AdaLoRA Review

ADALoRA: Adaptive Budget Allocation
for Parameter-Efficient Fine-Tuning

Sangho Kim

Contents

Introduction

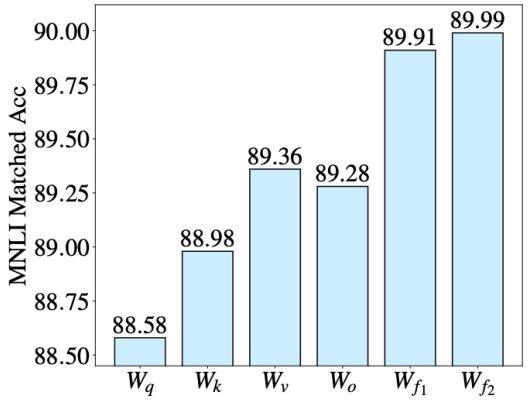
- 이 논문을 단 한 문장으로 요약하자면 다음과 같이 표현할 수 있다고 생각한다

How can we allocate the parameter budget adaptively according to importance of modules to improve the performance of parameter-efficient fine-tuning?

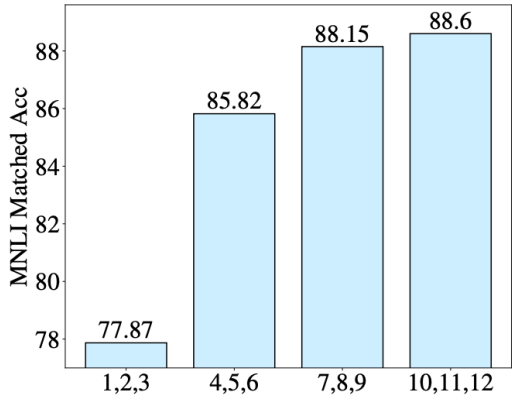
- 즉, LoRA와 달리 성능을 효율적으로 향상시킬 수 있는 modules에 집중하자는 것

Introduction

- LoRA는 Transformer 내부의 각 module 마다의 중요성을 고려하지 않았다는 것을 간과함



(a) Selected weight matrix



(b) Selected layers

- 위 그림을 살펴보자
- 왼쪽 그림은 Attention, FFN module 별로 LoRA를 탑재해 성능을 비교한 것
- Attention module보다 FFN module에 탑재하는 것이 더 중요도가 높은 것을 알 수 있음
- 또한, 오른쪽 그림을 통해 bottom layers보다 top layers에 LoRA를 탑재하는 것이 더 효율적임을 알 수 있음

Introduction

- 그래서 이 논문에서는 이러한 딜레마를 해결하기 위해 AdaLoRA를 제안
- AdaLoRA는 각 module 별로 LoRA의 rank를 조정함
- 즉, 중요도가 높은 module에는 high rank, 중요도가 낮은 module는 low rank를 탑재
- LLM을 fine-tuning한다면 SVD를 적용해 계산하기에는 비용이 너무 커 $\Delta = P\Lambda Q$로 parameterize를 수행
- 이때 P와 Q의 orthogonality를 regularize하기 위해 additional penalty가 training loss에 추가 됨
- Low importance를 가지는 Triplets에는 low priority를 가지며 따라서 singular values는 0으로 바꾼다
- 반대로 high importance를 가지는 Triplets에는 유지가 된다
- 또한 이 논문에서는 training을 하기 위해 별도로 global budget scheduler를 제안

Contents

Method

- 먼저 SVD-based Adaptation을 살펴보자
- AdaLoRA는 다음의 SVD form을 가진다

$$W = W^{(0)} + \Delta = W^{(0)} + P\Lambda Q$$

- 이때 $P \in R^{d_1 \times r}$, $Q \in R^{r \times d_2}$이며 이는 Δ의 left, right singular vector 그리고 $\Lambda \in R^{r \times r}$는 diagonal matrix이며 singular values를 표현
- $G_i = \{P_{*i}, \lambda_i, Q_{i*}\}$는 i-th singlular values와 vector를 포함하며 triplet이라고 표현
- Λ는 diagonal하기 때문에 따로 vector로 표현하며 P와 Q는 random Gaussian initialization
- $P^T P = QQ^T = I$를 유지하기 위해 우리는 다음의 regularizer를 활용한다

