Recent works on Knowledge Editing

전용찬

Natural Language Processing
& Artificial Intelligence

Knowledge Editing Overview

- Knowledge Editing 이란?
  - 모델이 가지고 있는 지식을 수정하는 것
  - 즉, (Subject, Relation)가 주어졌을 때 원하는 Object로 답할 수 있도록 만드는 것
  - 예시: 미국의 현재 대통령은? ~~조 바이든~~ 트럼프

    subject relation object

- 최근 Knowledge Editing의 트렌드:

1. Lifelong editing: 지속적으로 모델의 지식을 효과적으로 수정하는 것
   1. AlphaEdit: Null-Space Constrained Knowledge Editing for Language Models
   2. WilKE: Wise-Layer Knowledge Editor for Lifelong Knowledge Editing
   3. Aging with GRACE: Lifelong Model Editing with Discrete Key-Value Adaptors
   4. WISE: Rethinking the Knowledge Memory for Lifelong Model Editing of Large Language Models
   5. Knowledge in Superposition: Unveiling the Failures of Lifelong Knowledge Editing for Large Language Models

2. Multi-token Edit: Object가 여러 개의 토큰으로 이루어져 있을 때의 지식 수정
   1. UnKE: Unstructured Knowledge Editing in Large Language Models
   2. AnyEdit: Edit Any Knowledge Encoded in Language Models

3. Batch editing: 여러개의 지식을 한꺼번에 업데이트 하는 것
   1. MEMIT: MASS-EDITING MEMORY IN A TRANSFORMER
   2. EMMET: A Unified Framework for Model Editing

Knowledge in Superposition: Unveiling the Failures of Lifelong Knowledge Editing for Large Language Models

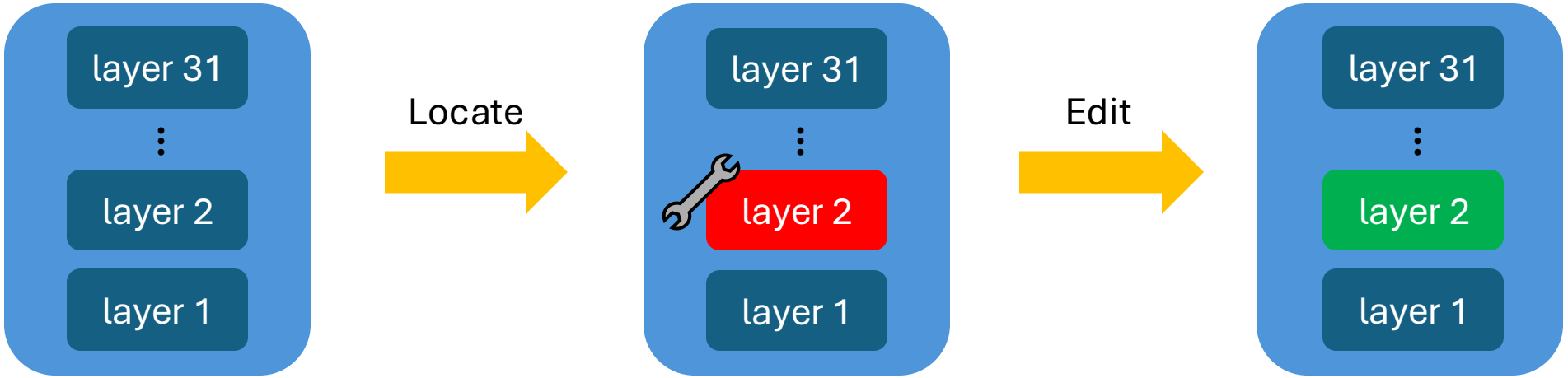
AAAI 2025 (Oral)

Natural Language Processing
& Artificial Intelligence

Introduction

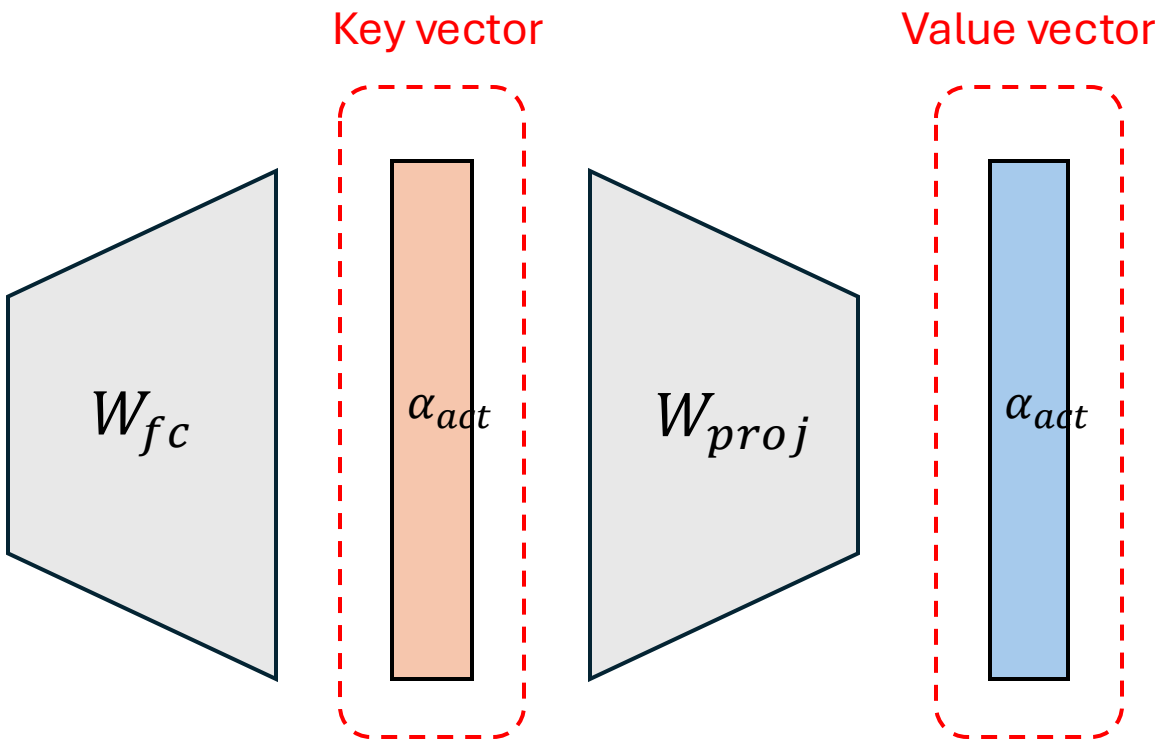
- Lifelong editing이 실패하는 이유: 수정하고 싶은 지식과 관계 없는 지식이 수정되는 현상이 발생함
- 이러한 현상을 superposition 관점에서 접근하여 분석함
- ROME, MEMIT과 같은 locate-and-edit 방법을 기반으로 함
  - locate-and-edit 방법: 지식을 가지고 있는 MLP layer을 찾은 후에 해당 MLP layer만을 수정하는 방법

