[논문리뷰]

"LoRA: Low-Rank Adaptation of Large Language Models"

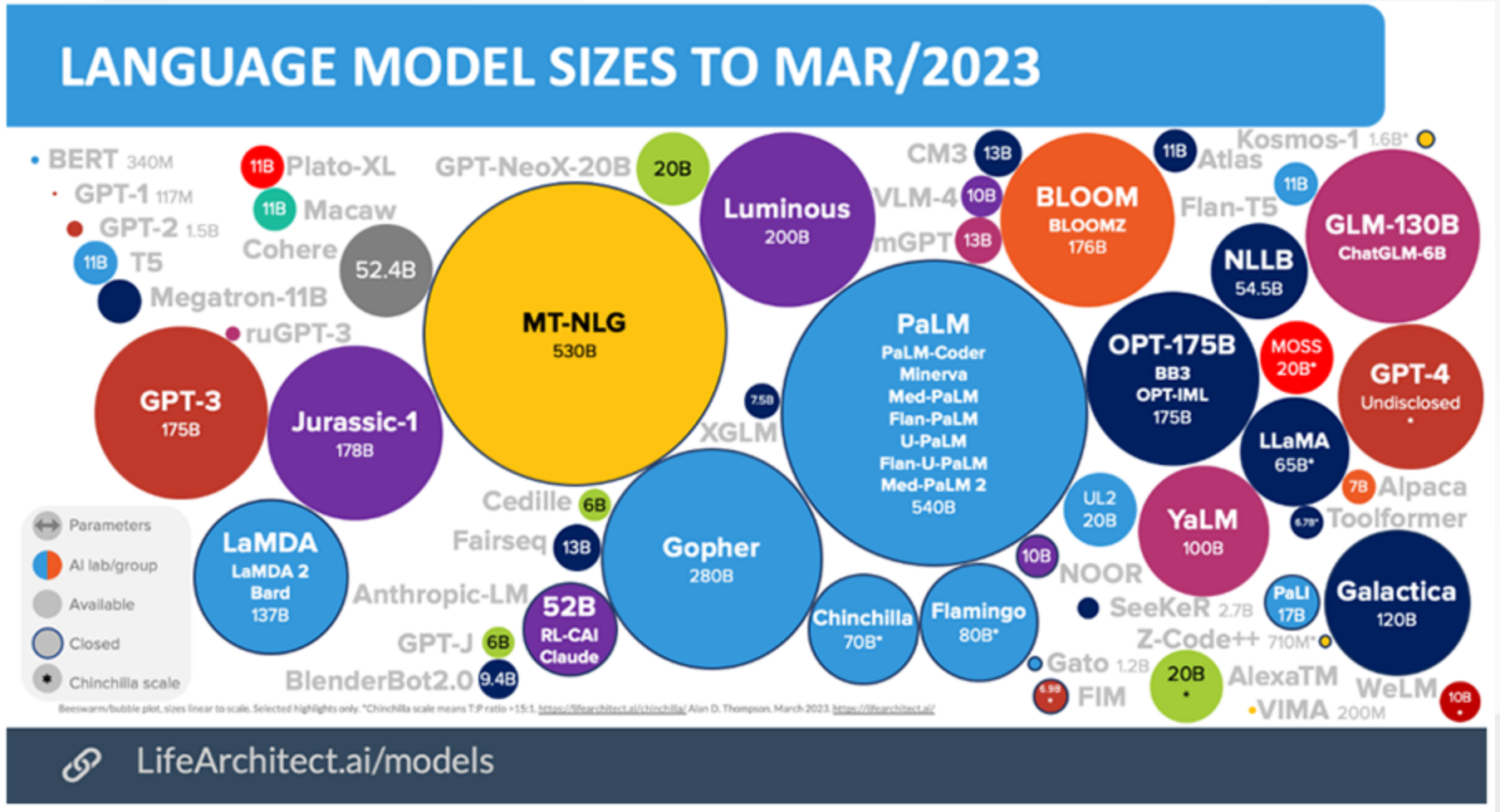
LoRA

-Microsoft Corporation-

2023.10.12

발표자 신중현

situation


LANGUAGE MODEL SIZES TO MAR/2023
BERT 340M
GPT-1 117M
GPT-2 1.5B
11B T5
Megatron-11B
11B Plato-XL
11B Macaw
Cohere
52.4B
ruGPT-3
GPT-NeoX-20B
20B
MT-NLG
530B
Luminous
200B
CM3 13B
VLM-4 10B
mGPT 13B
BLOOM
BLOOMZ
176B
11B Atlas
Kosmos-1 1.6B*
11B Flan-T5
GLM-130B
ChatGLM-6B
NLLB
54.5B
GPT-3
175B
Jurassic-1
178B
7.5B XGLM
PaLM
PaLM-Coder
Minerva
Med-PaLM
Flan-PaLM
U-PaLM
Flan-U-PaLM
Med-PaLM 2
540B
OPT-175B
BB3
OPT-IML
175B
MOSS
20B*
GPT-4
Undisclosed
LLaMA
65B*
7B Alpaca
6.7B* Toolformer
UL2
20B
YaLM
100B
Cedille 6B
Fairseq 13B
LaMDA
LaMDA 2
Bard
137B
Gopher
280B
10B NOOR
SeeKeR 2.7B
Z-Code++ 710M*
PaLI
17B
Galactica
120B
Anthropic-LM
52B
RL-CAI
Claude
GPT-J 6B
BlenderBot2.0 9.4B
Chinchilla
70B*
Flamingo
80B*
Gato 1.2B
4.5B FIM
20B
AlexaTM
VIMA 200M
WeLM
10B
Parameters
AI lab/group
Available
Closed
Chinchilla scale
LifeArchitect.ai/models

motivation

pretrained model - 다양한 downstream task adaptation

규모를 키운 pretrained 모델의 성능이 경험적으로, 성능적으로 확인된 가운데, 이 모델들을 활용해서 downstream task에 adaptation해서 여러문제를 풀겠다.