$$R(P, Q) = \|P^T P - I\|_F^2 + \|QQ^T - I\|_F^2$$

- Λ는 gradient descent를 수행하면서 반복적으로 pruning 됨

Method

- 이제 LoRA의 단점에 대해 살펴보자
- Doublet Pruning이란 아이디어를 가지고 적용을 할 수 있지만 unimportant factor를 찾고 prune를 하면 해당 rank에 대해 모두 수행을 해야 한다
  - 다시 말해, 학습하면서 importance가 높아져도 다시 활성화할 방법이 없음
  - , AdaLoRA는 singular value에만 masking 처리를 하고 singular vector는 살리기 때문에 단점 극복
  - 즉, 학습하면서 importance가 높아져도 이를 다시 활성화할 여지가 존재
- 두 번째로는 LoRA의 A와 B는 서로 orthogonal하지 않아 dependent하다는 것
  - Doublet은 large variation representation이 불가능
  - 자세한 것은 뒤에 Experiments 파트에 실험 결과를 살펴보자

Method

- Importance-aware Rank Allocation에 대해 살펴보자
- Δ_k의 i-th triplet을 $G_{k,i} = \{P_{k,*i}, \Lambda_{k,i}, Q_{k,i*}\}$라 하고 이의 importance score를 $S_{k,i}$라 하자
- 또한 parameter sets $P = \{P_k\}_{k=1}^n$, $\mathrm{E} = \{\Lambda_k\}_{k=1}^n$, $Q = \{Q_k\}_{k=1}^n$이라 하고 training cost를 $C(P, \mathrm{E}, Q)$라 하자
- 그러면 objective는 $L(P, \mathrm{E}, Q) = C(P, \mathrm{E}, Q) + \gamma \sum_{k=1}^n R(P_k, Q_k)$
- 이때 $\gamma > 0$은 regularization coefficient
- $\Lambda_k^{(t)}$는 다음과 같이 gradient descent로 update를 수행

$$\widetilde{\Lambda}_k^{(t)} = \Lambda_k^{(t)} - \eta \nabla_{\Lambda_k} L(P^{(t)}, \mathrm{E}^{(t)}, Q^{(t)})$$

- 이때 $\eta > 0$는 learning rate

Method

- 그러면 importance score $S_k^{(t)}$가 주어지면 singular value는 다음과 같이 pruning을 수행

$$\Lambda_k^{(t+1)} = \mathcal{T}(\tilde{\Lambda}_k^{(t)}, S_k^{(t)}), \text{ with } \mathcal{T}(\tilde{\Lambda}_k^{(t)}, S_k^{(t)})_{ii} = \begin{cases} \tilde{\Lambda}_{k,ii}^{(t)} & S_{k,i}^{(t)} \text{ is in the top-} b^{(t)} \text{ of } S^{(t)}, \\ 0 & \text{otherwise,} \end{cases}$$

- 즉, 특정 score에 미치지 못한 singular values는 0으로 바꾸는 것이다

Method

- 이제 importance score를 어떻게 구할 것인지 살펴보자
- 가장 간단한 방법은 $S_{k,i} = |\lambda_{k,i}|$를 설정해 score를 구하는 것이다
- 그러나 이 논문에서는 각 파라미터가 성능에 어떤 contribution을 가지는지 수치화하기 위해 importance metric을 고안

$$S_{k,i} = s(\lambda_{k,i}) + \frac{1}{d_1}\sum_{j=1}^{d_1} s(P_{k,ji}) + \frac{1}{d_2}\sum_{j=1}^{d_2} s(Q_{k,ij})$$

- 여기서 $s(\cdot)$은 single entries에 대한 importance function
- $s(\cdot)$에 대해 sensitivity를 사용할 수 있으며 이는 gradient-weight product의 크기로 정의됨

$$I(w_{ij}) = |w_{ij}\nabla_{w_{ij}}\mathcal{L}|$$

- 그러나 타 논문에서 위 식은 sensitivity가 신뢰할 수 있는 importance metric이 아니라고 지적
- 이 score는 sampled mini batch에서 추정되며 stochastic sampling과 complicated training dynamics은 sensitivity를 estimate하기에 불안을 일으킴

