2025 하계 세미나

Weight Decomposition

Sogang University
Vision & Display Systems Lab, Dept. of Electronic Engineering

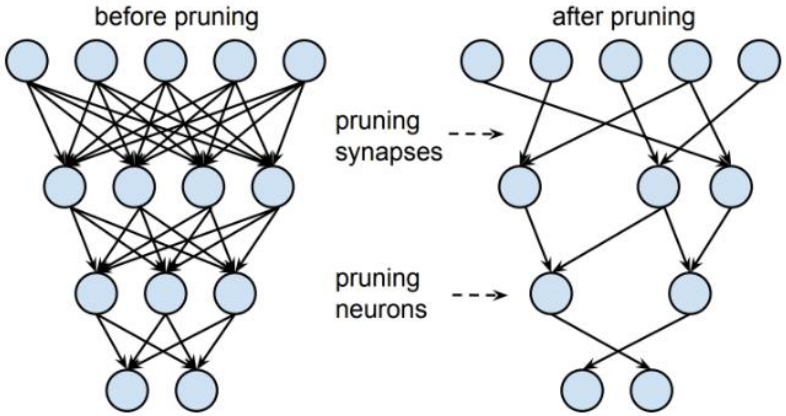
Presented by
유현우

Outline

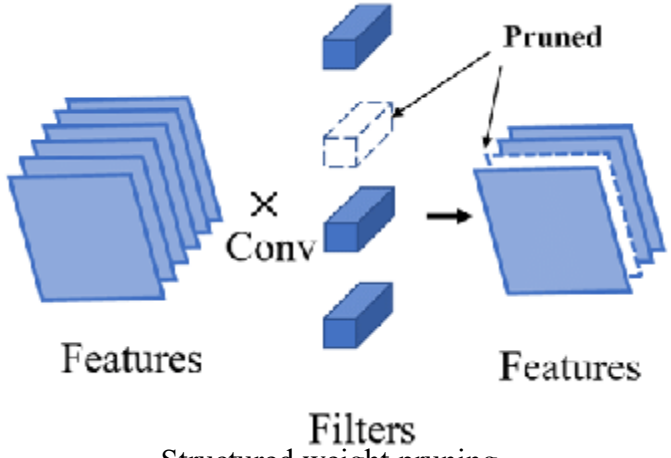
- Background
- ASVD: ACTIVATION-AWARE SINGULAR VALUE DECOMPOSITION FOR COMPRESSING LARGE LANGUAGE MODELS (Arxiv 2023)
- SVD-LLM: TRUNCATION-AWARE SINGULAR VALUE DECOMPOSITION FOR LARGE LANGUAGE MODEL COMPRESSION (ICLR 2025)
- COMCAT: Towards Efficient Compression and Customization of Attention-Based Vision Models (ICML 2023)
- Conclusion

Weight decomposition 이란?

- Unstructured weight pruning
  - 개별 weight 단위로 pruning 하는 접근
    - 주로 작은 값을 갖고 있는 weight를 제거 (0)으로 설정
    - 제거된 가중치가 weight 행렬 내에서 무작위로 흩어져 있으므로 실제 하드웨어 가속기에서 연산 효율성 개선의 한계가 있음
- Structured weight pruning
  - 구조 단위 (channel, block 등) 으로 가중치를 제거
    - 큰 단위의 구조를 제거하기 때문에 하드웨어 상에서 연산량과 메모리 사용량의 효과가 큼



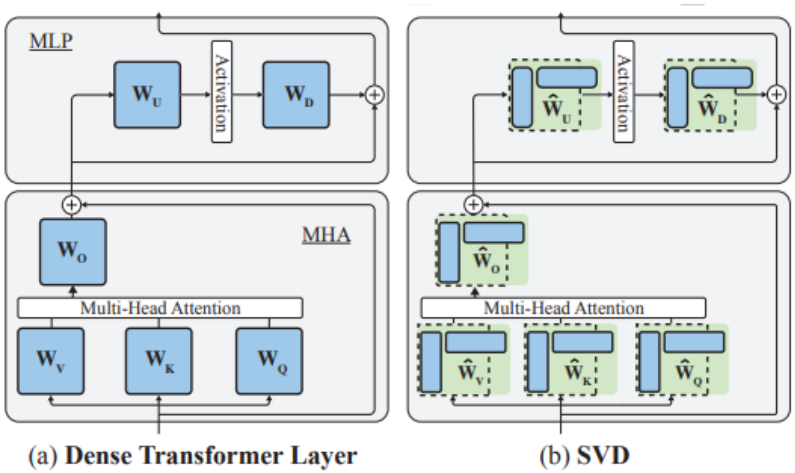
Unstructured weight pruning



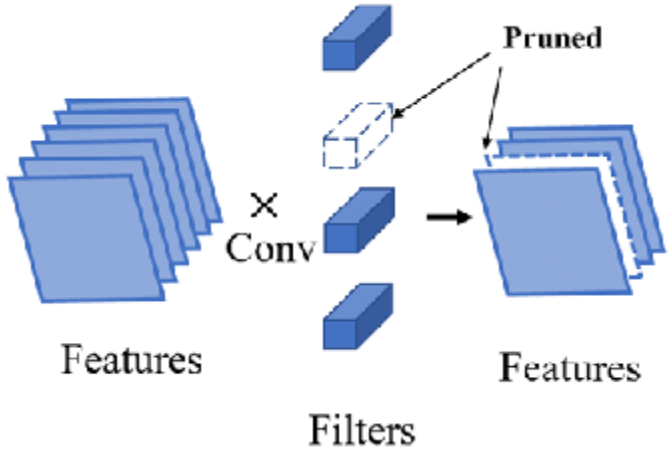
Structured weight pruning

Weight decomposition 이란?