-ex) summarization, reading comprehension,

finetuning

이러한 down stream task들은 training data of context-target pairs 로 표현

$Z = (x_i, y_i)_{i=1,\ldots,N}$

fully finetuning

많은 language model의 objective function을 보면,

Maximum likelihood estimation MLE로 푼다.

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

이런 문제를 Backward propagation을 통해서 parameters를 업데이트를 하면서 풉니다.

근데 이럴때, optimizer가 사용되는데, 어떻게 gradient 값들이 지정될까요?

SGD

SGD는 각 배치(데이터 샘플의 작은 그룹)마다 모델 가중치를 업데이트합니다. 가중치(W)를 업데이트 하기 위한 수식은 다음과 같습니다:

$$W_{t+1} = W_t - \eta \nabla L(W_t)$$

여기서, W_t는 현재 가중치, W_{t+1}는 업데이트된 가중치,

- η는 학습률(learning rate).
- $\nabla L(W_t)$는 비용 함수 L에 대한 그래디언트(gradient)입니다.

ADAM

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla f(x_{t-1})$$

$$g_t = \beta_2 g_{t-1} + (1-\beta_2)(\nabla f(x_{t-1}))^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \hat{g}_t = \frac{g_t}{1-\beta_2^t}$$

$$x_t = x_{t-1} - \frac{\eta}{\sqrt{\hat{g}_t + \epsilon}} \cdot \hat{m}_t$$

- β_1: Momentum의 지수이동평균 ≈ 0.9
- β_2: RMSProp의 지수이동평균 ≈ 0.999
- $\hat{m}, \hat{g}$: 학습 초기 시 m_t, g_t가 0이 되는 것을 방지하기 위한 보정 값
- ϵ: 분모가 0이 되는 것을 방지하기 위한 작은 값 $\approx 10^{-8}$
- η: 학습률 ≈ 0.001

optimizer	VRAM 사용량
SGD	가중치의 크기
Adam	가중치의 크기 + m + g

Insipration

over-parameterized model이 실제로 낮은 low intrinsic dimension에 있다. 가정.

Measuring the Intrinsic Dimension of Objective Landscapes

Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning Aghajanyan

LoRA: Low-Rank Adaptation
LoRA를 사용하면?

Pre-trained weight를 고정된 상태(freeze)로 유지하면서 Adaptation 중 dense layer의 변화에 대한 rank decomposition matrices를 최적화
이를 통해 신경망의 일부 dense layer를 간접적으로 훈련시키는 것이 가능

- LoRA는 trainable parameter의 수가 적고 학습 처리량이 높으며 inference latency가 이전 연구 대비 적음
- 그럼에도 불구하고 ROBERTa, DeBERTa, GPT-2, GPT-3에서 fine-tuning보다 같거나 더 나은 성능 을 보여줌

