인공지능의 이해

Understanding Artificial Intelligence

유준수 (Joon Soo Yoo, Ph.D.)

Fall 2026

Defense & Advanced Security Lab (DAS Lab)
Department of Advanced Defense Technology and Industry
Jeonbuk National University

Understanding and Using LLMs

From Attention to RAG

Week 10

1. From Week 09 to Week 10

지난 Week에서는 Language Model이 Text를 생성하는 과정을 학습했다.

Text → Tokens → Embeddings

↓

Neural Network → Logits → Softmax

↓

Next Token Prediction

오늘의 질문

Neural Network는 여러 Token 사이의 관계를 어떻게 처리할까?
그리고 LLM이 모르는 정보를 답변에 활용하도록 하려면 어떻게 해야 할까?

1-1. Today's Mission

이번 Week에서는 LLM의 내부 구조와 실제 활용 방법을 함께 살펴본다.

1. Transformer와 Attention의 역할 이해
2. Query, Key, Value의 기본 개념 이해
3. Self-Attention의 계산 과정 이해
4. Prompt Engineering의 의미 이해
5. Hallucination과 Knowledge Limitation 이해
6. Retrieval-Augmented Generation (RAG) 이해

학습 목표

LLM이 Context를 처리하는 원리를 이해하고, Prompt와 External Knowledge를 활용하여 답변을 개선하는 방법을 설명할 수 있다.

1-2. Three Important Questions

이번 수업에서는 세 가지 질문을 해결한다.

Question 1

LLM은 문장 안의 Token들이 서로 어떤 관계를 가지는지 어떻게 계산할까?

Question 2

같은 LLM이라도 Prompt를 다르게 작성하면 왜 다른 답변이 생성될까?

Question 3

LLM이 학습하지 않은 최신 정보나 개인 문서를 활용하도록 만들 수 있을까?

2. Why Does Context Matter?

다음 두 문장을 비교해보자.

Sentence A

I deposited money in the **bank**.

Sentence B

I sat on the **bank** of the river.

두 문장에서 **bank**는 같은 단어이지만, 서로 다른 의미로 사용된다.

- Sentence A: 금융기관
- Sentence B: 강둑

오늘의 질문

Language Model은 동일한 Token인 bank의 의미가 문맥에 따라 달라진다는 것을 어떻게 처리할 수 있을까?

2-1. Initial Token Embedding

Week 09에서는 Token을 Embedding Vector로 변환하는 과정을 배웠다.

$$\text{Token} \rightarrow \text{Token ID} \rightarrow \text{Embedding}$$

예를 들어 다음과 같이 가정하자.

$$\text{bank} \rightarrow 100 \rightarrow [0.2,\ 0.5,\ -0.1]$$

동일한 Token ID는 동일한 Embedding Table의 행을 참조한다.

따라서 두 문장의 bank는 같은 초기 Token Embedding을 가질 수 있다.

핵심

초기 Token Embedding은 Token의 기본적인 Vector 표현이다.
하지만 이것만으로 문장 전체의 Context가 반영되는 것은 아니다.

2-2. From Embedding to Context

같은 bank라도 주변 Token은 서로 다르다.

Sentence	Context	Meaning
A	money, deposited	금융기관
B	river, sat	강둑

Transformer는 주변 Token의 정보를 활용하여 각 Token의 Representation을 변경한다.

$$\underbrace{[0.2, 0.5, -0.1]}_{\text{Initial Embedding}}$$

↓ Transformer

Sentence A: $[0.8, 0.1, 0.3]$

Sentence B: $[-0.2, 0.7, 0.9]$

주의

위 Vector는 설명을 위한 가상의 값이다.
실제 Transformer에서는 Position Information과 여러 Layer의 계산도 영향을 준다.

2-3. How Can We Use Context?

다음 문장을 다시 살펴보자.

I deposited money in the bank.

bank의 의미를 파악할 때 모든 Token이 똑같이 유용할까?