- Structured weight pruning
  - 구조 단위 (channel, block 등) 으로 가중치를 제거
    - 큰 단위의 구조를 제거하기 때문에 하드웨어 상에서 연산량과 메모리 사용량의 효과가 큼
  - Weight decomposition 은 weight 를 Singular Value Decomposition (SVD) 등의 기법으로 decompose 해서 low-rank approximation 하는 방법
    - 기존 weight의 정보를 얼마나 잘 보존하면서 truncate 를 할 것인지가 중요



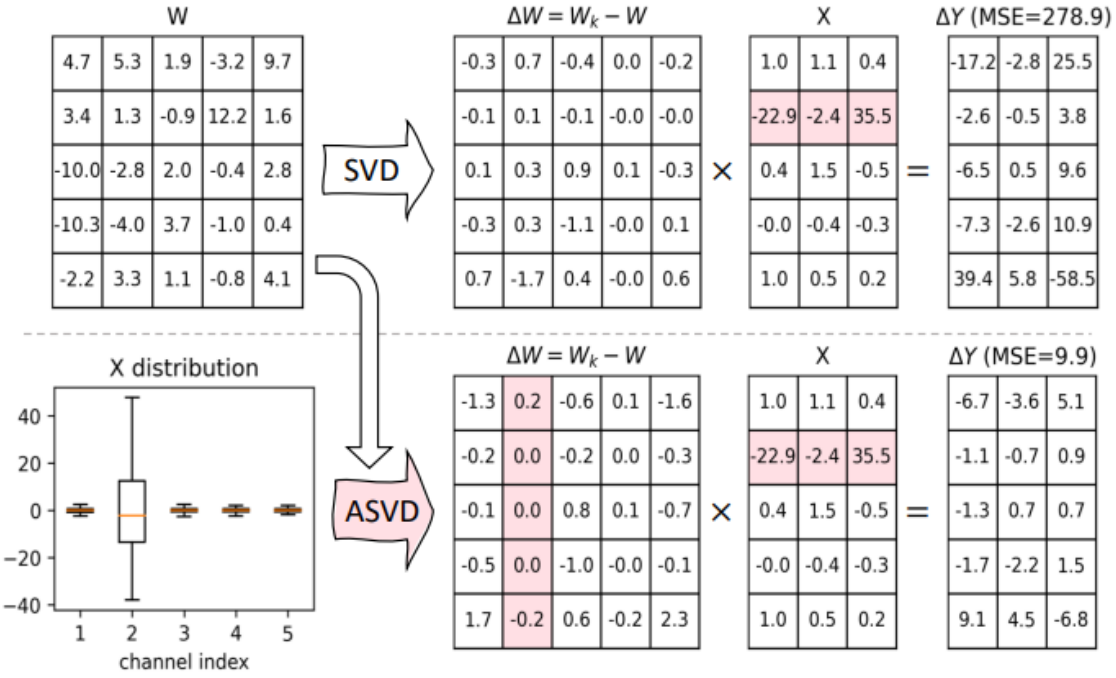
Weight decomposition



Structured weight pruning

ASVD[1]

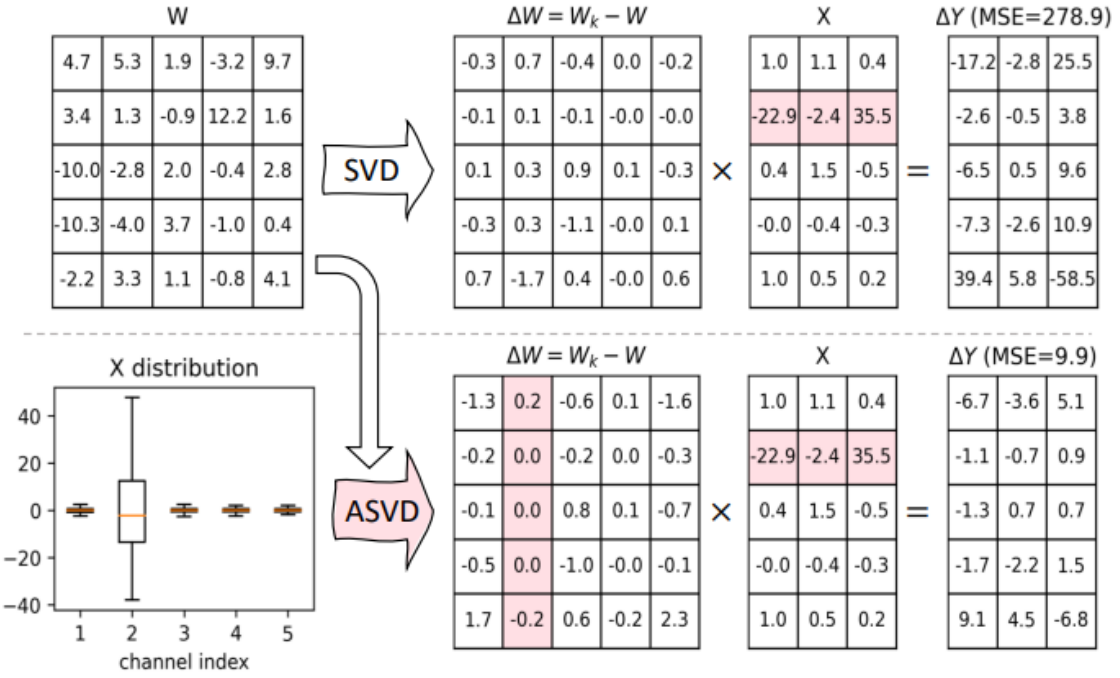
- 데이터를 고려한 decomposition 방법 제안
  - Weight (W)와 low-rank approximated weight (W_k)
  - Object function을 단순 두 weigh간의 차이로 설정하면 $W_k^* = argmin \|W_k - W\|_F^2$
    - Activation (input 되는 X) 에 channel 에 따른 outlier 때문에 $\|W_k - W\|_F^2$ 는 작아도 WX 까지 고려했을 때 차이가 크게 발생할수도 있음



ASVD 개념도

ASVD

- 데이터를 고려한 decomposition 방법 제안
  - 따라서 calibration set을 활용한 X에 weight 를 가해줬을 때의 차이값을 고려해서 objective 설정 $\mathbf{W}_k^\star = \arg\min_{\mathbf{W}_k} \|\mathbf{W}_k\mathbf{X} - \mathbf{W}\mathbf{X}\|_F^2$
  - $\Delta\mathbf{Y} = (\mathbf{W}_k - \mathbf{W})\mathbf{X}$. 를 최소화 하기 위한 접근