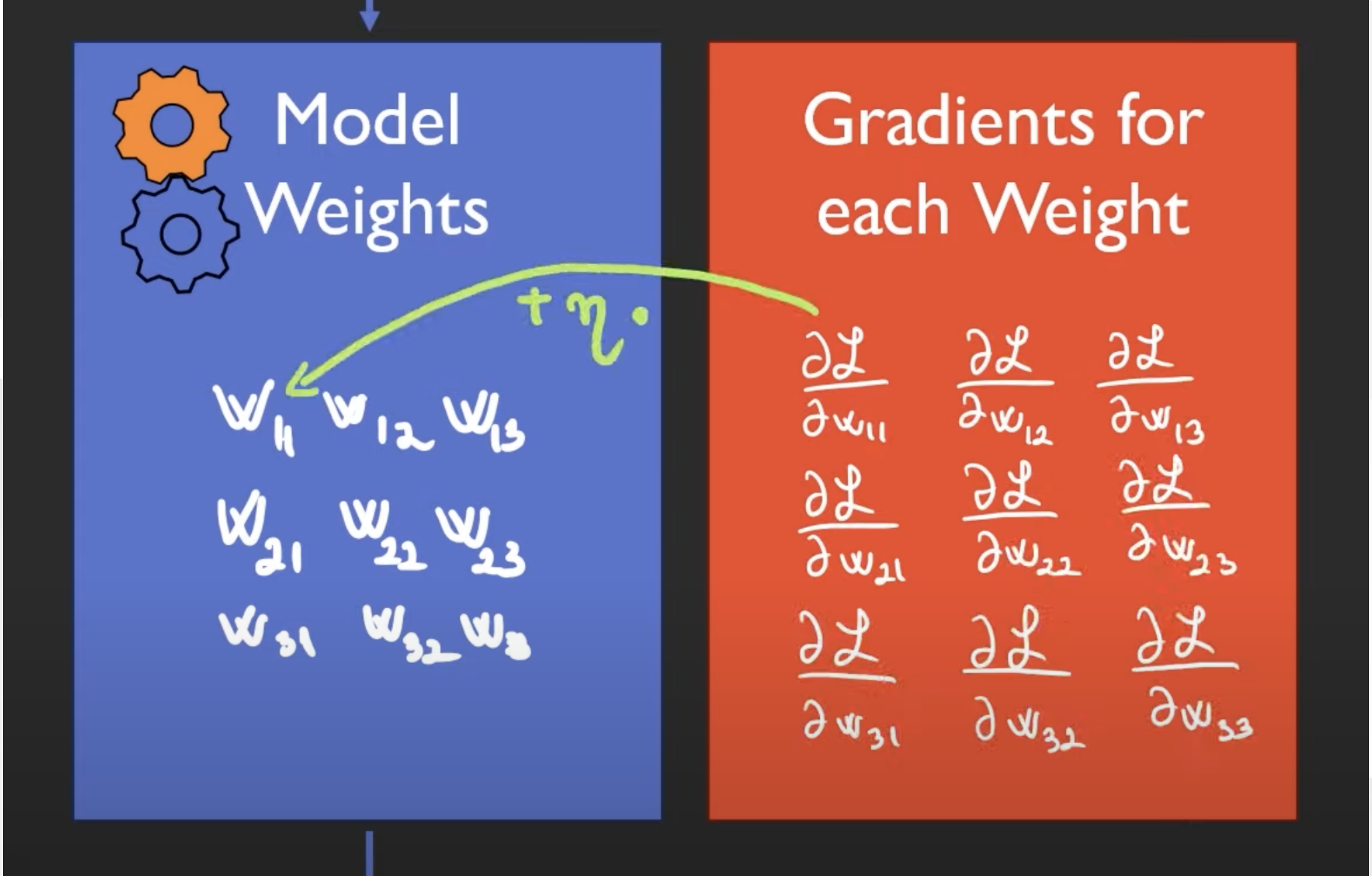
Terminologies and Conventions

d_{model} : input/output dimension size 768, 1024

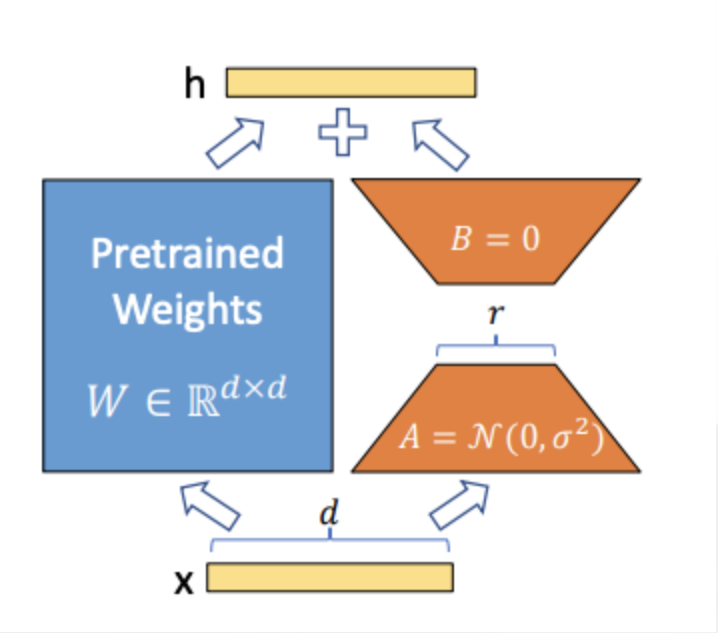
W_q, W_k, W_v, Wo : query/key/value/output projection matrices in self-attention module

W, W_o: pre-trained weight matrix
ΔW: accumulated gradient update during adaptation

r : rank of a LORA module
Transformers 논문의 setting을 따름
e.g., use Adam optimizer
= 4x dmodel


Model Weights
Gradients for each Weight
$+\eta \cdot$
w_{11} w_{12} w_{13}
w_{21} w_{22} w_{23}
w_{31} w_{32} w_{33}
$\frac{\partial \mathcal{L}}{\partial w_{11}}$ $\frac{\partial \mathcal{L}}{\partial w_{12}}$ $\frac{\partial \mathcal{L}}{\partial w_{13}}$
$\frac{\partial \mathcal{L}}{\partial w_{21}}$ $\frac{\partial \mathcal{L}}{\partial w_{22}}$ $\frac{\partial \mathcal{L}}{\partial w_{23}}$
$\frac{\partial \mathcal{L}}{\partial w_{31}}$ $\frac{\partial \mathcal{L}}{\partial w_{32}}$ $\frac{\partial \mathcal{L}}{\partial w_{33}}$

LoRA fintuning method



$$W(finetuning) \rightarrow W(freeze) + \nabla W$$

$$W \in R^{d \times k}$$

$$\nabla W = BA$$

$$A \in R^{d \times r}, B \in R^{r \times k}$$

Fully fintuning

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

LoRA

$$\max_{\Theta} \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{|y|} \log\left(P_{\Theta_0+\Delta\Phi(\Theta)}\left(y_t \mid x, y_{<t}\right)\right)$$

- $\Delta\Phi = \Delta\Phi(\Theta)$: the task-specific parameter increment
- 훨씬 작은 set of parameters Θ s.t $\Theta \ll \left|\Phi_0\right|$

fintunig으로 점점커지는 LLM을 감당하기 힘들기에, 그걸 효율적으로 파인튜닝 시키는 방법.