<Locate-and-edit method>



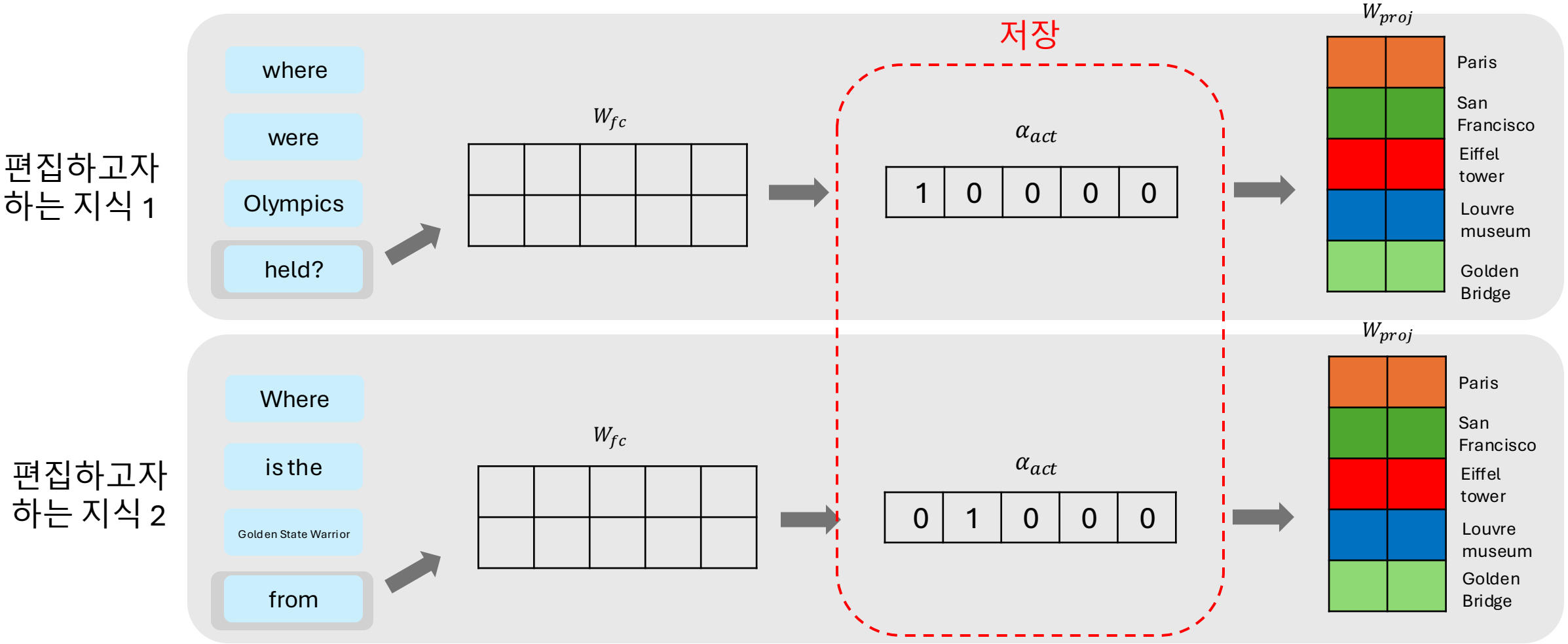
Locate-and-edit based Knowledge Editing

1. LLM의 MLP layer가 지식을 포함하고 있다는 Linear associative memory 이론을 기반으로 하는 방법
2. MLP layer의 projection up layer가 지식을 활성화시키는 Key vector를 만들고 projection down layer는 Key vector를 바탕으로 지식이 포함된 Value vector를 만듬