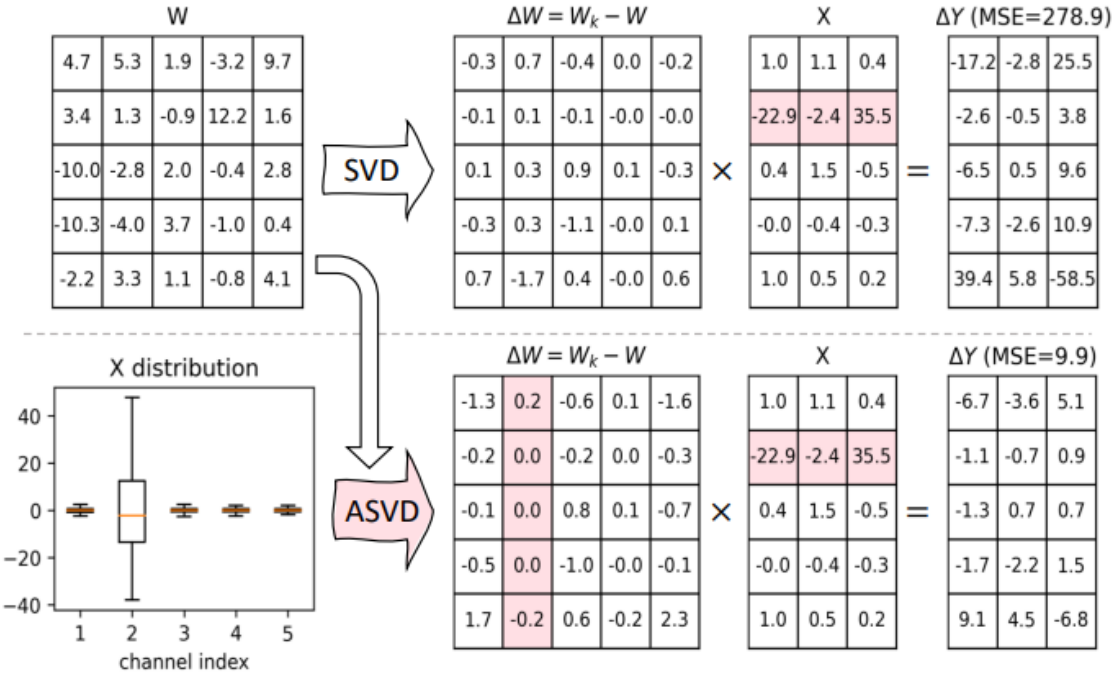


ASVD 개념도

ASVD

- 데이터를 고려한 decomposition 방법 제안
  - 이를 위해 ASVD는 activation-aware singular value decomposition을 제안
    - Scaling을 수행할 수 있는 diagonal matrix S를 W에 곱해준 뒤 SVD 수행
    - Scaling matrix는 X의 각 channel 의 절대 평균값
    - 수식에서 i는 channel, n은 total number of activation

$$\mathbf{S}_{ii} := (\frac{1}{n}\sum_{j=1}^{n}|\mathbf{X}_{ij}|)^{\alpha},$$



ASVD 개념도

ASVD

$$\mathbf{S}_{ii} := (\frac{1}{n}\sum_{j=1}^{n}|\mathbf{X}_{ij}|)^{\alpha},$$

- 데이터를 고려한 decomposition 방법 제안
  - Channel 별 가중치가 고려된 WS로 만들고 decomposition 수행

    $\mathbf{W} = \mathbf{W}\mathbf{S}\mathbf{S}^{-1} = (\mathbf{W}\mathbf{S})\mathbf{S}^{-1}.$

    $\mathbf{W}\mathbf{S} \approx \mathbf{U}'_k\mathbf{\Sigma}'_k{\mathbf{V}'_k}^T.$
  - 이를 통해 decomposition 과정에서 각 activation channel이 weight 에 미치는 영향을 고려하여 SVD 수행
    - 즉, activation을 고려하여 rank 선택이 이루어지도록 함
  - 그리고 다시 S의 역행렬을 곱해주어 원래 weight 공간으로 복원

    $\mathbf{W} = (\mathbf{W}\mathbf{S})\mathbf{S}^{-1} \approx (\mathbf{U}'_k\mathbf{\Sigma}'_k{\mathbf{V}'_k}^T)\mathbf{S}^{-1} = \mathbf{U}'_k\mathbf{\Sigma}'_k{\mathbf{V}''_k}^T = \mathbf{W}_k.$

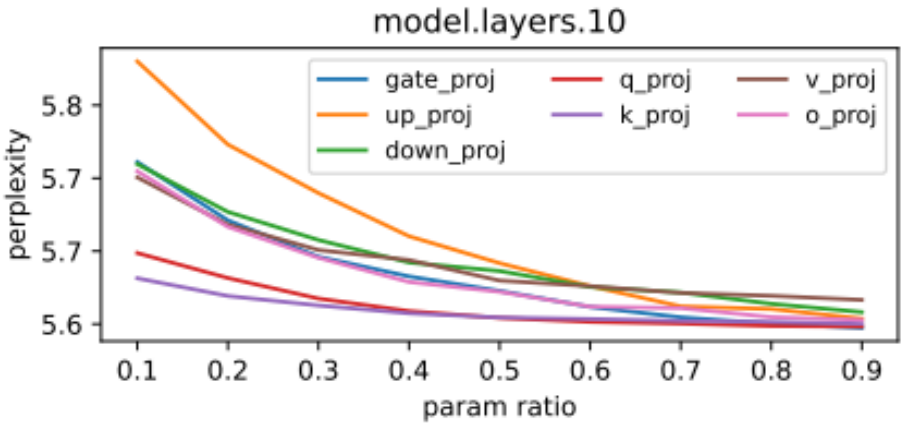
ASVD

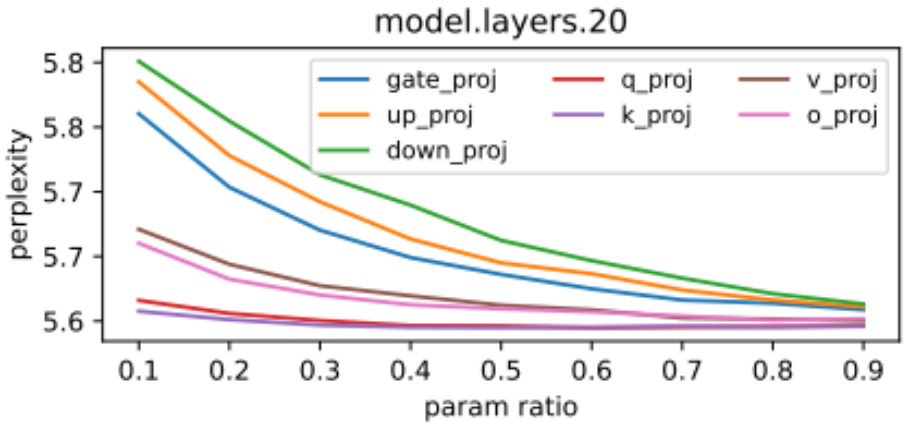
- 실험 결과
  - MMLU는 57개 과목에서 다지선다형 문제에서 정답을 맞춘 비율을 평가
  - WIKI랑 PTB는 언어 모델링 벤치마크로 다음 단어를 얼마나 잘 예측하는지 평가
  - 일반적인 SVD를 사용했을 때와 비교하여 activation aware SVD를 사용했을 때 훨씬 성능 방어가 잘 되는 것을 확인할 수 있음