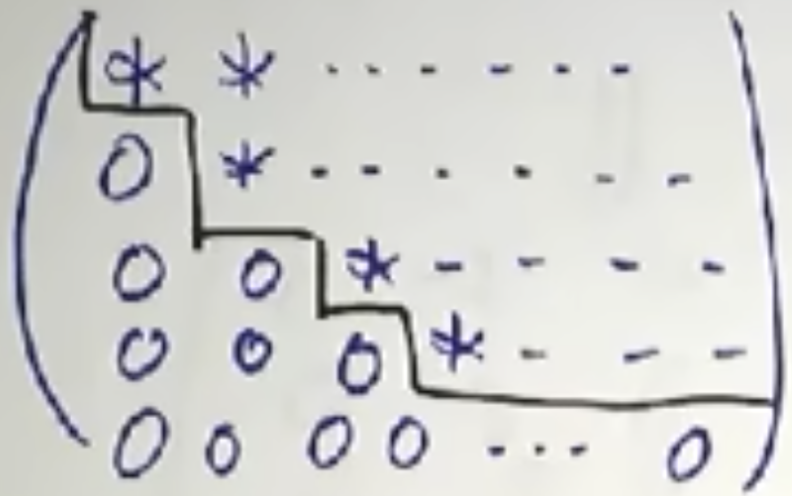
what is Rank?

- numbers how many linearly independent columns in matrix. = rank of columns = rank of rows
- rank : row reduced echelon form의 1의 개수

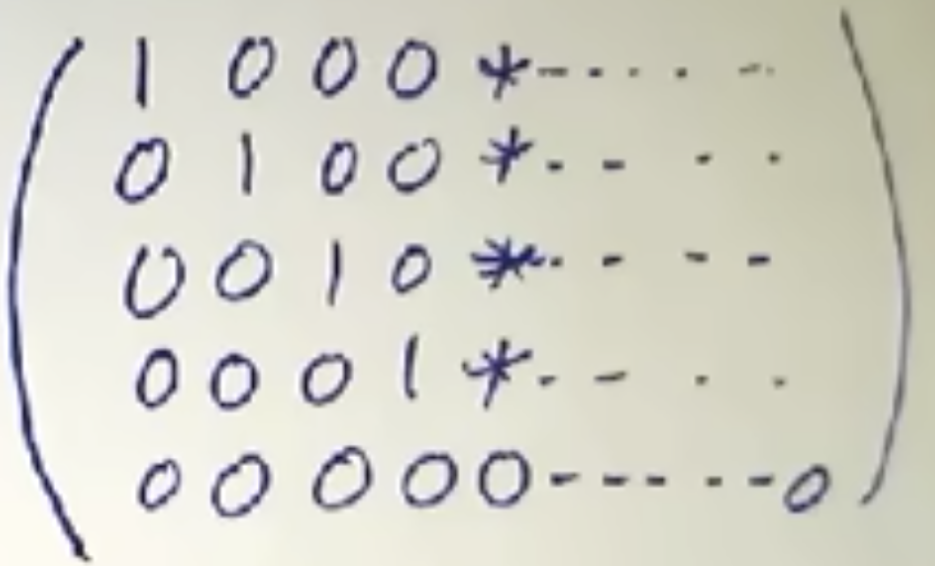
rref

elementary matrix를 곱해서 만들수있음.

Goal: Put matrix in REF or RREF



REF



RREF

$$A = \begin{pmatrix} 2 & -2 & 4 & \\ 2 & 1 & 10 & 7 \\ -4 & 4 & -8 & 4 \\ 4 & -1 & 14 & 6 \end{pmatrix}$$

RREF ↓.

$$E_k \cdots E_2 E_1 A = \begin{pmatrix} 1 & 0 & 4 & 0 \\ 0 & 1 & 2 & 0 \\ & & & 1 \end{pmatrix}$$

$A \sim_r B$
$A \sim_r C$
$B = C$

rref의 유일성과 존재성 증명됨.

LoRA

$$W_0 + \Delta W = W_0 + BA, \quad B \in \mathbb{R}^{d \times r}$$

freeze pretrained — trainable LoRA params

if $A \in \mathbb{R}^{r \times k}$ $\quad r \ll \min(d, k)$

$A \sim r \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \\ & & & \\ & & & \end{pmatrix}$ (k columns)