Method

- 따라서 아래와 같이 exponential moving average 아이디어를 적용해 문제를 해결

$$\overline{I}^{(t)}(w_{ij}) = \beta_1 \overline{I}^{(t-1)}(w_{ij}) + (1-\beta_1) I^{(t)}(w_{ij})$$
$$\overline{U}^{(t)}(w_{ij}) = \beta_2 \overline{U}^{(t-1)}(w_{ij}) + (1-\beta_2) \left| I^{(t)}(w_{ij}) - \overline{I}^{(t)}(w_{ij}) \right|$$

- 이때 $0 < \beta_1, \beta_2 < 1$
- 그리고 $\bar{I}^{(t)}$는 EMA에 의한 smooth sensitivity, $\bar{U}^{(t)}$는 $I^{(t)}$와 $\bar{I}^{(t)}$ 사이의 loca variation에 의해 정량화된 uncertainty
- 그리고 importance는 아래와 같이 $\bar{I}^{(t)}$와 $\bar{U}^{(t)}$의 product로 정의한다

$$s^{(t)}(w_{ij}) = \overline{I}^{(t)}(w_{ij}) \cdot \overline{U}^{(t)}(w_{ij})$$

Method

- 앞에서 서술한 것을 정리해 AdaLoRA의 algorithm을 표현하면 다음과 같음

Algorithm 1 AdaLoRA

Input: Dataset $\mathcal{D}$; total iterations T; budget schedule $\{b^{(t)}\}_{t=0}^{T}$; hyperparameters $\eta, \gamma, \beta_1, \beta_2$.
for $t = 1, \dots, T$ **do**
  Sample a mini-batch from $\mathcal{D}$ and compute the gradient $\nabla \mathcal{L}(\mathcal{P}, \mathcal{E}, \mathcal{Q})$;
  Compute the sensitivity $I^{(t)}$ in (8) for every parameter in $\{\mathcal{P}, \mathcal{E}, \mathcal{Q}\}$;
  Update $\overline{I}^{(t)}$ as (9) and $\overline{U}^{(t)}$ as (10) for every parameter in $\{\mathcal{P}, \mathcal{E}, \mathcal{Q}\}$;
  Compute $S_{k,i}^{(t)}$ by (7), for $k = 1, \dots, n$ and $i = 1, \dots, r$;
  Update $P_k^{(t+1)} = P_k^{(t)} - \eta \nabla_{P_k} \mathcal{L}(\mathcal{P}, \mathcal{E}, \mathcal{Q})$ and $Q_k^{(t+1)} = Q_k^{(t)} - \eta \nabla_{Q_k} \mathcal{L}(\mathcal{P}, \mathcal{E}, \mathcal{Q})$;
  Update $\Lambda_k^{(t+1)} = \mathcal{T}(\Lambda_k^{(t)} - \eta \nabla_{\Lambda_k} \mathcal{L}(\mathcal{P}, \mathcal{E}, \mathcal{Q}), S_k^{(t)})$ given the budget $b^{(t)}$.
end for
Output: The fine-tuned parameters $\{\mathcal{P}^{(T)}, \mathcal{E}^{(T)}, \mathcal{Q}^{(T)}\}$.