		LLaMA-7b			LLaMA-2-7b			LLaMA-2-13b		
method	param ratio	MMLU	wiki	ptb	MMLU	wiki	ptb	MMLU	wiki	ptb
original	1	30.76%	5.68	29.63	34.86%	5.47	20.82	40.16%	4.88	29.21
SVD	0.95	22.98%	2800	5458	-	nan	nan	-	nan	nan
SVD*	0.95	23.92%	136.05	183.92	24.78%	46.79	363.37	24.86%	167.63	567.02
SVD*	0.9	23.54%	698.66	262.03	24.31%	114.45	27660	-	nan	nan
ASVD	0.95	30.26%	5.78	32.64	33.24%	5.64	23.98	39.52%	4.94	31.93
ASVD	0.9	29.67%	6.09	37.80	32.58%	5.93	32.63	40.04%	5.12	34.03
ASVD	0.85	29.70%	6.80	52.11	31.57%	6.74	59.84	37.95%	5.54	39.32
ASVD	0.8	27.85%	8.89	88.09	28.15%	8.91	114.70	34.63%	6.53	59.68
ASVD	0.75	24.94%	14.51	212.80	25.97%	18.97	432.57	28.59%	8.71	110.10

ASVD

- 실험 결과
  - 각 내부 모듈의 compression ratio에 따른 성능 변화
  - 주로 Q,K projection 단이 성능 감소에 덜 민감하고, MLP 단이 성능 감소에 큰 영향을 줌





SVD-LLM[1]

- Truncation loss와 truncated singular value 사이에 직접적인 비례관계를 만듦
  - Compression loss : $||WX - W'X||_F$
    - Compression loss는 ASVD와 동일하게 설정
  - 이때 Frobenius norm은 다음과 같음 → $\|A\|_F \triangleq \left(\sum_{j=1}^{n}\sum_{i=1}^{m}|a_{ij}|^2\right)^{\frac{1}{2}} = \left[\mathbf{Trace}\left(A^TA\right)\right]^{\frac{1}{2}}$
  - 따라서 i번째 singular value를 truncate 했을 때 설정한 compression loss를 전개하면 아래와 같음

$$L_i = \|WX - W'X\|_F = \left\|WSS^{-1}X - \mathbf{SVD}(WS)S^{-1}X\right\|_F = \left\|(WS - \mathbf{SVD}(WS))S^{-1}X\right\|_F$$

$$= \left\|\sigma_i u_i v_i^T S^{-1}X\right\|_F = \sigma_i \mathbf{Trace}\left(u_i v_i^T S^{-1}XX^T\left(S^{-1}\right)^T v_i u_i^T\right)^{\frac{1}{2}}$$

  - 여기서 $\mathbf{Trace}\left(u_i v_i^T S^{-1}XX^T\left(S^{-1}\right)^T v_i u_i^T\right)^{\frac{1}{2}}$ term 때문에 singular value와 loss가 직접적인 비례관계를 갖지 못함
  - 이때 $U = [u_1, u_2, u_3, ..., u_r]$ $V = [v_1, v_2, v_3, ..., v_r]$ 는 orthonormal matrix 이므로

$$v_i^T v_i = u_i^T u_i = 1; v_i^T v_j = u_i^T u_j = 0, \forall i \neq j; \mathbf{Trace}(v_i v_i^T) = \mathbf{Trace}(u_i u_i^T) = 1$$

  - 따라서 $S^{-1}X$를 orthonormal로 만들면 loss와 truncate한 singular value가 직접적인 비례관계를 갖게 됨
    - 다른 말로 하면 $S^{-1}X$의 공분산 구조 때문에 영향을 받을 수 있음

SVD-LLM

- 데이터를 고려한 decomposition 방법 제안
  - $S^{-1}X$의 공분산 $(S^{-1}XX^T(S^{-1})^T)$을 I로 만들면 가장 낮은 singular value를 truncate 했을 때 가장 낮은 loss로 작용함
  - Cholesky decomposition 방법 사용
    - Cholesky decomposition 을 사용해서 activation의 공분산 행렬($C = XX^T$)을 decompose 함
      - Cholesky decomposition은 Symmetric positive-definite 행렬에 대해서 수행 가능한데 일단 activation의 공분산 행렬이 이에 해당된다고 가정하고 접근
      - 완벽한 가정은 아니기 때문에 이를 해결한 방법은 SVD-LLM v2[1] 참고
    - Cholesky decomposition 은 Cholesky(C) = SS^T의 형태로 수행됨
    - 이 때, 아래의 증명에 의해 Cholesky decomposition을 활용하면 $S^{-1}XX^T(S^{-1})^T$을 I로 만들 수 있음

$$S^{-1}XX^\top S^{-\top} = S^{-1}(SS^\top)S^{-\top} = (S^{-1}S)(S^\top S^{-\top}) = I \cdot I = I.$$

    - 따라서 loss와 truncate한 singular value가 직접적인 비례관계를 갖도록 rank truncation 수행
    - Decomposition 과정은 ASVD와 마찬가지로 아래의 과정으로 수행

$$\mathbf{W} = \mathbf{W}\mathbf{S}\mathbf{S}^{-1} = (\mathbf{W}\mathbf{S})\mathbf{S}^{-1}.$$

$$\mathbf{W}\mathbf{S} \approx \mathbf{U}'_k\mathbf{\Sigma}'_k{\mathbf{V}'_k}^T.$$

$$\mathbf{W} = (\mathbf{W}\mathbf{S})\mathbf{S}^{-1} \approx (\mathbf{U}'_k\mathbf{\Sigma}'_k{\mathbf{V}'_k}^T)\mathbf{S}^{-1} = \mathbf{U}'_k\mathbf{\Sigma}'_k\mathbf{V}''^T_k = \mathbf{W}_k.$$