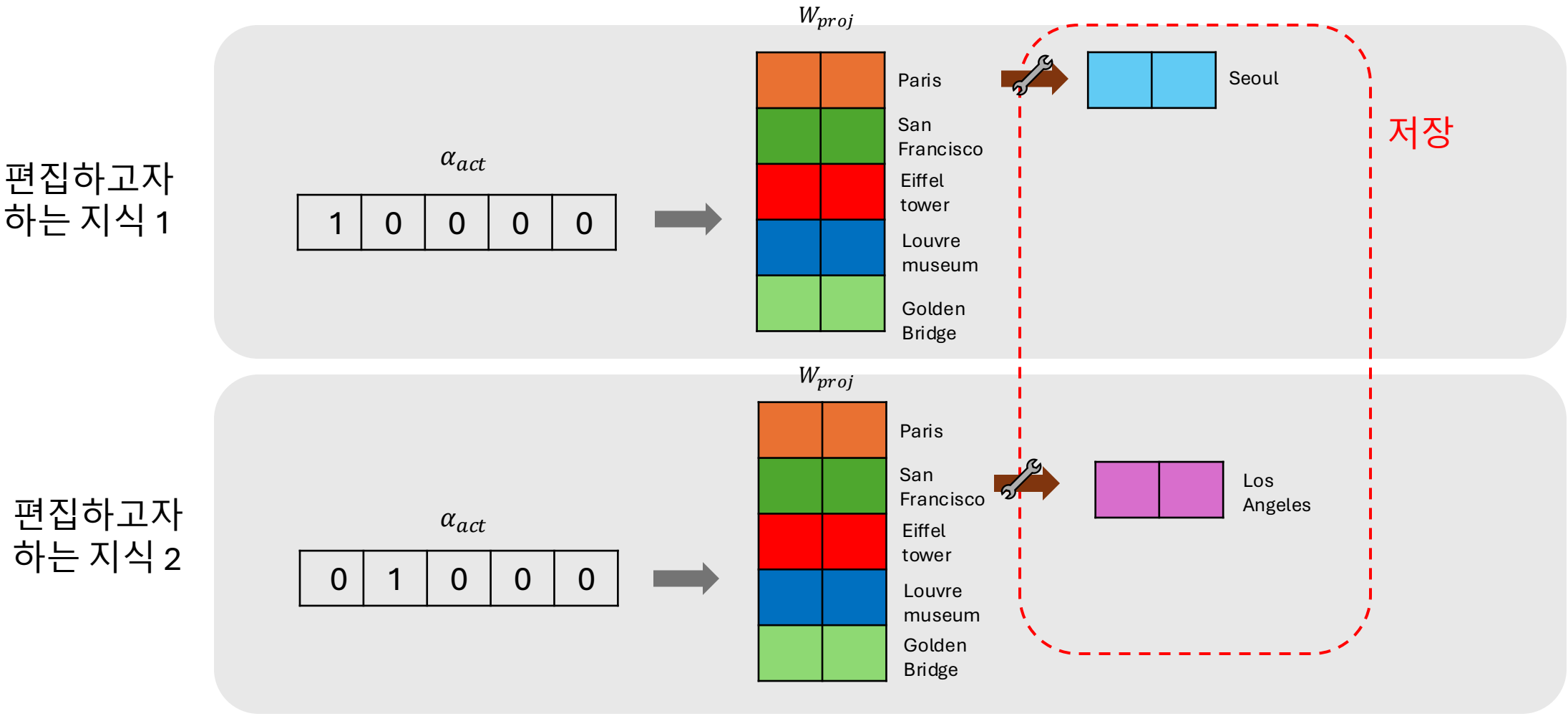
Locate-and-edit based Knowledge Editing

1. 지식을 가지고 있는 MLP layer를 찾음. 주로 causal tracing 방법 사용함
2. 찾은 MLP layer를 대상으로 지식을 활성화 시키는 Key vector를 계산하고 저장함



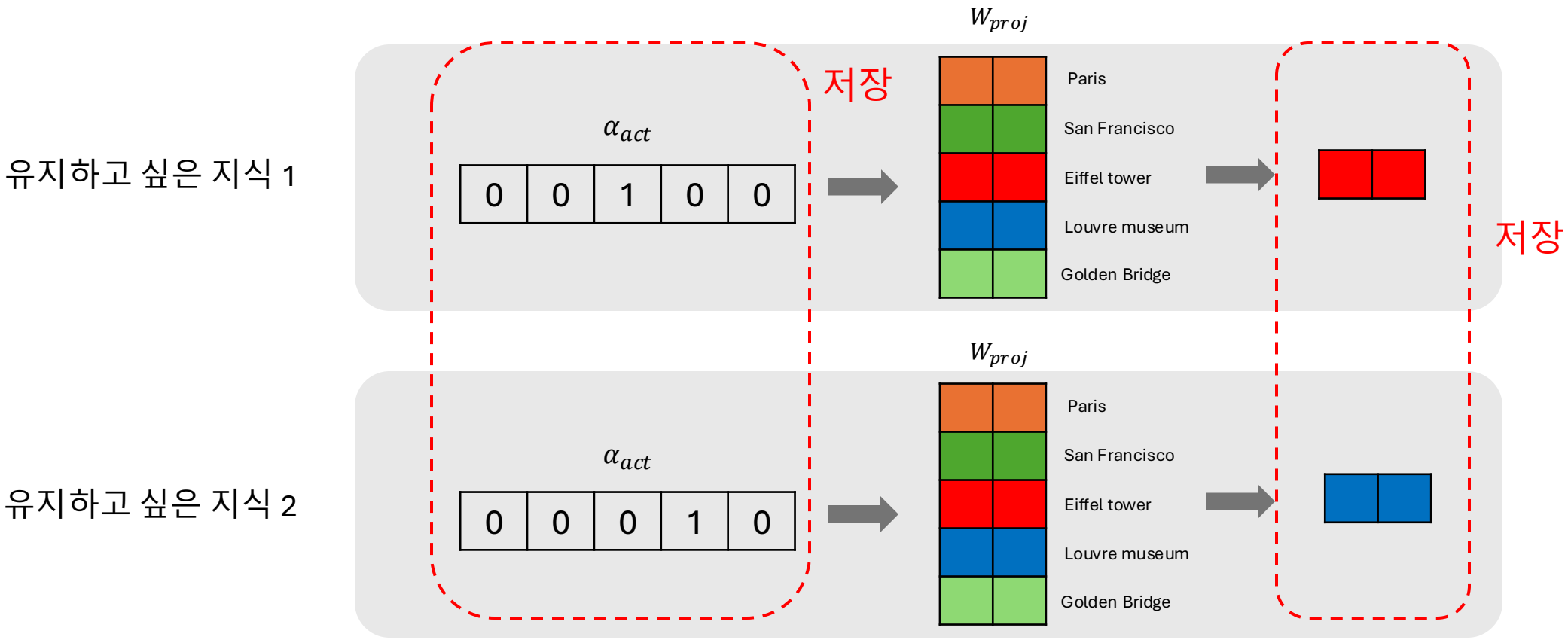
Locate-and-edit based Knowledge Editing

3. 편집하고 싶은 지식의 value vector를 계산하고 저장함. Value vector는 gradient descent를 통해 계산함



Locate-and-edit based Knowledge Editing

4. 유지하고 싶은 지식에 대한 key vector와 value vector 값들을 얻음



Locate-and-edit based Knowledge Editing

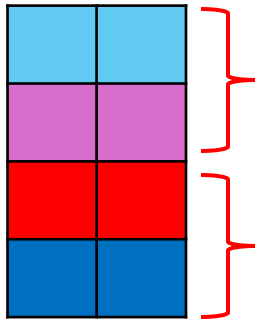
5. 편집하고자 하는 지식과 유지하고자 하는 지식을 이어 붙여서 matrix로 만듦