Contents

Experiments

- 실험 결과를 보자

Method	# Params	MNLI m/mm	SST-2 Acc	CoLA Mcc	QQP Acc/F1	QNLI Acc	RTE Acc	MRPC Acc	STS-B Corr	All Ave.
Full FT	184M	89.90/90.12	95.63	69.19	**92.40/89.80**	94.03	83.75	89.46	91.60	88.09
BitFit	0.1M	89.37/89.91	94.84	66.96	88.41/84.95	92.24	78.70	87.75	91.35	86.02
HAdapter	1.22M	90.13/90.17	95.53	68.64	91.91/89.27	94.11	84.48	89.95	91.48	88.12
PAdapter	1.18M	90.33/90.39	95.61	68.77	92.04/89.40	94.29	85.20	89.46	91.54	88.24
LoRA$_{r=8}$	1.33M	90.65/90.69	94.95	69.82	91.99/89.38	93.87	85.20	89.95	91.60	88.34
AdaLoRA	1.27M	**90.76/90.79**	**96.10**	**71.45**	**92.23/89.74**	**94.55**	**88.09**	**90.69**	**91.84**	**89.31**
HAdapter	0.61M	90.12/90.23	95.30	67.87	91.65/88.95	93.76	85.56	89.22	91.30	87.93
PAdapter	0.60M	90.15/90.28	95.53	69.48	91.62/88.86	93.98	84.12	89.22	91.52	88.04
HAdapter	0.31M	90.10/90.02	95.41	67.65	91.54/88.81	93.52	83.39	89.25	91.31	87.60
PAdapter	0.30M	89.89/90.06	94.72	69.06	91.40/88.62	93.87	84.48	89.71	91.38	87.90
LoRA$_{r=2}$	0.33M	90.30/90.38	94.95	68.71	91.61/88.91	94.03	85.56	89.71	**91.68**	88.15
AdaLoRA	0.32M	**90.66/90.70**	**95.80**	**70.04**	**91.78/89.16**	**94.49**	**87.36**	**90.44**	91.63	**88.86**

- LoRA와 비교해 미미하지만 조금 더 효율적임을 알 수 있다

Experiments

Table 2: Results with DeBERTaV3-base on SQuAD v1.1 and SQuADv2.0. Here *# Params* is the number of trainable parameters relative to that in full fine-tuning. We report EM/F1. The best results in each setting are shown in **bold**.

	SQuADv1.1				**SQuADv2.0**			
Full FT	86.0 / 92.7				85.4 / 88.4			
# Params	0.08%	0.16%	0.32%	0.65%	0.08%	0.16%	0.32%	0.65%
HAdapter	84.4/91.5	85.3/92.1	86.1/92.7	86.7/92.9	83.4/86.6	84.3/87.3	84.9/87.9	85.4/88.3
PAdapter	84.4/91.7	85.9/92.5	86.2/92.8	86.6/93.0	84.2/87.2	84.5/87.6	84.9/87.8	84.5/87.5
LoRA	86.4/92.8	86.6/92.9	86.7/93.1	86.7/93.1	84.7/87.5	83.6/86.7	84.5/87.4	85.0/88.0
AdaLoRA	**87.2/93.4**	**87.5/93.6**	**87.5/93.7**	**87.6/93.7**	**85.6/88.7**	**85.7/88.8**	**85.5/88.6**	**86.0/88.9**