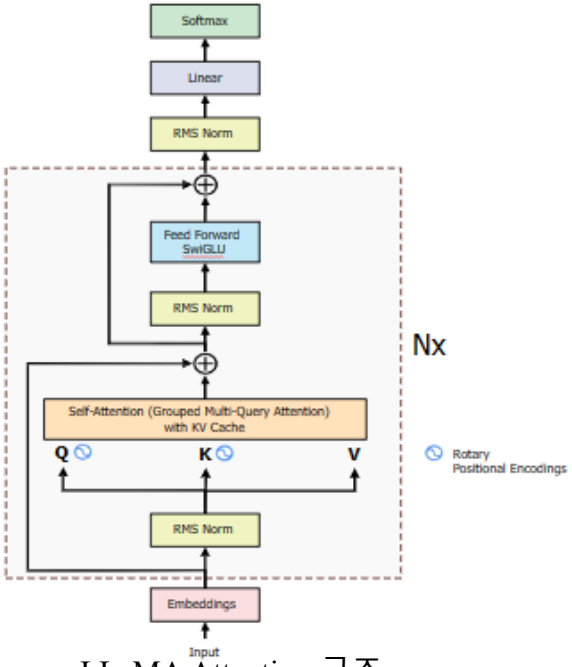
SVD-LLM

- 실험 결과 ASVD 보다 좋은 성능을 보이며, 특히 reduction ratio가 늘어났을 때도 안정적인 값을 보임
  - SVD-LLM(w) 의 값으로 비교하면 되고, SVD-LLM은 fine-tuning이 들어간 결과

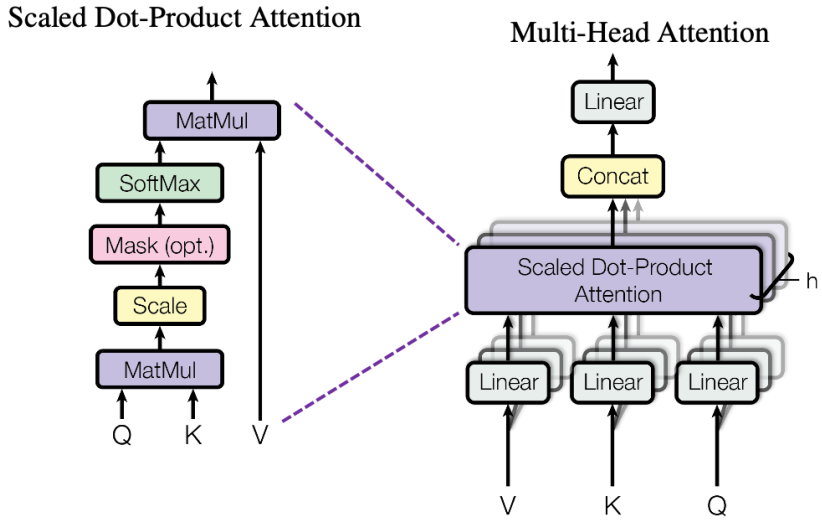
Ratio (Mem.)	Method	WikiText-2↓	C4↓	Openb.	ARC_e	WinoG.	HellaS.	PIQA	MathQA	**Average↑**	TruthfulQA↑	GSM8K↑
0% (13.5 GB)	Original	5.68	7.34	0.34	0.75	0.70	0.57	0.79	0.27	0.57	0.30	0.09
20% (10.2 GB)	SVD	20061	18800	0.05	0.04	0.01	0.03	0.02	0.03	0.03	0.00	0.00
	FWSVD	1727	1511	0.09	0.11	0.05	0.08	0.10	0.05	0.08	0.00	0.00
	ASVD	11.14	15.93	0.29	0.53	0.64	0.41	0.68	0.17	0.45	0.21	0.04
	SVD-LLM (W)	7.94 (↓29%)	15.84 (↓1%)	0.31	0.62	0.61	0.45	0.71	0.21	0.49 (↑9%)	0.26 (+0.05)	0.05 (+0.01)
	SVD-LLM	**7.73** (↓31%)	**12.23** (↓23%)	**0.33**	**0.67**	**0.69**	**0.55**	**0.79**	**0.26**	**0.55** (↑22%)	**0.28** (+0.07)	**0.08** (+0.04)
40% (7.76 GB)	SVD	52489	47774	0.04	0.04	0.05	0.01	0.03	0.02	0.03	0.00	0.00
	FWSVD	18156	12847	0.06	0.05	0.02	0.00	0.05	0.03	0.04	0.00	0.00
	ASVD	1407	1109	0.08	0.11	0.09	0.08	0.13	0.08	0.10	0.01	0.00
	SVD-LLM (W)	13.73 (↓99%)	75.42 (↓93%)	0.25	0.33	0.55	0.40	0.63	0.12	0.38 (↑280%)	0.17 (+0.17)	0.02 (+0.02)
	SVD-LLM	**9.27** (↓99%)	**15.63** (↓99%)	**0.29**	**0.59**	**0.68**	**0.52**	**0.69**	**0.20**	**0.50** (↑400%)	**0.24** (+0.23)	**0.07** (+0.07)
60% (5.35 GB)	SVD	105474	106976	0.01	0.03	0.01	0.00	0.01	0.02	0.01	0.00	0.00
	FWSVD	32194	29292	0.06	0.02	0.01	0.01	0.02	0.03	0.03	0.00	0.00
	ASVD	57057	43036	0.05	0.04	0.06	0.09	0.08	0.05	0.06	0.00	0.00
	SVD-LLM (W)	66.62 (↓99%)	471.83 (↓99%)	0.10	0.05	0.17	0.10	0.21	0.04	0.11 (↑83%)	0.01 (+0.01)	0.00 (+0.00)
	SVD-LLM	**15.00** (↓99%)	**26.26** (↓99%)	**0.18**	**0.42**	**0.44**	**0.31**	**0.35**	**0.12**	**0.30** (↑400%)	**0.14** (+0.14)	**0.04** (+0.04)
80% (2.58 GB)	SVD	687291	708243	0.00	0.01	0.02	0.01	0.01	0.00	0.01	0.00	0.00
	FWSVD	96872	89243	0.01	0.02	0.00	0.01	0.01	0.00	0.01	0.00	0.00
	ASVD	80425	67927	0.04	0.03	0.03	0.02	0.01	0.01	0.03	0.00	0.00
	SVD-LLM (W)	1349 (↓98%)	6224 (↓91%)	0.07	0.03	0.04	0.02	0.07	0.01	0.04 (↑33%)	0.00 (+0.00)	0.00 (+0.00)
	SVD-LLM	**31.79** (↓99%)	**43.71** (↓99%)	**0.11**	**0.23**	**0.21**	**0.14**	**0.17**	**0.08**	**0.16** (↑433%)	**0.04** (+0.04)	**0.02** (+0.02)

COMCAT[1]

- LLaMA와 Vision Transformer(ViT)의 차이점
  - LLaMA에서 사용하는 attention 와 다르게 Rotary Position Embedding을 query, key에 적용하고 있음
    - Rotary position embedding은 입력되는 sequence 길이에 따라서 다른 값이 더해짐
  - ViT는 query, key, value weight 간 pre-compute가 가능한 구조



