Efficient Cross-Task Generalization via Dynamic Lora Composition

1. Introduction

- Large language models의 큰 파라미터 때문에 fine-tuning에서의 computational efficiency와 memory usage를 줄이기 위해 LoRA 계열 연구가 많이 진행됨
- 기존 LoRA 연구에서는 각 태스크 혹은 도메인에 specialize되도록 학습
- 본 논문에서는 LoRA 모듈이 unseen task에 대해 효율적으로 일반화하도록 구성할 수 있을지에 대해 연구
- LoRA 모듈이 자동으로 assemble 되도록 하여 human expertise에 대한 의존을 낮춤
- 다양한 LoRA 모듈이 특정 학습을 위해 사용될 수 있기에 LoraHub learning 방법론이라고 칭함

3. LoraHub: Efficient Cross-Task Generalization via Dynamic Lora Composition

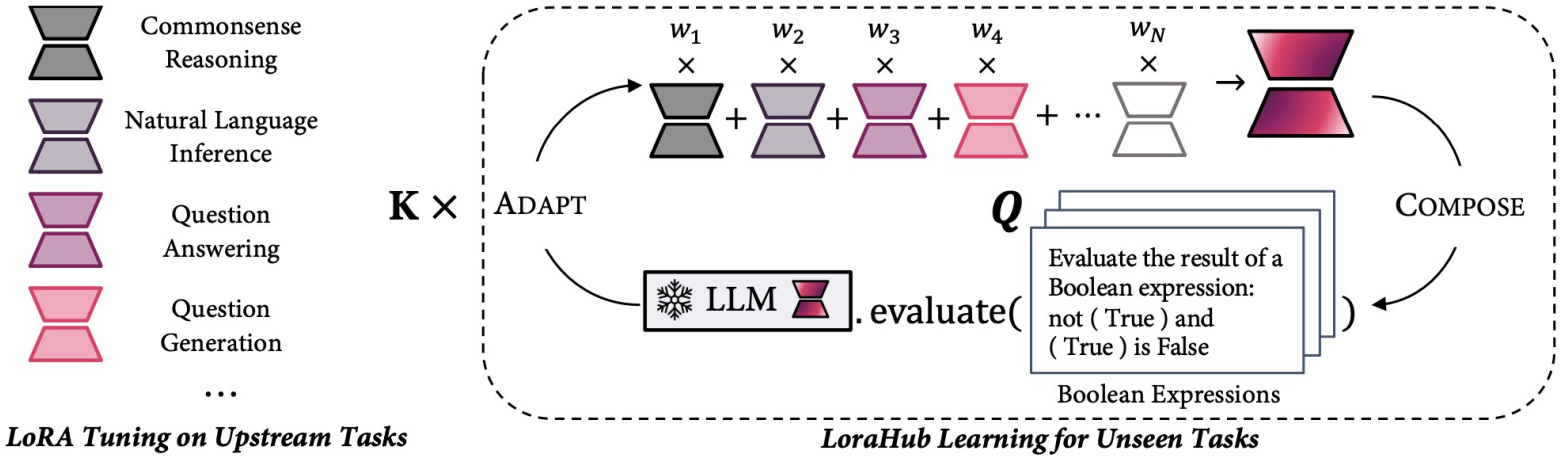
2. Methodology

- 이전에 학습하지 않았던 새로운 태스크 $\mathcal{T}'$를 잘 수행하는 cross-task generalization 능력
- 그 중 제한된 몇 개의 labeled examples Q만이 주어졌을 때 pre-trained large language model M_θ 해당 태스크를 수행해야하는 few-shot learning
- M_θ을 직접 fine-tuning하여 M_ϕ로 업데이트 하는 방식은 좋은 성능을 낼 수 있지만, inefficient, time-consuming하며 Q가 적을 때 unstable함
- LoRA는 전체 모델이 아닌 외부의 작은 모듈에 대한 튜닝만을 필요로 하며 적은 수의 examples이 주어졌을 때 빠르고 강건한 adaptation이 가능하지만,너무 적은 수의 샘플 (e.g., 5 pairs)로는 낮은 성능을 피할 수 없음
- 따라서 LoRA 연구는 같은 태스크내에서 학습 및 평가에 사용되었음