- 이 실험에서도 마찬가지로 LoRA보다 아주 조금 더 좋은 결과를 보여줌

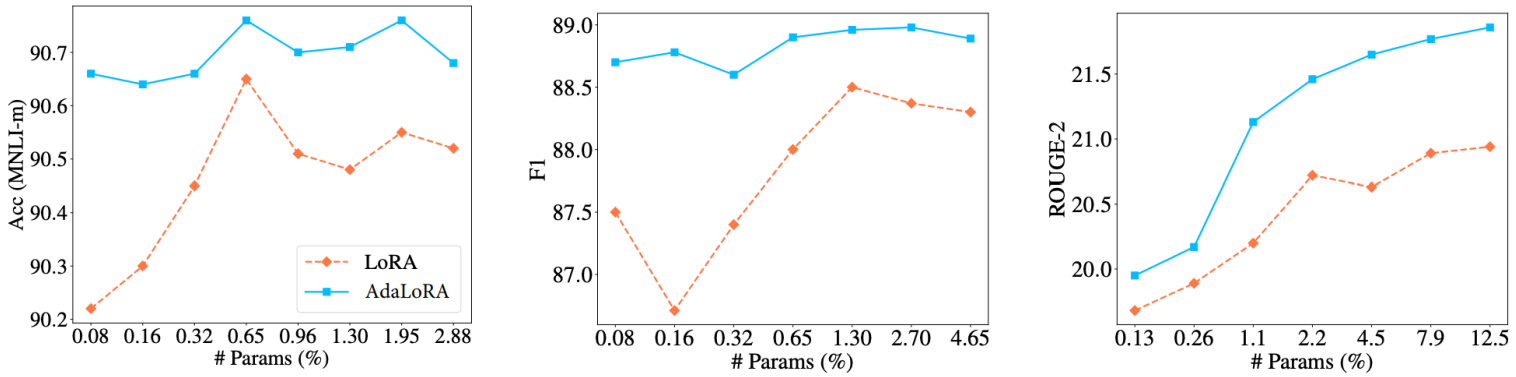
Experiments

Table 3: Results with BART-large on XSum and CNN/DailyMail. Here *# Params* is the number of trainable parameters relative to that in full fine-tuning. We report R-1/2/L. The best results are shown in **bold**.

# Params	Method	XSum	CNN/DailyMail
100%	Full FT	**45.49 / 22.33 / 37.26**	44.16 / 21.28 / 40.90
2.20%	LoRA	43.95 / 20.72 / 35.68	**45.03** / 21.84 / 42.15
	AdaLoRA	**44.72 / 21.46 / 36.46**	45.00 / **21.89 / 42.16**
1.10%	LoRA	43.40 / 20.20 / 35.20	44.72 / 21.58 / 41.84
	AdaLoRA	**44.35 / 21.13 / 36.13**	**44.96 / 21.77 / 42.09**
0.26%	LoRA	43.18 / 19.89 / 34.92	43.95 / 20.91 / 40.98
	AdaLoRA	**43.55 / 20.17 / 35.20**	**44.39 / 21.28 / 41.50**
0.13%	LoRA	42.81 / 19.68 / 34.73	43.68 / 20.63 / 40.71
	AdaLoRA	**43.29 / 19.95 / 35.04**	**43.94 / 20.83 / 40.96**

Experiments

- 아래 그림에서도 LoRA와 비교해 budget 대비 조금 더 좋은 성능을 보여주며 안정적인 모습



(a) MNLI (b) SQuADv2.0 (c) XSum

Figure 2: Fine-tuning performance under different budget levels. We compare AdaLoRA with the generalized LoRA that applies to every weight matrix.

Experiments

- 아래 실험은 importance score에 대해, sensitivity만 사용한 것과 S_i를 $|\lambda_i|$로 사용한 것 그리고 AdaLoRA를 비교

Table 4: We present two ablation studies in this table: (i) Comparison between AdaLoRA and structured pruning on LoRA. (ii) Comparison of different importance metrics for AdaLoRA.

	SST-2			**RTE**			**CoLA**		
# Params	0.08%	0.16%	0.65%	0.08%	0.16%	0.65%	0.08%	0.16%	0.65%
Prune LoRA	94.84	94.50	94.95	86.28	86.15	87.00	66.71	69.29	69.57
AdaLoRA	95.52	95.80	96.10	87.36	87.73	88.09	70.21	70.04	71.45
$s(\cdot) = I(\cdot)$	94.61	95.30	95.64	87.36	87.71	88.10	66.71	68.83	70.19
$S_i = \|\lambda_i\|$	95.41	95.41	95.87	87.00	86.28	88.00	67.67	68.44	70.38