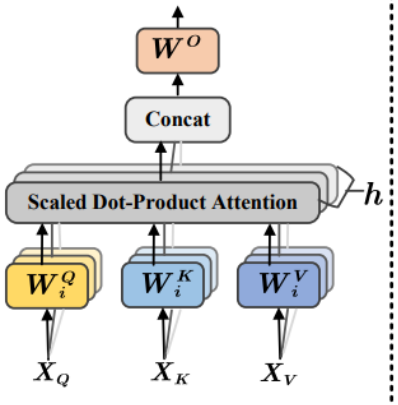
LLaMA Attention 구조



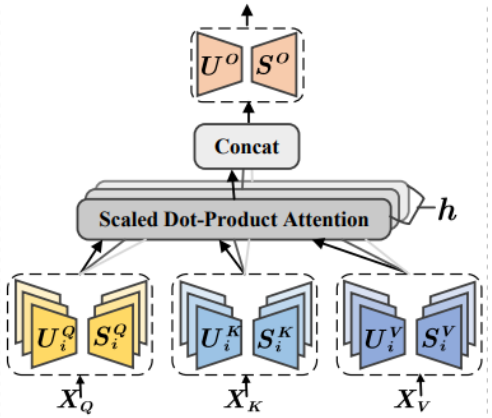
Vision Transformer Attention 구조

COMCAT[1]

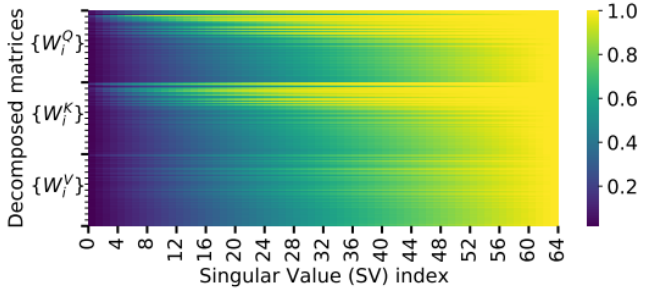
- Vision transformer 에서 naïve weight decomposition을 하면 아래 그림에서와 같이 비효율적인 구조를 가지게 됨
  - CxC weight를 low rank r로 approximation 했을 때, r이 $\frac{C}{2}$ 이하가 돼야 그때부터 이득이 있는 문제가 있음
  - 특정 head에서 Q,K,V,O weight가 low-rank approximation에 따른 민감도가 다름
- ViT에서 query, key, value weight 간 pre-compute가 가능한 구조라면?
  - Query-key, value-out weight 간 연산을 미리 해둘 수 있음



기존 weight



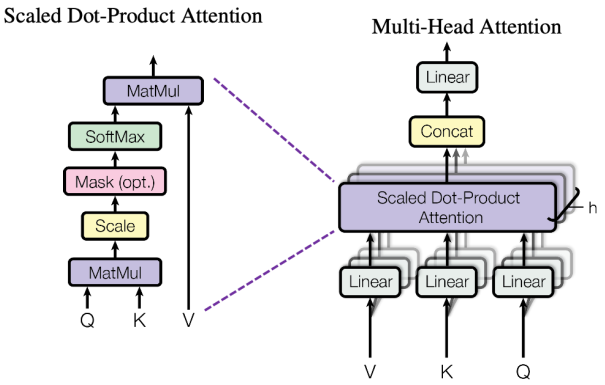
Naïve weight decomposition



Head-wise로 singular value에 따른 민감도

COMCAT

- ViT에서 Multi-Head Attention(MHA)의 더 깊은 이해
  - 기본적으로 $\text{attention}(X) = Softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$
  - 풀어서 쓰면 $\text{attention}(X) = Softmax\left(\frac{XW^Q(XW^K)^T}{\sqrt{d_k}}\right)XW^V = Softmax\left(\frac{X\,W^QW^{K^T}\,X^T}{\sqrt{d_k}}\right)XW^V$
  - Head까지 고려하면 $head_i = Softmax\left(\frac{X\,W_i^QW_i^{K^T}\,X^T}{\sqrt{d_k}}\right)XW_i^V$
    - $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{C\times D_h}$, $D_h = \frac{C}{H}$ (C: embedded dimension, H : number of head)
  - 따라서 $\text{MHA}(X) = Concat(head_i)W^O$
    - $W^O \in \mathbb{R}^{C\times C}$