3. LoraHub: Efficient Cross-Task Generalization via Dynamic Lora Composition

2. Methodology

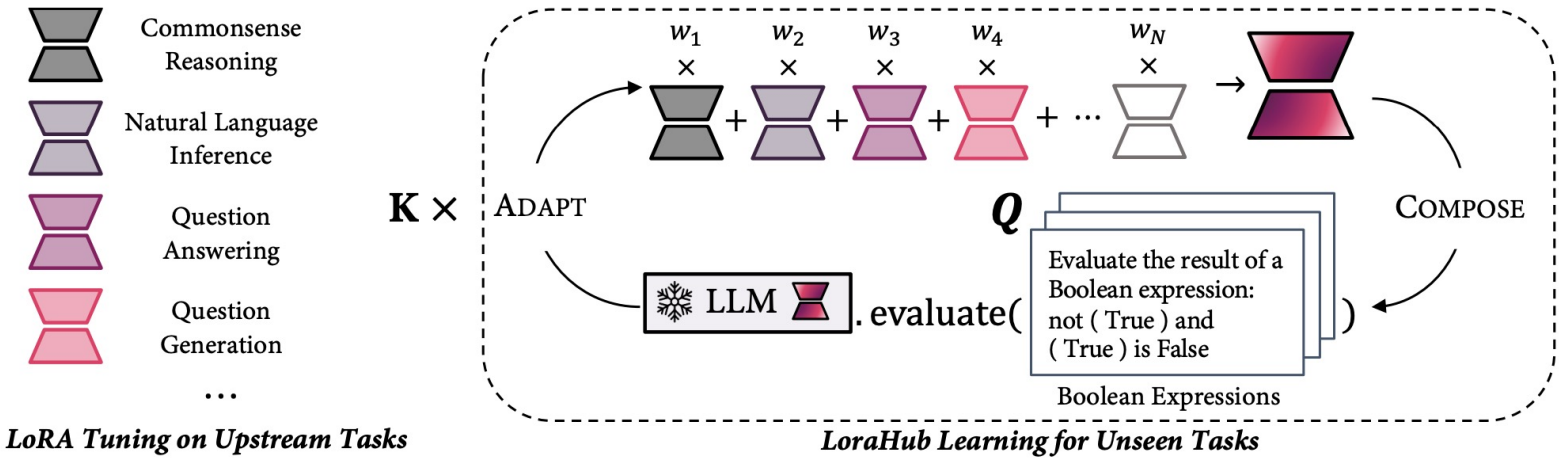
- N개의 태스크에 각각 학습시킨 N개 LoRA 모듈 $\{m_1, \ldots, m_N\}$
- LoraHub learning: (1) Compose, (2) Adapt phase
- Compose: 각 모듈에 w_i coefficients를 곱해 하나의 모듈로 더해 만든 $\hat{m}$ (w_i는 positive or negative)
- Adapt: M_θ에 $\hat{m}$이 더해지고 새로운 태스크 $\mathcal{T}'$에 대한 평가에 사용



3. LoraHub: Efficient Cross-Task Generalization via Dynamic Lora Composition

2. Methodology

- 이후 gradient-free algorithm을 이용하여 주어진 Q 샘플로 w를 업데이트하는 데 사용
- K(=40)번의 스텝을 iterate하고 optimum performing LoRA 모듈을 M_θ 에 적용
- 최종 LLM은 $M_\phi = \mathrm{LoRA}(M_\theta, \hat{m})$