수정하고자 하는 지식

유지하고자 하는 지식

1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0

K



수정하고자 하는 지식

유지하고자 하는 지식

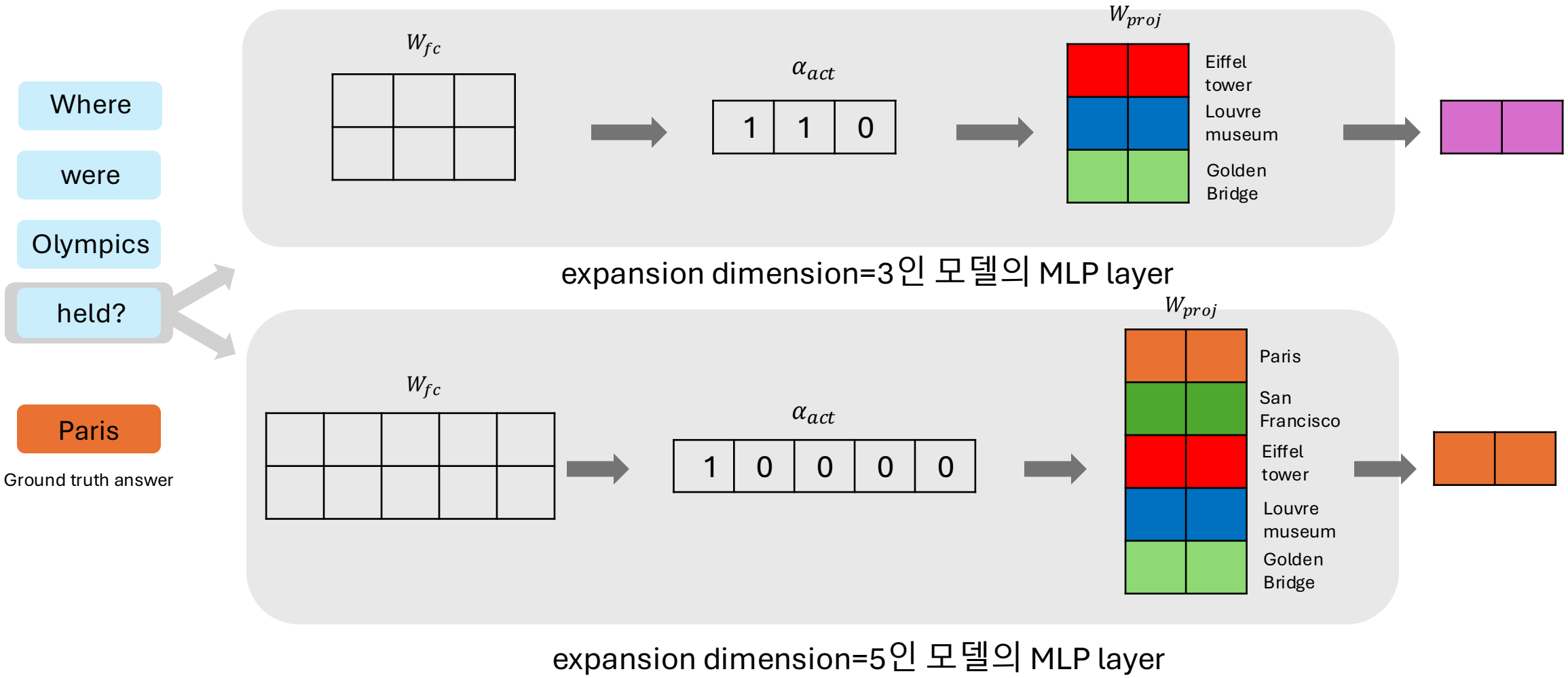
V

6. V에 K의 pseudoinverse를 곱해줌으로써 W_{proj}를 업데이트 함

$$\hat{W}_{proj} = VK^{+}$$

Superposition theory

- 한정된 차원안에서 많은 수의 feature를 표현하고자 할 때 feature가 서로 중첩되어 나타나는 것



Knowledge in superposition

- 만약 지식들이 독립적으로 나타난다면 지식을 활성화 시키는 key vector들이 서로 독립적이여야 함.
  즉, key matrix가 $K \cdot K^T = I$ 을 만족해야 함

지식이 중첩되지 않을 때:

1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0

•

1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1
0	0	0	0

=

1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

지식이 중첩될 때:

1	1	0
0	1	1
1	0	1
0	1	1

•

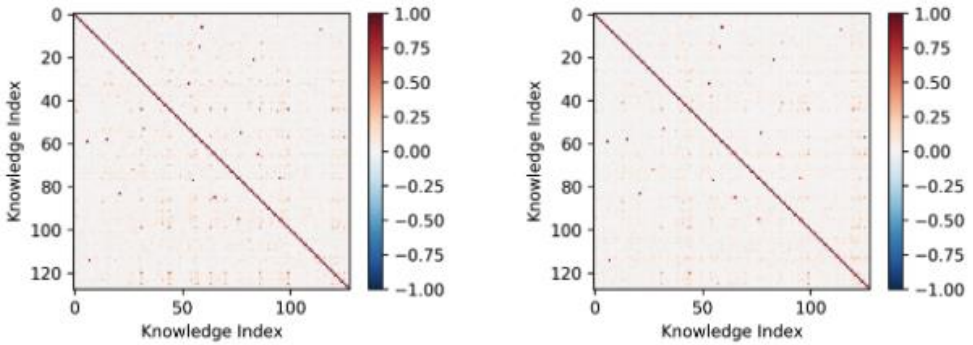
1	0	1	0
1	1	0	1
0	1	1	1

=

2	1	1	1
1	2	1	2
1	1	2	1
1	2	1	2

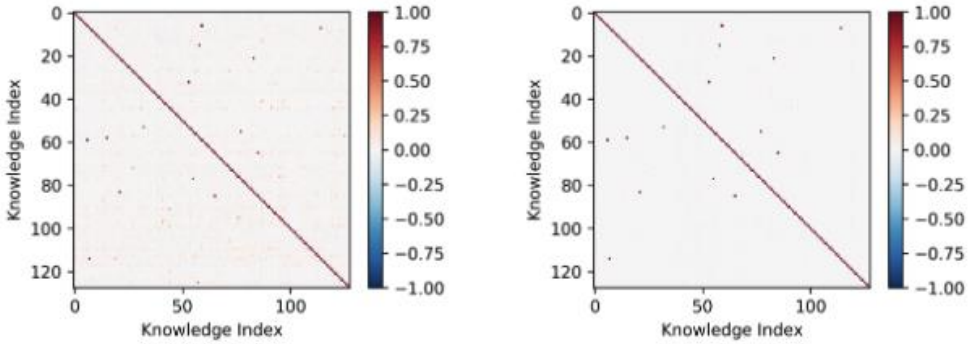
Knowledge in superposition

- GPT2와 GPT-J에서 $K \cdot K^T$ 를 통해 지식이 중첩되는 현상을 관측함.
- 모델이 커질수록 지식이 중첩되는 현상이 적어짐



(a) GPT2-Small

(b) GPT2-Medium

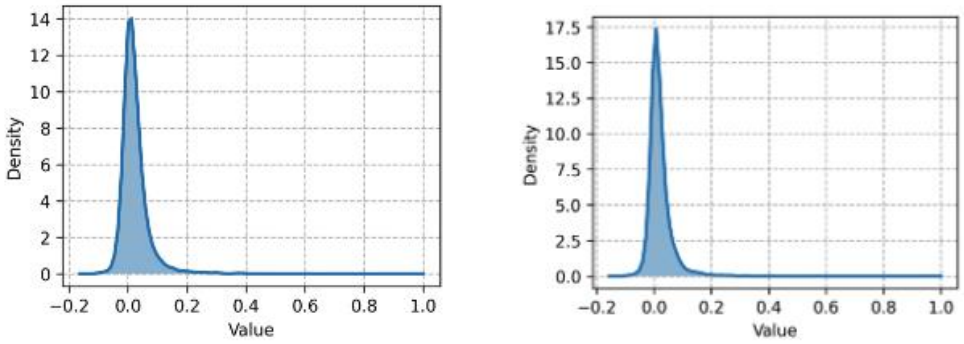


(c) GPT2-Large

(d) GPT-J-6B

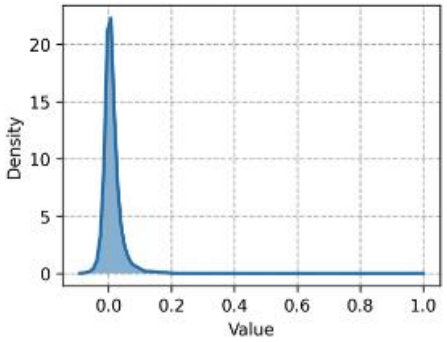
Knowledge in superposition

- 첫번째 layer에서 $K \cdot K^T$의 원소의 density를 확인함
- 모델이 커질수록 0을 중심으로 높은 첨도를 보임. 즉, 모델이 커질수록 지식 중첩 현상이 적어짐.