$\max(\mathrm{rk}(A)) = r$

only adapting the attention weights

W_q, W_k, W_v, W_o

freeze MLP modules

Alternatives

1. adding Adapter Layer in each transformer block
2. Prefix tuning

reference

youtube channels : 딥러닝논문읽기모임

https://www.youtube.com/watch?v=BJqwmDpa0wM

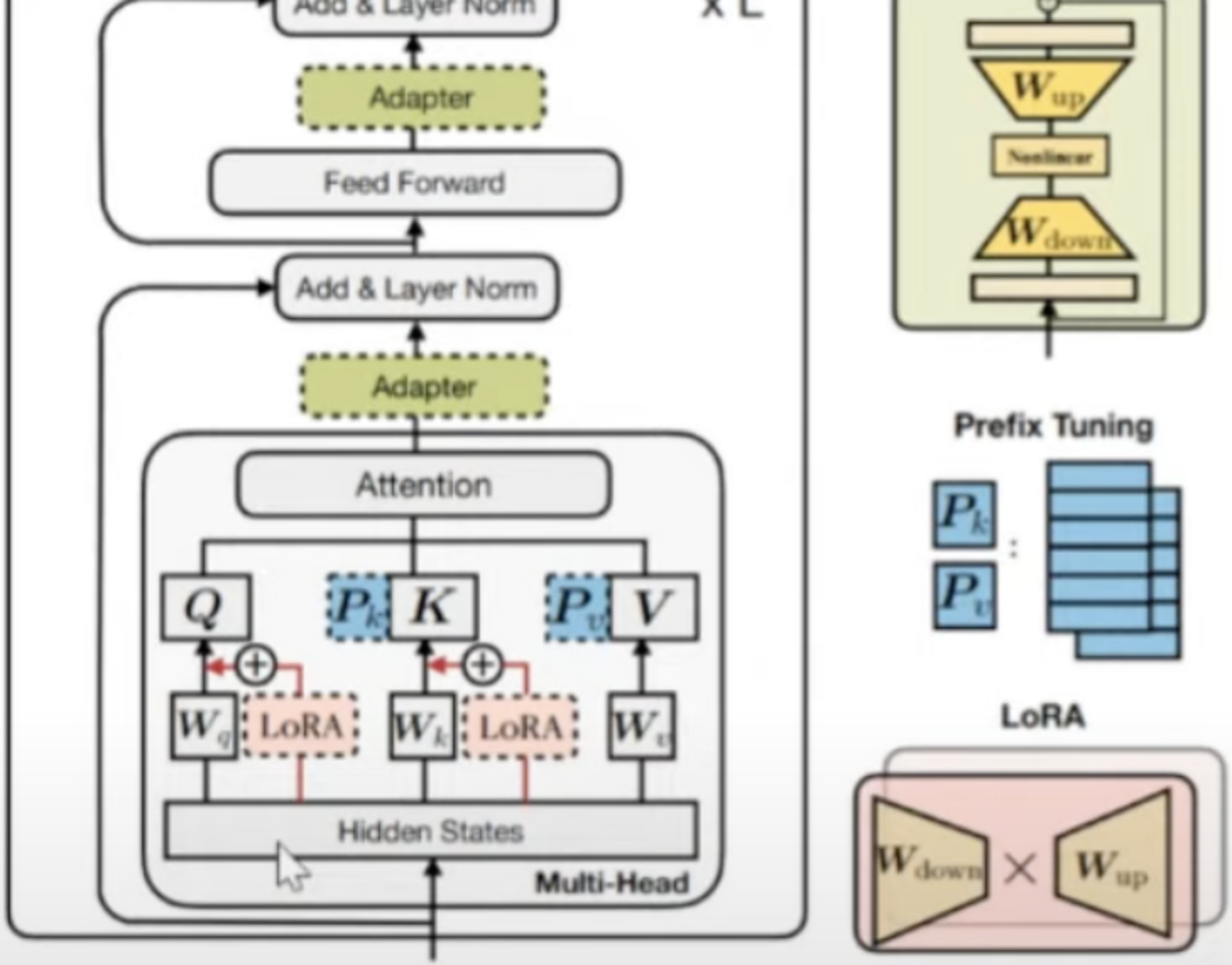
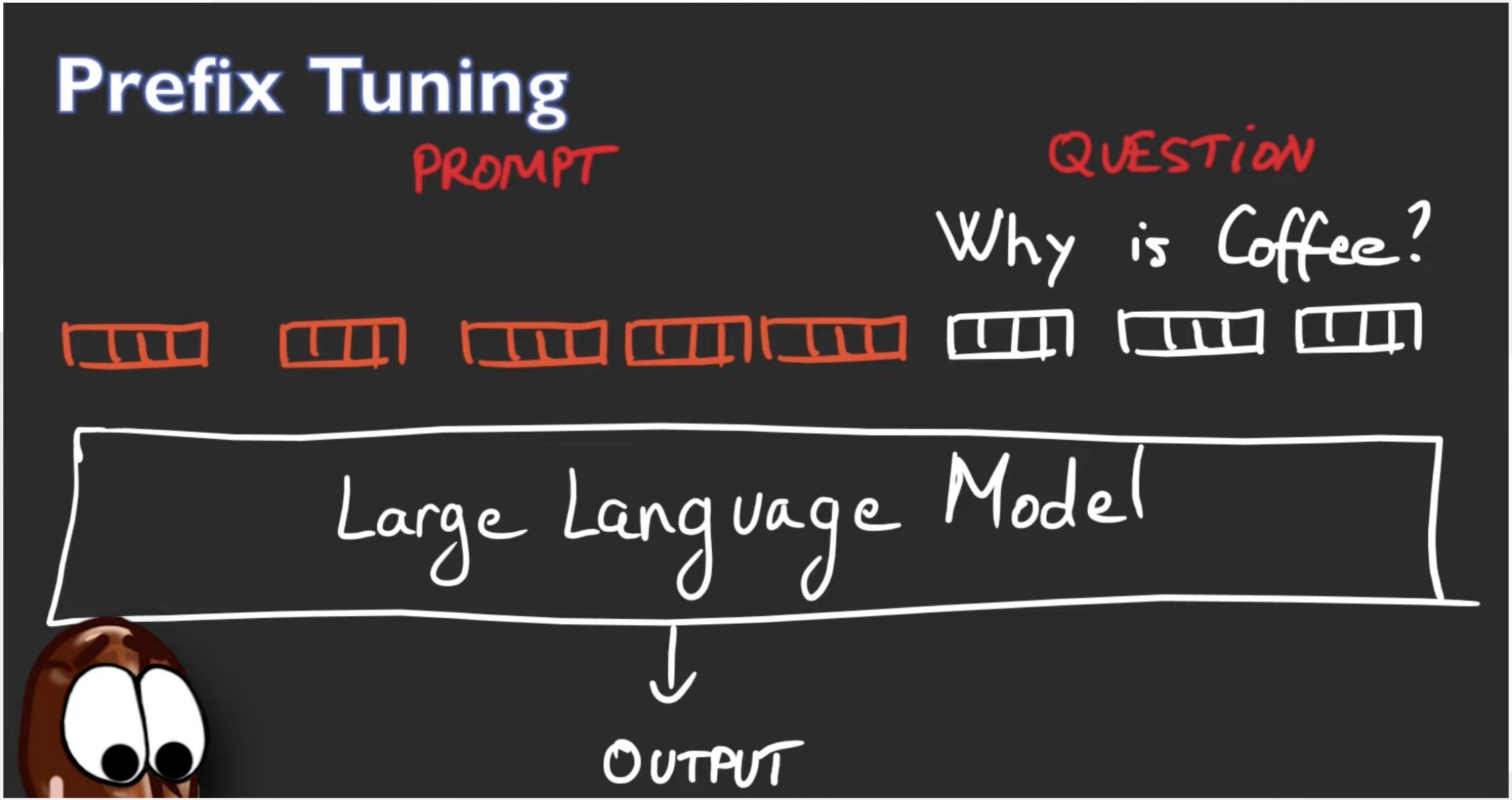