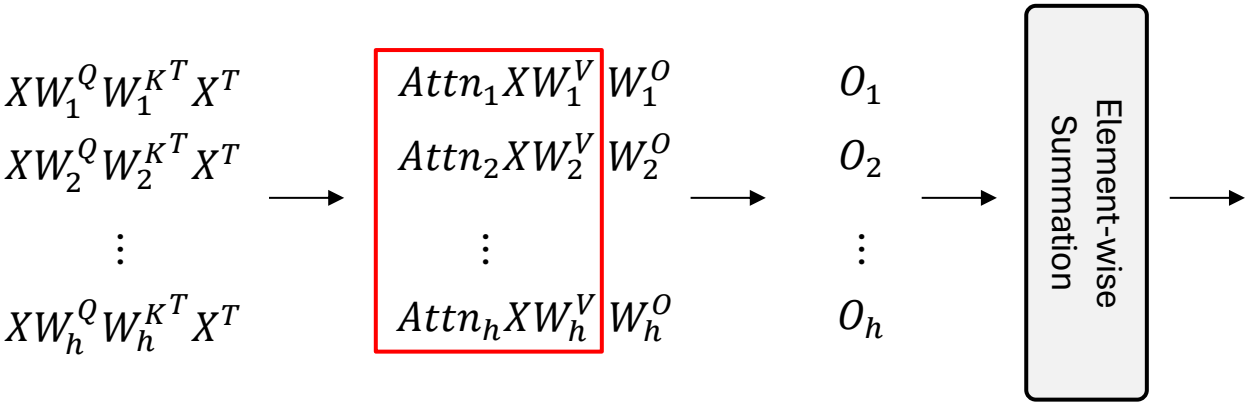


Vision Transformer Attention 구조

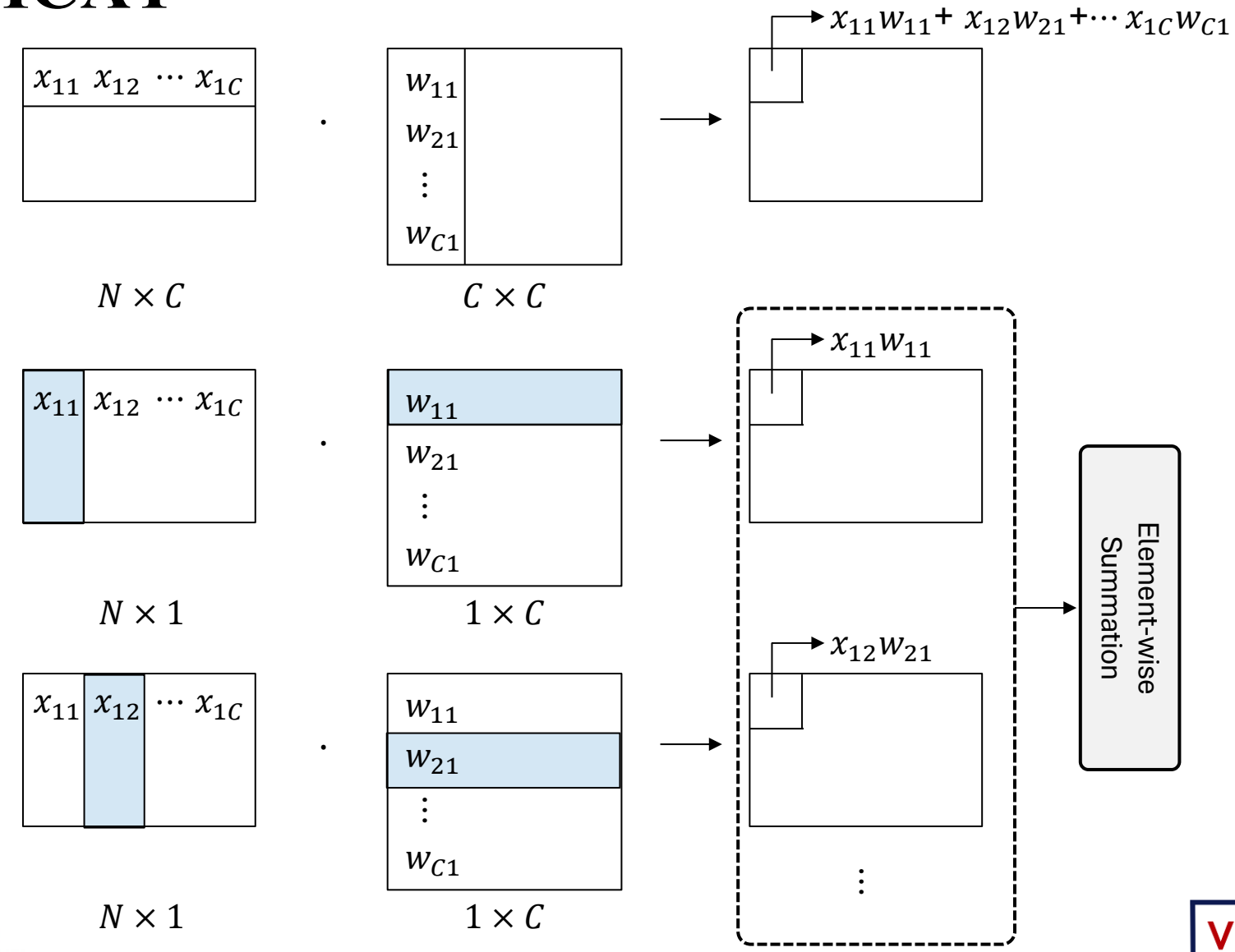
COMCAT

- ViT에서 Multi-Head Attention(MHA)의 더 깊은 이해
  - $\mathrm{MHA}(X) = Concat(head_i)W^O\ ,\ head_i = Softmax\left(\frac{X\,W_i^Q W_i^{K^T}\,X^T}{\sqrt{d_k}}\right)XW_i^V$
    - $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{C\times D_h}$, $W^O \in \mathbb{R}^{C\times C}$, $D_h = \frac{C}{H}$ (C: embedded dimension, H : number of head)
  - 위의 두 단계에 거친 수식이랑 아래 수식이랑 같은 연산
    - Attention map이 head wise로 적용돼서 W^V랑 W^O랑 pre-compute 하려면 W^V, W^O 도 head-wise로 나눠서 처리 해야함 $W_i^O \in \mathbb{R}^{D_h\times C}$

$$\mathrm{MHA}(X) = \sum_{i=1}^{h} \mathrm{head_i} W_i^O = \sum_{i=1}^{h} \mathrm{Softmax}\left(\frac{X(W_i^Q W_i^{K\top})X^\top}{\sqrt{d_h}}\right) X(W_i^V W_i^O)$$



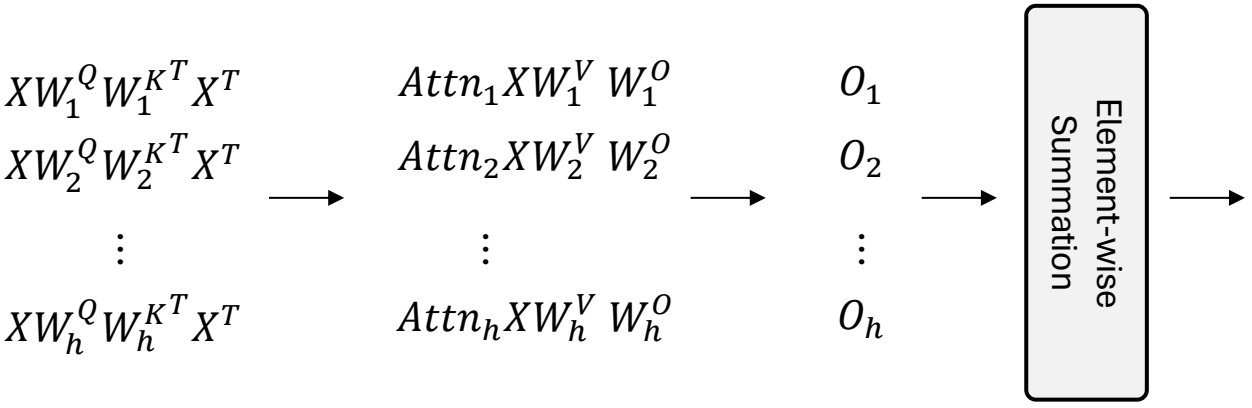
COMCAT


x_{11} x_{12} $\cdots$ x_{1C}
w_{11}
w_{21}
w_{C1}
$x_{11}w_{11}+ x_{12}w_{21}+\cdots x_{1C}w_{C1}$
$N \times C$
$C \times C$
x_{11} x_{12} $\cdots$ x_{1C}
w_{11}
w_{21}
w_{C1}
$x_{11}w_{11}$
$N \times 1$
$1 \times C$
x_{11} x_{12} $\cdots$ x_{1C}
w_{11}
w_{21}
w_{C1}
$x_{12}w_{21}$
$N \times 1$
$1 \times C$
Element-wise Summation