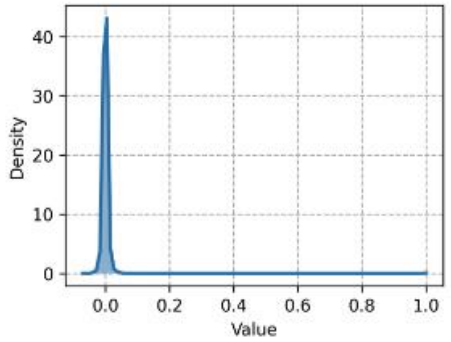


(a) GPT2-Small

(b) GPT2-Medium



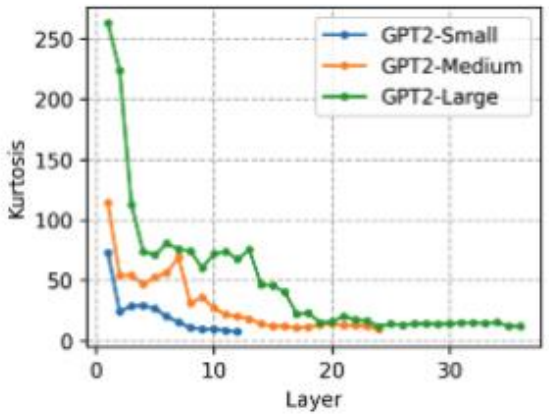
(c) GPT2-Large



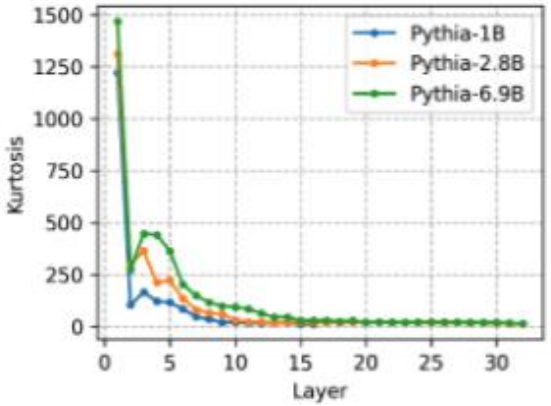
(d) GPT-J-6B

Knowledge in superposition

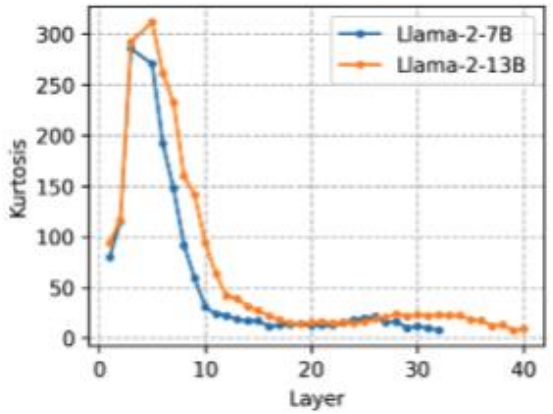
- Layer가 높아질수록 첨도가 낮아짐. 즉, 지식 중첩 현상이 더 증가함
- GPT2 뿐만이 아닌 Pythia, Llama2도 비슷한 현상이 발생함



(a) GPT2 Family



(b) Pythia Family



(c) Llama2 Family

Conclusion

- Lifelong editing이 실패하는 이유를 superposition 관점에서 분석함.
- superposition은 여러 family의 모델들에서 공통적으로 나타났으며, 모델이 작을수록, layer가 높아질수록 superposition 현상이 더 심하다는 것을 확인함
- Lifelong editing을 위해서는 superposition 문제를 해결해야 할 필요성을 제시함

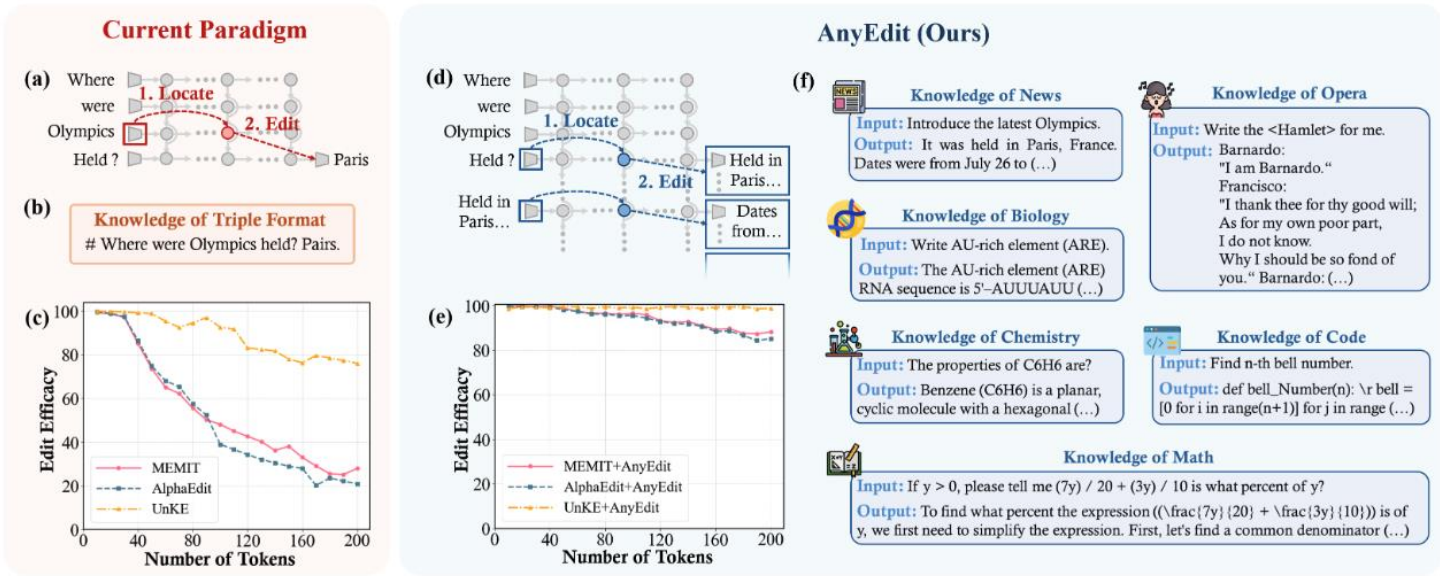
AnyEdit: Edit Any Knowledge Encoded in Language Models

Arxiv

2025.02

Introduction

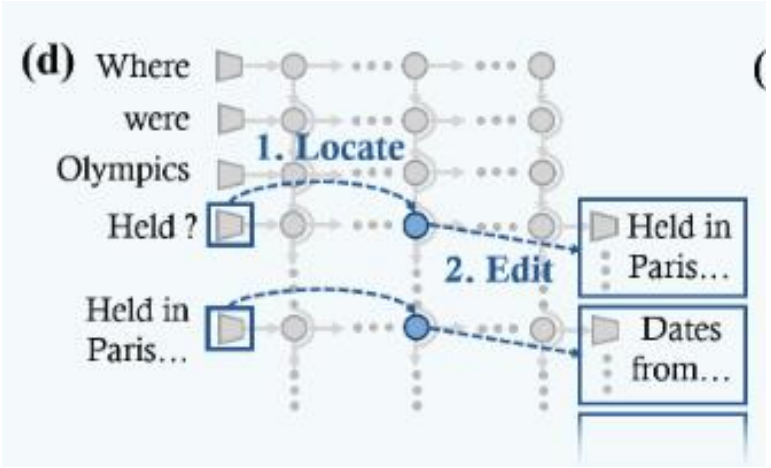
- 기존 연구들은 single token으로 이루어진 object만을 수정하였다는 한계가 있음
- 해당 논문에는 object가 여러개의 토큰으로 이루어져있을 때 효과적으로 수정할 수 있는 방법을 제시함
- 예시: Where were Olympics held?
  - 기존 연구: Paris
  - 해당 논문: It was held in paris, France, Dates were from July 26 to ...
- ROME, MEMIT과 같은 locate-and-edit knowledge editing 방법을 기반으로 함



Methodology

X := (subject, relation), Y := (object)

1. Y를 K개의 chunk들로 쪼갬. 이 때, chunk는 고정된 window size로 잘라서 만듬.
2. 각각의 chunk의 마지막 토큰들만 활용해서 ROME의 방법으로 지식을 편집함



Where were Olympics held? It was held in paris, Dates were from July 26 to … held successfully.