- money: 금융기관과 관련된 정보
- deposited: 돈을 맡기는 행동
- the: 문법적으로 필요하지만 의미 구분에는 상대적으로 덜 유용할 수 있음

생각해보기

bank의 Representation을 새롭게 계산할 때, 다른 Token의 정보를 서로 다른 비율로 반영하면 어떨까?

2-4. The Idea of Attention

각 Token의 정보를 서로 다른 비율로 결합한다고 생각해보자.

bank를 처리할 때 다음과 같은 Weight를 계산했다고 가정하자.

Token	Attention Weight
deposited	0.20
money	0.50
the	0.10
bank	0.20
Total	1.00

핵심

Attention은 다른 Token의 정보를 얼마나 반영할지 나타내는 Weight를 계산한다.
이 Weight를 이용하여 새로운 Representation을 만든다.

2-5. What Is Self-Attention?

Self-Attention에서는 같은 Sequence 안의 Token들이 서로의 정보를 활용한다.

예를 들어 다음 Token들이 있다고 하자.

$$[\text{deposited},\ \text{money},\ \text{the},\ \text{bank}]$$

Self-Attention에서는 다음을 수행한다.

1. 각 Token의 Vector를 준비한다.
2. Token 사이의 관련성 Score를 계산한다.
3. Score를 Attention Weight로 변환한다.
4. 다른 Token의 정보를 Weighted Sum으로 결합한다.

핵심

Self-Attention은 Token 사이의 정보를 결합하여 Contextual Representation을 계산하는 Mechanism이다.

2-6. Every Token Has Q, K, and V

Self-Attention에서는 각 Token마다 Query, Key, Value를 계산한다.

Token	Query	Key	Value
deposited	q_1	k_1	v_1
money	q_2	k_2	v_2
the	q_3	k_3	v_3
bank	q_4	k_4	v_4

여기서 중요한 점은 bank만 Q, K, V를 갖는 것이 아니라는 것이다.

핵심

모든 Token은 Q, K, V Vector를 가진다.
각 Token은 자신의 Query를 이용하여 참조 가능한 Token들의 Key와 비교한다.

2-7. What Do Query, Key, and Value Mean?

Query, Key, Value는 각각 다른 역할을 한다.

Vector	Intuitive Meaning
Query (Q)	현재 Token이 어떤 정보를 찾는가?
Key (K)	각 Token이 어떤 정보와 관련되어 있는가?
Value (V)	실제로 전달하고 결합할 정보는 무엇인가?

bank를 기준으로 생각하면,

- Query: bank가 찾으려는 정보의 표현
- Key: 각 Token이 가진 비교용 표현
- Value: 각 Token이 전달할 정보의 표현

핵심

Q와 K는 Attention Weight를 계산하는 데 사용된다.
V는 Weight에 따라 실제로 결합되는 Vector이다.

2-8. Where Do Q, K, and V Come From?

Q, K, V는 별도의 단어가 아니다.

각 Token의 Input Representation으로부터 계산되는 Vector이다.

Token i의 Input Vector를 x_i라고 하자.

$$q_i = x_i W_Q$$

$$k_i = x_i W_K$$

$$v_i = x_i W_V$$

W_Q, W_K, W_V는 Training을 통해 학습되는 Weight Matrix이다.

핵심

Value는 Initial Embedding 자체가 아니다.
Input Representation에 학습된 Weight Matrix를 적용한 결과이다.

2-9. What Does the Query of bank Do?

bank의 Representation을 업데이트한다고 하자.

bank의 Query를 다른 Token들의 Key와 비교한다.

$$q_{\text{bank}} k_{\text{deposited}}^{\top}$$

$$q_{\text{bank}} k_{\text{money}}^{\top}$$

$$q_{\text{bank}} k_{\text{the}}^{\top}$$

$$q_{\text{bank}} k_{\text{bank}}^{\top}$$

각 Dot Product는 하나의 Attention Score를 계산한다.

핵심

bank의 Query는 bank의 Key만 비교하는 것이 아니다.
참조 가능한 모든 Token의 Key와 비교한다.