Figure 1: Illustration of the transformer architecture and several state-of-the-art parameter-efficient tuning methods. We use blocks with dashed borderlines to

1. Prefix tuning



https://www.youtube.com/watch?v=KEv-F5UkhxU

LoRA

원래 weight에 BA를 통해서 weight_0에서 variants를 가하겠다.

CODE

https://github.com/microsoft/LoRA

A,B Definition

```
class Linear(nn.Linear, LoRALayer):
    # LoRA implemented in a dense layer

        # Actual trainable parameters

    ###################################################
    ################   Definition !!!!!   ##############
        if r > 0:
            self.lora_A = nn.Parameter(self.weight.new_zeros((r, in_features)))
            self.lora_B = nn.Parameter(self.weight.new_zeros((out_features, r)))
            self.scaling = self.lora_alpha / self.r
            # Freezing the pre-trained weight matrix
            self.weight.requires_grad = False
        self.reset_parameters()
        if fan_in_fan_out:
            self.weight.data = self.weight.data.transpose(0, 1)
```

A,B initalize

```
def reset_parameters(self):
    self.conv.reset_parameters()
    if hasattr(self, 'lora_A'):
        # initialize A the same way as the default for nn.Linear and B to zero
        nn.init.kaiming_uniform_(self.lora_A, a=math.sqrt(5))
        nn.init.zeros_(self.lora_B)
```

```
    # LoRA implemented in a dense layer
    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)
```

elementwise sum

```
    def train(self, mode: bool = True):
        def T(w):
            return w.transpose(0, 1) if self.fan_in_fan_out else w
        nn.Linear.train(self, mode)
        if mode:
            if self.merge_weights and self.merged:
                # Make sure that the weights are not merged
                if self.r > 0:
                    self.weight.data -= T(self.lora_B @ self.lora_A) * self.scaling
                self.merged = False
```