- 결론적으로 역시 그 차이는 근소하지만 AdaLoRA가 가장 좋은 성능을 보여주고 있다

Experiments

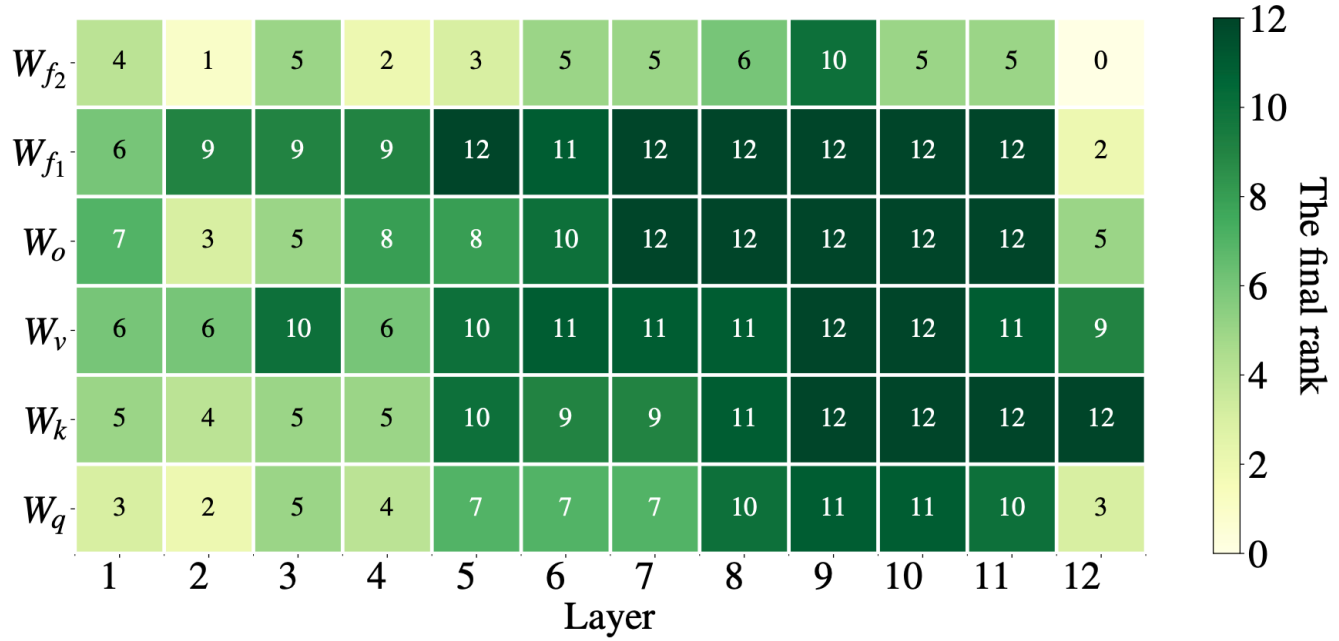
- 이 실험은 LoRA와 SVD-based adaptation, regularization term인 γ을 0으로 설정한 AdaLoRA를 비교

Table 5: We present ablation studies about SVD-based adaptation, orthogonal regularization, and budget allocation in this table. For MNLI, we report the average score of m/mm acc.

	SST-2				**MNLI**			
# Params	0.08%	0.16%	0.32%	0.65%	0.08%	0.16%	0.32%	0.65%
LoRA	94.38	94.95	-	94.95	90.19	90.34	-	90.57
$\text{LoRA}_{\text{regu}}$	-	94.61	94.72	94.61	-	90.30	90.40	90.66
SVD-LoRA	95.33	95.18	95.07	95.53	90.28	90.25	90.52	90.62
$\text{AdaLoRA}_{\gamma = 0}$	95.41	95.10	95.30	95.10	90.37	90.34	90.56	90.43
AdaLoRA	95.64	95.80	96.10	96.10	90.65	90.68	90.66	90.77

Experiments

- 아래 실험은 FFN, Self-attention 그리고 layer의 순서에 따른 최적의 rank의 값을 표현



- Bottom layer에서는 FFN이, Top layer에서는 전체적으로 높은 중요도를 가지고 있음을 알 수 있고
- 최적의 rank를 찾을 수 있다는 것이 AdaLoRA의 가장 큰 장점인 것 같다

Contents

Conclusion

- 이 논문에서는 Adaptive가 가능한 AdaLoRA를 제안했다
- Weight matrices의 update를 SVD로 풀어냄에 따라 parameter budget을 dynamic하게 할당
- 그리고 이 논문에서 제안한 새로운 importance metric에 따라 singular value를 조정
- 따라서 이 method를 통해 LoRA에 비해 조금 더 효율적으로 학습을 할 수 있다

- 논문의 앞부분을 읽었을 때는 단순히 SVD의 아이디어만 적용한 줄 알았는데 singular value까지 조정할 수 있도록 설계를 했단 점에 꽤 놀랐다
- 다만 아쉬운 것은 꽤 좋은 아이디어를 펼쳤음에도 불구하고 실험결과 부분에서 큰 개선이 없었다는 점