2-10. Attention Weight Matrix

모든 Token의 Query와 Key를 비교하면 Attention Matrix를 계산할 수 있다.

다음은 가상의 Attention Weight Matrix이다.

Query / Key	deposited	money	the	bank
deposited	0.50	0.20	0.10	0.20
money	0.20	0.50	0.10	0.20
the	0.10	0.20	0.50	0.20
bank	**0.20**	**0.50**	**0.10**	**0.20**

- Row: Query Token
- Column: Key Token
- 각 Row의 Attention Weight 합은 1

핵심

bank의 Row는 bank가 다른 Token들의 정보를 얼마나 반영할지 나타낸다.

2-11. Step 1: Attention Score

다음 4개의 Token을 생각해보자.

$$[\text{deposited},\ \text{money},\ \text{the},\ \text{bank}]$$

bank의 Query와 각 Token의 Key를 비교한다.

$$q_{\text{bank}} = [1, 0]$$

$$k_{\text{deposited}} = [0, 0] \quad k_{\text{money}} = [1, 0]$$
$$k_{\text{the}} = [-1, 0] \quad k_{\text{bank}} = [0, 0]$$

Dot Product를 계산하면,

$$q_{\text{bank}} k_{\text{deposited}}^{\top} = 0$$
$$q_{\text{bank}} k_{\text{money}}^{\top} = 1$$
$$q_{\text{bank}} k_{\text{the}}^{\top} = -1$$
$$q_{\text{bank}} k_{\text{bank}}^{\top} = 0$$

핵심

bank의 Query를 모든 Key와 비교하여 4개의 Attention Score를 계산한다.

$$\text{Raw Scores} = [0, 1, -1, 0]$$

2-12. Step 2: Softmax

앞에서 계산한 Raw Score는 다음과 같다.

$$[0, 1, -1, 0]$$

Key Vector의 차원이 $d_k = 2$이므로,

$$s_i = \frac{q_{\text{bank}} k_i^\top}{\sqrt{2}}$$

$$s = [0,\ 0.707,\ -0.707,\ 0]$$

Softmax를 적용하면,

$$\alpha_i = \frac{e^{s_i}}{\sum_j e^{s_j}}$$

Token	Scaled Score	Weight
deposited	0	0.221
money	0.707	0.448
the	-0.707	0.109
bank	0	0.221

핵심

Softmax는 4개의 Score를 합이 1인 Attention Weight로 변환한다.

2-13. Step 3: Weighted Sum

bank의 Attention Weight를 계산했다.

$$\alpha_{\text{bank}} \approx [0.221,\ 0.448,\ 0.109,\ 0.221]$$

각 Token의 Value Vector를 다음과 같이 가정하자.

$$v_{\text{deposited}} = [1, 0] \quad v_{\text{money}} = [2, 0]$$
$$v_{\text{the}} = [0, 1] \quad v_{\text{bank}} = [0, 2]$$

Attention Output은 Value의 Weighted Sum이다.

$$o_{\text{bank}} = 0.221[1, 0] + 0.448[2, 0] + 0.109[0, 1] + 0.221[0, 2]$$

$$o_{\text{bank}} \approx [1.117,\ 0.551]$$

핵심

bank의 Attention Output은 4개 Token의 Value를 가중합한 새로운 Vector이다.
이는 bank의 초기 Embedding 자체가 아니다.

2-14. What Is an Attention Output?

Attention Output은 Initial Embedding과 다르다.

bank를 기준으로 생각하면,

$$o_{\text{bank}} = \alpha_1 v_{\text{deposited}} + \alpha_2 v_{\text{money}}$$

$$+\alpha_3 v_{\text{the}} + \alpha_4 v_{\text{bank}}$$

즉, bank의 Attention Output에는 다른 Token의 정보가 함께 반영된다.

핵심

Attention Output은 여러 Token의 정보를 결합하여 새롭게 계산한 Vector이다.
이것이 Contextual Representation을 만드는 중요한 과정이다.