3. Experiments

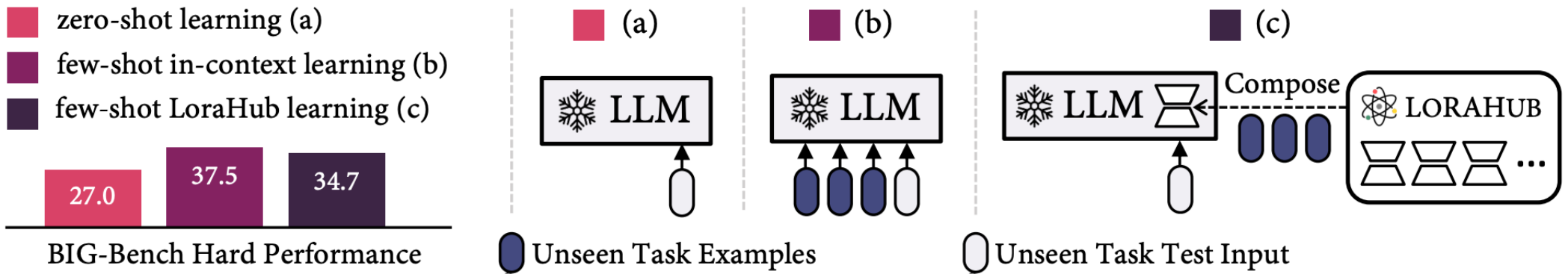
- LLM: Flan-T5 Large
- LoRA modules: Flan-T5 학습 시 사용한 200여개의 태스크에 각각 학습
- Pre-filtering: Compose 전 20개의 LoRA 모듈을 랜덤으로 선택
- Dataset: Big Bench Hard (BBH) – 다양한 도메인의 multiple-choice questions 중 27개 태스크 선택
- LoRA rank: 16, Batch: 64, LR: 1e-4, Epochs: 10
- Gradient-free method: Nevergrad optimization library

3. Experiments

- 5 samples, 5 runs
- (1) Zero-shot learning (**Zero**), (2) Few-shot in-context learning (**ICL**) , (3) Few-shot LoraHub learning (**LoraHub**)
- LoraHub에서 사용하는 토큰 수는 Zero에서와 동일하고 ICL 보다 작음
- Inference cost와 직결되는 입력 길이를 줄이며 ICL보다 적은 수의 토큰으로 비슷한 성능을 보임