X Y_1 Y_2 Y_K

Experiment

- Baseline Methods:
  1. FT-L: X의 마지막 토큰으로만 특정 layer를 파인튜닝하는 것
  2. UnKE: X의 마지막 토큰으로만 MEMIT 방식으로 모든 layer를 지식 편집 하는 것
  3. MEND
  4. ROME
  5. MEMIT
  6. AlphaEdit

- Datasets
  1. EditEverything: multiple-token multi-domain edit
  2. UnKEBench: multiple-token edit
  3. CounterFact: single-token edit
  4. MQUAKE: single-token multi-hop edit

- Evaluation Metric
  1. Bert Score
  2. Rouge-L

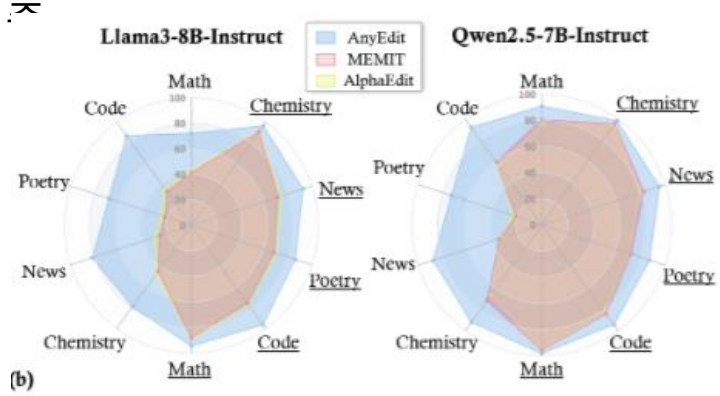
Experiment

- AnyEdit은 특정 layer만 수정한 것, AnyEdit* 은 모든 layer를 수정한 것
- 모든 데이터와 지표에서 AnyEdit이 뛰어난 성능을 보임

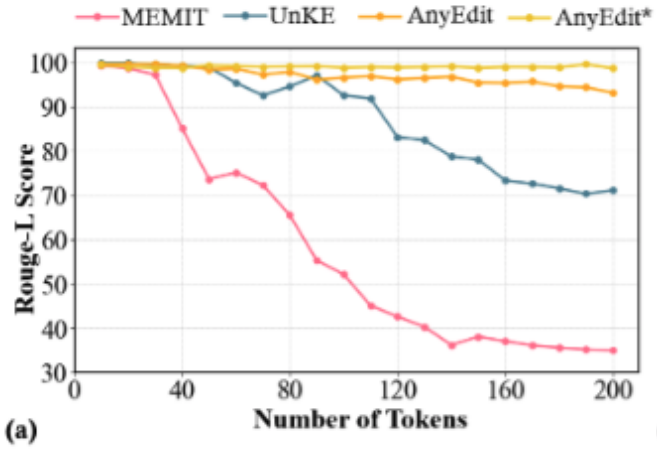
LLM	Method	UnKEBench				AKEW (Counterfact)				AKEW (MQUAKE)	
		Ori.		Para.		Ori.		Para.		Ori.	
		Bert Score	Rouge-L	Bert Score	Rouge-L	Bert Score	Rouge-L	Bert Score	Rouge-L	Bert Score	Rouge-L
Llama3-8B-It	Pre-edited	63.18±0.38	23.67±0.52	62.73±0.31	23.52±0.51	64.03±0.32	15.74±0.42	40.20±0.46	5.52±0.10	65.77±0.37	16.25±0.47
	FT-L	40.31±0.45	11.39±0.56	37.29±0.41	8.51±0.51	42.89±0.43	13.12±0.58	31.44±0.49	5.24±0.08	45.87±0.43	12.99±0.52
	MEND	68.73±0.34	29.24±0.52	64.11±0.32	28.05±0.54	68.81±0.31	30.30±0.48	41.56±0.40	10.95±0.57	67.85±0.33	22.48±0.56
	ROME	72.16±0.31	23.74±0.46	70.54±0.25	22.39±0.54	72.90±0.36	25.86±0.52	43.59±0.51	12.37±0.55	70.10±0.43	21.07±0.59
	MEMIT	76.21±0.36	30.49±0.52	74.25±0.31	28.65±0.61	76.44±0.33	32.20±0.48	47.80±0.34	16.09±0.59	75.31±0.37	22.73±0.61
	AlphaEdit	73.92±0.29	26.59±0.49	72.96±0.26	25.92±0.51	72.63±0.31	24.95±0.50	44.67±0.46	13.79±0.49	69.85±0.36	23.04±0.59
	AnyEdit	**97.76**±0.11	**92.96**±0.24	**96.60**±0.19	**95.60**±0.35	**97.76**±0.14	**95.87**±0.23	**62.63**±0.44	**46.51**±0.59	**96.33**±0.21	**94.32**±0.23
	UnKE	98.34±0.15	93.33±0.26	93.38±0.21	78.42±0.32	98.62±0.14	96.37±0.22	59.62±0.44	32.89±0.59	98.33±0.13	95.42±0.20
	AnyEdit*	**99.86**±0.08	**99.68**±0.21	**94.70**±0.12	**85.75**±0.23	**99.95**±0.01	**99.98**±0.01	**64.24**±0.48	**45.31**±0.55	**99.89**±0.06	**99.69**±0.09
Qwen2.5-7B-It	Pre-edited	64.18±0.37	25.88±0.59	64.39±0.34	24.02±0.55	65.50±0.34	18.24±0.60	44.74±0.41	17.29±0.51	67.71±0.37	19.58±0.49
	FT-L	44.02±0.43	13.78±0.56	40.33±0.36	12.93±0.49	46.66±0.48	14.63±0.58	32.34±0.50	12.31±0.62	47.47±0.42	15.75±0.55
	MEND	69.49±0.38	27.77±0.61	62.01±0.44	27.92±0.57	69.54±0.54	25.47±0.49	52.86±0.40	22.81±0.54	69.40±0.32	32.39±0.44
	ROME	74.73±0.33	31.52±0.42	71.90±0.21	28.12±0.38	75.89±0.38	36.42±0.45	55.67±0.47	25.79±0.59	72.18±0.373	35.61±0.49
	MEMIT	78.03±0.30	38.04±0.47	76.50±0.31	28.65±0.50	77.19±0.32	38.95±0.48	56.04±0.40	25.73±0.57	73.15±0.32	34.39±0.54
	AlphaEdit	80.48±0.29	42.77±0.36	78.38±0.21	38.26±0.38	80.66±0.25	45.55±0.37	56.99±0.49	27.69±0.59	74.35±0.31	41.07±0.44
	AnyEdit	**98.05**±0.16	**94.89**±0.29	**93.56**±0.15	**79.98**±0.28	**98.08**±0.15	**95.09**±0.19	**65.40**±0.38	**43.49**±0.47	**98.14**±0.13	**96.39**±0.18
	UnKE	96.97±0.18	91.01±0.24	89.17±0.15	67.00±0.29	97.34±0.13	90.44±0.16	59.29±0.48	29.27±0.61	95.04±0.23	87.60±0.25
	AnyEdit*	**99.35**±0.12	**98.82**±0.24	**94.81**±0.13	**82.60**±0.26	**99.63**±0.09	**98.99**±0.10	**60.78**±0.39	**32.95**±0.59	**99.09**±0.07	**97.98**±0.10