2-15. The Attention Formula

지금까지 배운 과정을 정리해보자.

1. Query와 Key의 Dot Product 계산
2. Score를 $\sqrt{d_k}$로 Scaling
3. Softmax로 Attention Weight 계산
4. Value Vector의 Weighted Sum 계산

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

핵심

Attention은 Q와 K로 Weight를 계산하고, 그 Weight를 이용하여 V를 결합한다.

2-16. Mini Problem: Attention

어떤 Token의 Attention Weight가 다음과 같다고 하자.

$$\alpha = [0.2, 0.3, 0.5]$$

Value Vector는 다음과 같다.

$$V_1 = [1, 0]$$

$$V_2 = [0, 2]$$

$$V_3 = [2, 2]$$

Questions

1. 가장 높은 Attention Weight는?
2. Attention Weight의 합은?
3. Weighted Sum으로 계산한 Output Vector는?

2-17. Why Do We Need Multi-Head Attention?

Single-Head Attention만으로도 Contextual Representation을 만들 수 있다.

그렇다면 왜 여러 Head를 사용할까?

하나의 문장에는 여러 종류의 관계가 존재한다.

Example

The cat chased the mouse because it was hungry.

- 대명사가 가리키는 대상
- 주어와 동사의 관계
- 단어 사이의 의미적 관계

핵심

Multi-Head Attention은 서로 다른 Attention Pattern을 병렬로 학습할 수 있도록 한다.

2-18. How Does Multi-Head Attention Work?

각 Head는 서로 다른 Q, K, V Projection Matrix를 사용한다.

Input Representations

↓

Head 1 (Q_1, K_1, V_1) Head 2 (Q_2, K_2, V_2) Head 3 (Q_3, K_3, V_3)

↓

Concatenate Head Outputs

↓

Output Projection

핵심

Multi-Head는 여러 Attention 결과를 계산한 뒤 하나의 Representation으로 결합한다.

2-19. Single-Head vs. Multi-Head

Single-Head와 Multi-Head를 비교해보자.

	Single-Head	Multi-Head
Attention	하나의 Weight Pattern	여러 Weight Pattern
Projection	하나의 Q/K/V 세트	Head별 Q/K/V 세트
Output	하나의 Attention Output	여러 Output을 결합

핵심

Multi-Head는 단순히 Vector 차원을 늘리는 것과 다르다.
여러 Attention Distribution을 독립적으로 계산할 수 있다는 차이가 있다.

2-20. Why Multiple Transformer Blocks?

Multi-Head Attention을 한 번 수행해도 새로운 Representation을 얻을 수 있다.

그렇다면 왜 Transformer Block을 여러 번 반복할까?

$$X^{(0)} \xrightarrow{\text{Block 1}} X^{(1)}$$

$$X^{(1)} \xrightarrow{\text{Block 2}} X^{(2)}$$

$$X^{(2)} \xrightarrow{\text{Block 3}} X^{(3)}$$

핵심

각 Block은 이전 Block에서 계산한 Representation을 다시 입력으로 사용한다.
따라서 여러 단계에 걸쳐 더 복잡한 관계를 표현할 수 있다.

2-21. Multi-Head vs. Multi-Layer

두 개념은 서로 다른 목적을 가진다.

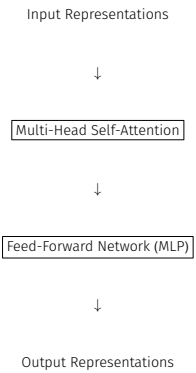
	Multi-Head	Multi-Layer
Structure	병렬 Attention	순차적 Block
Input	같은 Layer의 Representation	이전 Block의 Output
Purpose	다양한 관계를 동시에 처리	정보를 단계적으로 변환

핵심

Multi-Head는 여러 Attention Pattern을 병렬로 계산한다.
Multi-Layer는 계산된 Representation을 여러 단계에 걸쳐 다시 처리한다.

2-22. Inside a Transformer Block

Multi-Head Attention은 Transformer Block의 일부이다.