Experiments

Model & Method	# Trainable Parameters	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
RoB_{base} (FT)*	125.0M	**87.6**	94.8	90.2	**63.6**	92.8	**91.9**	78.7	91.2	86.4
RoB_{base} (BitFit)*	0.1M	84.7	93.7	**92.7**	62.0	91.8	84.0	81.5	90.8	85.2
RoB_{base} (Adpt^{D})*	0.3M	$87.1_{\pm.0}$	$94.2_{\pm.1}$	$88.5_{\pm1.1}$	$60.8_{\pm.4}$	$93.1_{\pm.1}$	$90.2_{\pm.0}$	$71.5_{\pm2.7}$	$89.7_{\pm.3}$	84.4
RoB_{base} (Adpt^{D})*	0.9M	$87.3_{\pm.1}$	$94.7_{\pm.3}$	$88.4_{\pm.1}$	$62.6_{\pm.9}$	$93.0_{\pm.2}$	$90.6_{\pm.0}$	$75.9_{\pm2.2}$	$90.3_{\pm.1}$	85.4
RoB_{base} (LoRA)	0.3M	$87.5_{\pm.3}$	$\mathbf{95.1}_{\pm.2}$	$89.7_{\pm.7}$	$63.4_{\pm1.2}$	$\mathbf{93.3}_{\pm.3}$	$90.8_{\pm.1}$	$\mathbf{86.6}_{\pm.7}$	$\mathbf{91.5}_{\pm.2}$	**87.2**
$\text{RoB}_{\text{large}}$ (FT)*	355.0M	90.2	**96.4**	**90.9**	68.0	94.7	**92.2**	86.6	92.4	88.9
$\text{RoB}_{\text{large}}$ (LoRA)	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.9}_{\pm1.2}$	$\mathbf{68.2}_{\pm1.9}$	$\mathbf{94.9}_{\pm.3}$	$91.6_{\pm.1}$	$\mathbf{87.4}_{\pm2.5}$	$\mathbf{92.6}_{\pm.2}$	**89.0**
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	3.0M	$90.2_{\pm.3}$	$96.1_{\pm.3}$	$90.2_{\pm.7}$	$\mathbf{68.3}_{\pm1.0}$	$\mathbf{94.8}_{\pm.2}$	$\mathbf{91.9}_{\pm.1}$	$83.8_{\pm2.9}$	$92.1_{\pm.7}$	88.4
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	0.8M	$\mathbf{90.5}_{\pm.3}$	$\mathbf{96.6}_{\pm.2}$	$89.7_{\pm1.2}$	$67.8_{\pm2.5}$	$\mathbf{94.8}_{\pm.3}$	$91.7_{\pm.2}$	$80.1_{\pm2.9}$	$91.9_{\pm.4}$	87.9
$\text{RoB}_{\text{large}}$ (Adpt^{H})†	6.0M	$89.9_{\pm.5}$	$96.2_{\pm.3}$	$88.7_{\pm2.9}$	$66.5_{\pm4.4}$	$94.7_{\pm.2}$	$92.1_{\pm.1}$	$83.4_{\pm1.1}$	$91.0_{\pm1.7}$	87.8
$\text{RoB}_{\text{large}}$ (Adpt^{H})†	0.8M	$90.3_{\pm.3}$	$96.3_{\pm.5}$	$87.7_{\pm1.7}$	$66.3_{\pm2.0}$	$94.7_{\pm.2}$	$91.5_{\pm.1}$	$72.9_{\pm2.9}$	$91.5_{\pm.5}$	86.4
$\text{RoB}_{\text{large}}$ (LoRA)†	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.2}_{\pm1.0}$	$68.2_{\pm1.9}$	$\mathbf{94.8}_{\pm.3}$	$91.6_{\pm.2}$	$\mathbf{85.2}_{\pm1.1}$	$\mathbf{92.3}_{\pm.5}$	**88.6**
DeB_{XXL} (FT)*	1500.0M	91.8	**97.2**	92.0	72.0	**96.0**	92.7	93.9	92.9	91.1
DeB_{XXL} (LoRA)	4.7M	$\mathbf{91.9}_{\pm.2}$	$96.9_{\pm.2}$	$\mathbf{92.6}_{\pm.6}$	$\mathbf{72.4}_{\pm1.1}$	$\mathbf{96.0}_{\pm.1}$	$\mathbf{92.9}_{\pm.1}$	$\mathbf{94.9}_{\pm.4}$	$\mathbf{93.0}_{\pm.2}$	**91.3**

Table 2: $\text{RoBERTa}_{\text{base}}$, $\text{RoBERTa}_{\text{large}}$, and $\text{DeBERTa}_{\text{XXL}}$ with different adaptation methods on the GLUE benchmark. We report the overall (matched and mismatched) accuracy for MNLI, Matthew's correlation for CoLA, Pearson correlation for STS-B, and accuracy for other tasks. Higher is better for all metrics. * indicates numbers published in prior works. † indicates runs configured in a setup similar to Houlsby et al. (2019) for a fair comparison.

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 (Adapter^{L})*	0.37M	66.3	8.41	45.0	69.8	2.40
GPT-2 M (Adapter^{L})*	11.09M	68.9	8.71	46.1	71.3	2.47
GPT-2 M (Adapter^{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 (Adapter^{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 (Adapter^{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	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.89}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{72.0}_{\pm.2}$	$2.47_{\pm.02}$

Table 3: GPT-2 medium (M) and large (L) with different adaptation methods on the E2E NLG Challenge. For all metrics, higher is better. LoRA outperforms several baselines with comparable or fewer trainable parameters. Confidence intervals are shown for experiments we ran. * indicates numbers published in prior works.

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 ($\text{Adapter}^{\text{H}}$)	7.1M	71.9	89.8	53.0/28.9/44.8
GPT-3 ($\text{Adapter}^{\text{H}}$)	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

Table 4: Performance of different adaptation methods on GPT-3 175B. We report the logical form validation accuracy on WikiSQL, validation accuracy on MultiNLI-matched, and Rouge-1/2/L on SAMSum. LoRA performs better than prior approaches, including full fine-tuning. The results on WikiSQL have a fluctuation around $\pm 0.5\%$, MNLI-m around $\pm 0.1\%$, and SAMSum around $\pm 0.2/\pm 0.2/\pm 0.1$ for the three metrics.

	# of Trainable Parameters = 18M						
Weight Type	W_q	W_k	W_v	W_o	W_q, W_k	W_q, W_v	W_q, W_k, W_v, W_o
Rank r	8	8	8	8	4	4	2
WikiSQL ($\pm 0.5\%$)	70.4	70.0	73.0	73.2	71.4	**73.7**	**73.7**
MultiNLI ($\pm 0.1\%$)	91.0	90.8	91.0	91.3	91.3	91.3	**91.7**

Table 5: Validation accuracy on WikiSQL and MultiNLI after applying LoRA to different types of attention weights in GPT-3, given the same number of trainable parameters. Adapting both W_q and W_v gives the best performance overall. We find the standard deviation across random seeds to be consistent for a given dataset, which we report in the first column.