Experiment

- EditEverything에서 모든 도메인에서 다른 baseline들을 압도하는 결과를 보여줌

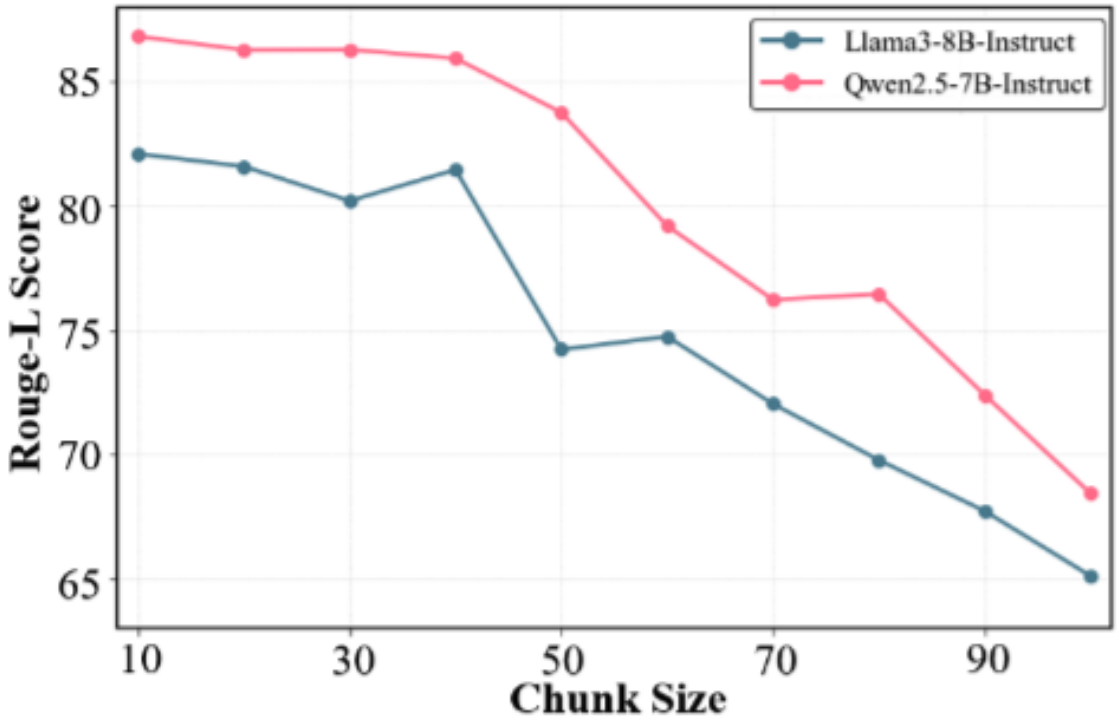


- EditEverything에서 편집하고자 하는 지식의 토큰수가 늘어날 때 AnyEdit이 가장 좋은 성능을 유지함



Experiment

- Chunk size에 따른 ablation study을 진행함
- chunk size가 작을 때 가장 성능이 좋았으나 그만큼 지식 편집하는데 걸리는 시간이 오래 걸린다는 단점이 있음. chunk size와 편집시간의 tradeoff를 고려했을 때 chunk size=40이 가장 적당했음



Conclusion

- 지식이 여러개의 토큰으로 이루어져 있을 때 효과적으로 지식을 편집하는 방법인 AnyEdit을 제시함
- AnyEdit은 multi-token edit에서 기존의 SOTA 성능을 보인 UnKE를 뛰어넘는 성능을 보임

Limitation

- 모든 토큰에 대해 파인튜닝한 baseline과의 비교결과가 궁금함
- 다른 방식의 chunking 방식을 적용했을 때의 성능이 궁금함

감사합니다