COMCAT

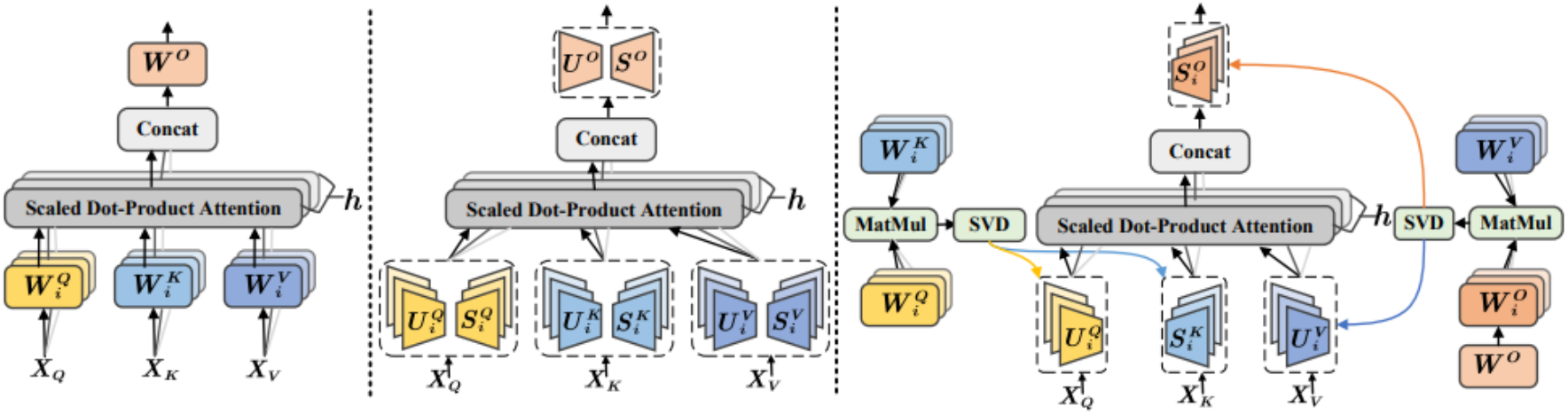
- ViT에서 Multi-Head Attention(MHA)의 더 깊은 이해
  - $\mathrm{MHA}(X) = Concat(head_i)W^O\ ,\ head_i = Softmax\left(\frac{X\,W_i^Q W_i^{K^T}\,X^T}{\sqrt{d_k}}\right)XW_i^V$
    - $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{C\times D_h}$, $W^O \in \mathbb{R}^{C\times C}$, $D_h = \frac{C}{H}$ (C: embedded dimension, H : number of head)
  - 위의 두 단계에 거친 수식이랑 아래 수식이랑 같은 연산
    - Attention map이 head wise로 적용돼서 W^V랑 W^O랑 pre-compute 하려면 W^O 도 head-wise로 나눠서 처리 해야함 $W_i^O \in \mathbb{R}^{D_h\times C}$

$$\mathrm{MHA}(X) = \sum_{i=1}^{h} \mathrm{head_i} W_i^O = \sum_{i=1}^{h} \mathrm{Softmax}\left(\frac{X(W_i^Q W_i^{K^\top})X^\top}{\sqrt{d_h}}\right) X(W_i^V W_i^O)$$



COMCAT

- $W_i^{QK} = W_i^Q W_i^{K^T}, W_i^{VO} = W_i^V W_i^O$ 로 미리 연산을 진행한 후 SVD 진행
- $W_i^{QK} \approx U_i^{QK}(\Sigma_i^{QK} V_i^{QK^T}) = U_i^Q S_i^K, \quad W_i^{VO} \approx U_i^{VO}(\Sigma_i^{VO} V_i^{VO^T}) = U_i^V S_i^O$
  - Q-K 및 V-O에서 한쪽이 low-rank의 민감도가 높다고 하더라도, matrix multiplication 과정을 통해 이를 완화할 수 있음
  - 이와 같이 더 효과적인 low-rank approximation 방법과 더불어 naïve approach와 비교했을 때 parameter를 2배 가까이 줄일 수 있음



기존 weight　　Naïve weight decomposition　　COMCAT

COMCAT

- 실험 결과

# of Params. in MHA ↓		20%	40%	60%	80%
Top-1(%)	**Head-Level**	**79.81**	**76.11**	**63.56**	**11.5**
	Matrix-Level	73.01	56.15	22.95	0.73

Naïve decomposition(Matrix-Level) 과
제안하는 COMCAT(Head-Level) 비교
(Original Acc. : 80.90)

Method	Compression	Top-1	FLOPs (↓%)	Params (↓%)
DeiT-small	Baseline	79.8	-	-
COMCAT (Ours)	Low-rank	79.27	**51.21**	**49.98**
COMCAT (Ours)	Low-rank	79.58	44.93	43.82
COMCAT (Ours)	Low-rank	**79.92**	41.15	40.11
UPop (Shi et al., 2023)	Pruning	79.6	39	39
UVC (Yu et al., 2022b)	Pruning	78.82	49.59	-
SCOP (Tang et al., 2020)	Pruning	77.5	43.6	-
S^2ViTE (Chen et al., 2021b)	Sparse	79.22	31.63	33.94
ToMe (Bolya et al., 2022)	Token	79.4	41.30	0
PS-ViT (Tang et al., 2022)	Token	79.4	43.5	0
HVT (Pan et al., 2021b)	Token	78.0	47.8	0
PoWER (Goyal et al., 2020)	Token	78.3	41.3	0
DeiT-base	Baseline	81.8	-	-
COMCAT (Ours)	Low-rank	**82.26**	**61.68**	**61.06**
CT-GFM (Yu & Wu, 2023)	Low-rank	81.28	-	40
MD-ViT (Hou & Kung, 2022)	Pruning	81.5	60	-
UVC (Yu et al., 2022b)	Pruning	80.57	54.5	-
VTP (Zhu et al., 2021)	Pruning	80.7	43.2	44.44
S^2ViTE (Chen et al., 2021b)	Sparse	82.22	33.13	34.41
PS-ViT (Tang et al., 2022)	Token	81.5	44.3	0
IA-RED^2 (Pan et al., 2021a)	Token	80.3	32.96	0

Weight decomposition과 다른 경량방법들과 비교

Conclusion

- Weight decomposition에서 loss를 설정하는 방법
- Attention 구조에 특화된 weight decomposition 방법