	Weight Type	$r = 1$	$r = 2$	$r = 4$	$r = 8$	$r = 64$
WikiSQL($\pm 0.5\%$)	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 ($\pm 0.1\%$)	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

Table 6: Validation accuracy on WikiSQL and MultiNLI with different rank r. To our surprise, a rank as small as one suffices for adapting both W_q and W_v on these datasets while training W_q alone needs a larger r. We conduct a similar experiment on GPT-2 in Section H.2.

This suggests the update matrix ΔW could have a very small

“intrinsic rank"

$$\phi(A_{r=8}, A_{r=64}, i, j) = \frac{||U^{i\top}_{A_{r=8}} U^{j}_{A_{r=64}}||^2_F}{\min(i,j)} \in [0,1] \tag{4}$$

where $U^{i}_{A_{r=8}}$ represents the columns of $U_{A_{r=8}}$ corresponding to the top-i singular vectors.

$\phi(\cdot)$ has a range of $[0, 1]$, where 1 represents a complete overlap of subspaces and 0 a complete separation. See Figure 3 for how ϕ changes as we vary i and j. We only look at the 48th layer (out of 96) due to space constraint, but the conclusion holds for other layers as well, as shown in Section H.1.

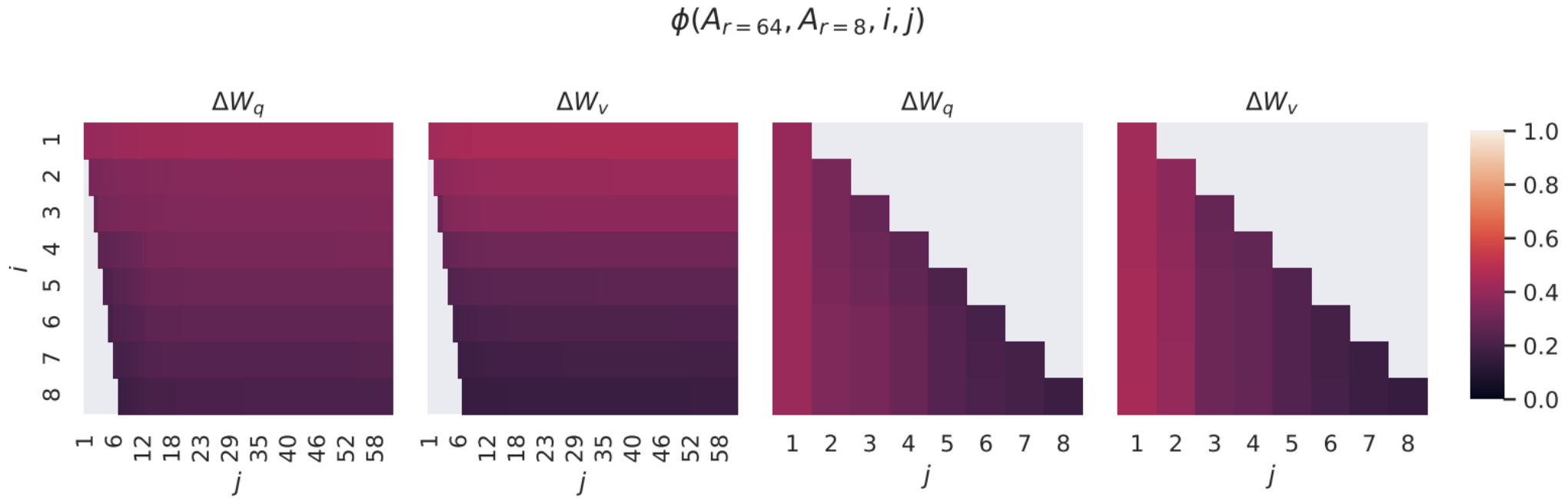


Figure 3: Subspace similarity between column vectors of $A_{r=8}$ and $A_{r=64}$ for both ΔW_q and ΔW_v. The third and the fourth figures zoom in on the lower-left triangle in the first two figures. The top directions in $r = 8$ are included in $r = 64$, and vice versa.

$$\phi(A, B, i, j) = \psi(U_A^i, U_B^j) = \frac{\sum_{i=1}^{p} \sigma_i^2}{p} = \frac{1}{p}\left(1 - d(U_A^i, U_B^j)^2\right)$$

This similarity satisfies that if U_A^i and U_B^j share the same column span, then $\phi(A, B, i, j) = 1$. If they are completely orthogonal, then $\phi(A, B, i, j) = 0$. Otherwise, $\phi(A, B, i, j) \in (0, 1)$.

conculusion

Conclusion

- Large-scale language model을 효율적으로 튜닝하는 LORA 제안
- Adapter 류의 기법과 다르게 inference latency가 발생하지 않음
- Prefix-tuning과 다르게 usable sequence length를 줄일 필요가 없음
- 가중치 업데이트 행렬이 low intrinsic rank를 가진다고 가정
- 논문에선 LM에 초점을 맞췄지만 이론적으로 모든 dense layer에 적용가능

END

읽어주셔서 감사합니다.

-Shin joong hyun-