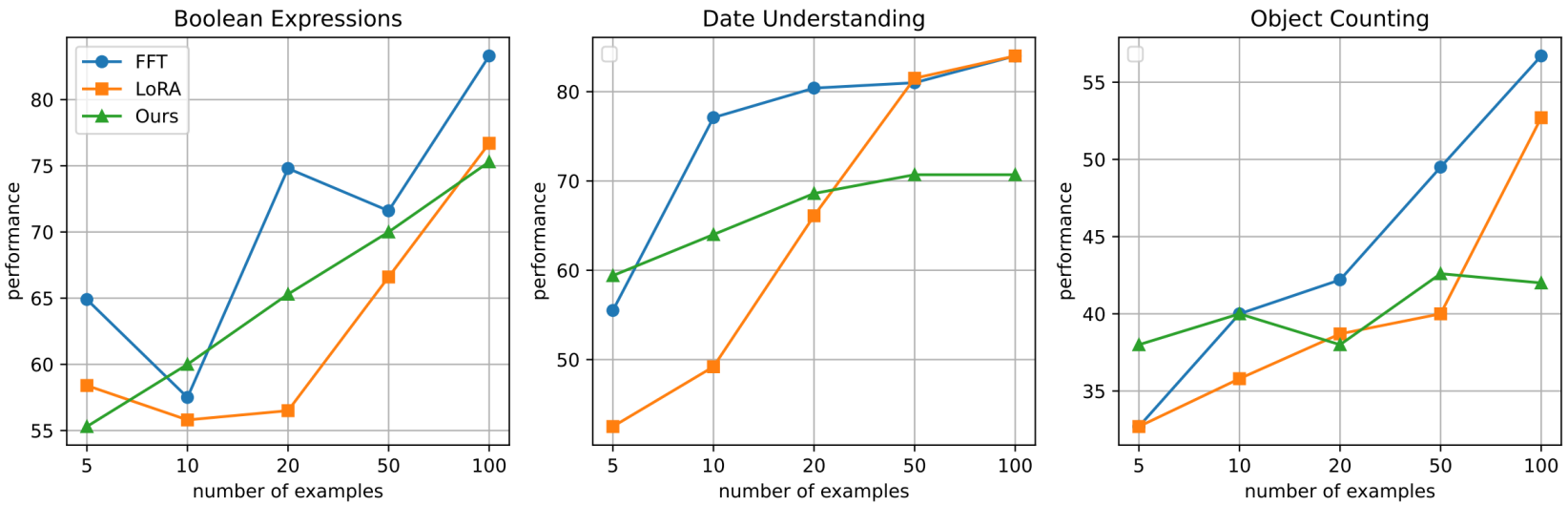
Task	Zero	ICL	LoraHub $_{avg}$	LoraHub $_{best}$
Boolean Expressions	54.0	**58.7**	56.0	59.3
Causal Judgement	57.5	56.3	**58.9**	63.2
Date Understanding	15.3	22.7	**29.6**	38.0
Disambiguation	0.0	**69.3**	46.0	69.3
Dyck Languages	1.3	**7.3**	0.3	1.3
Formal Fallacies	51.3	**58.0**	52.1	55.3
Geometric Shapes	6.7	**18.7**	7.5	9.3
Hyperbaton	6.7	**74.0**	57.5	61.3
Logical Deduction§ (five objects)	21.3	**40.0**	38.1	40.7
Logical Deduction§ (seven objects)	12.7	**42.0**	40.3	44.7
Logical Deduction§ (three objects)	0.0	**51.3**	49.7	51.3
Movie Recommendation	**62.7**	52.7	61.1	64.0
Multistep Arithmetic	**0.7**	**0.7**	**0.7**	0.7
Navigate	**47.3**	44.0	46.1	49.3
Object Counting	34.7	32.0	**35.0**	38.0
Penguins in a Table	43.5	39.1	**43.9**	45.7
Reasoning about Colored Objects	32.0	**38.7**	36.5	40.7
Ruin Names	**23.3**	18.7	21.0	23.3
Salient Translation Error Detection	37.3	**46.0**	37.3	37.3
Snarks	50.0	**55.1**	51.8	59.0
Sports Understanding	**56.0**	**56.0**	48.3	53.3
Temporal Sequences	16.7	**26.7**	18.7	21.3
Tracking Shuffled Objects§ (five objects)	**12.0**	**12.0**	**12.0**	12.0
Tracking Shuffled Objects§ (seven objects)	6.7	6.7	**7.1**	8.7
Tracking Shuffled Objects§ (three objects)	24.7	**30.7**	29.0	30.7
Web of Lies	**54.0**	**54.0**	53.0	54.7
Word Sorting	**1.3**	0.7	1.1	1.3
Average Performance per Task	27.0	**37.5**	34.7	38.3
Average Consumed Tokens per Example	**111.6**	597.8	**111.6**	111.6

3. Experiments



3. Experiments

- Examples 수를 늘렸을 때 Full-finetuning (FFT), standard LoRA, LoraHub 방법의 비교
  - Binary classification, multi-class, text generation
- Examples 수가 20개 보다 더 적을 때 LoraHub이 LoRA보다 더 좋은 성능을 보임



3. Experiments

- 가장 유용한 Top 5 LoRA 모듈 (27개의 BBH 태스크 중)
- 독해와 추론 능력을 내재적으로 요구하는 복잡한 태스크가 cross-task transfer에 도움

Rank	Task Name	Usefulness	Task Description
1	wiqa_last_process	0.719	Identifying the last step of a given process.
2	race_middle_Is_this_the_right_answer	0.681	Determining if given answer is correct.
3	wiqa_final_process	0.626	Identifying the final step of a given process.
4	adversarial_qa_dbidaf_based_on	0.611	Answering question created by an adversarial model-in-the-loop.
5	web_questions_whats_the_answer	0.583	Answering question based on information extracted from the web.

Q&A

감사합니다.