핵심

Transformer Block에는 Attention과 Feed-Forward Network가 포함된다.
실제 구조에서는 Residual Connection과 Normalization도 사용된다.

2-23. Why Position Information?

다음 두 문장을 비교해보자.

Sentence A

Dog bites man.

Sentence B

Man bites dog.

두 문장은 같은 단어들을 사용하지만 순서에 따라 의미가 달라진다.

핵심

Transformer는 Token의 내용뿐 아니라 Position Information도 활용한다.
이를 위해 Positional Embedding, Positional Encoding 등의 방법을 사용한다.

2-24. What Is a Causal Mask?

GPT 계열의 Autoregressive Language Model은 미래 Token을 미리 참조하지 못해야 한다.

예를 들어,

I love artificial intelligence

Predicting	Available Context
love	I
artificial	I love
intelligence	I love artificial

핵심

Causal Mask는 각 위치에서 미래 Token을 참조하지 못하도록 한다.
이를 통해 Autoregressive Prediction에 적합한 Attention을 계산한다.

2-25. From Attention to Next Token Prediction

Attention Output이 곧바로 다음 Token의 Probability가 되는 것은 아니다.

Input Tokens

↓

Embeddings + Position Information

↓

Transformer Blocks

↓

Final Hidden State

↓

Logits → Softmax → Next Token

핵심

마지막 Token 위치의 Final Hidden State를 이용하여 다음 Token의 Logit을 계산한다.

2-26. Interactive Transformer Visualization

Transformer의 내부 동작을 Interactive Visualization으로 살펴보자.

Transformer Explainer

Open Transformer Explainer

다음 요소들을 확인해보자.

1. Token Embeddings
2. Query, Key, Value
3. Attention Weights
4. Transformer Blocks
5. Next Token Probabilities

질문

Transformer는 어떻게 Context를 반영하여 다음 Token을 예측할까?

2-27. What We Learned About Transformers

이번 Section에서 배운 내용을 정리해보자.

- Initial Embedding은 Token의 기본 표현이다.
- Attention은 다른 Token의 정보를 결합한다.
- 모든 Token은 Q, K, V를 가진다.
- Q와 K로 Attention Score를 계산한다.
- Softmax로 Attention Weight를 계산한다.
- Value의 Weighted Sum으로 Output을 만든다.
- Multi-Head는 여러 Attention Pattern을 계산한다.
- Multi-Layer는 Representation을 반복적으로 변환한다.

핵심

Transformer는 Attention과 여러 Layer의 계산을 통해 Context를 반영한 Token Representation을 만들고, 이를 Next Token Prediction에 활용한다.

3. From Transformer to Prompt Engineering

지금까지는 LLM이 Context를 처리하는 방법을 살펴보았다.

그렇다면 Context 자체를 변경하면 어떤 일이 발생할까?

Different Prompts

↓

Different Context Representations

↓

Different Output Probabilities

핵심

같은 LLM이라도 Prompt가 달라지면 생성되는 답변이 달라질 수 있다.

3-1. What Is Prompt Engineering?

Prompt Engineering은 LLM이 원하는 작업을 수행하도록 입력 Prompt를 설계하는 방법이다.

Prompt A

Explain artificial intelligence.

Prompt B

Explain artificial intelligence to a first-year university student.
Use three simple examples and avoid mathematical equations.

핵심

Prompt에 대상, 목적, 조건, 출력 형식 등을 명확히 제시하면 원하는 결과를 얻는 데 도움이 될 수 있다.

3-2. Elements of a Good Prompt

Prompt를 작성할 때 다음 요소를 고려할 수 있다.

Element	Meaning
Task	무엇을 수행해야 하는가?
Context	어떤 배경정보가 필요한가?
Constraints	어떤 조건을 지켜야 하는가?
Output Format	어떤 형식으로 답해야 하는가?

Mini Problem

다음 Prompt를 더 구체적으로 바꾸어보자.

Explain machine learning.

대상 독자, 예시 개수, 답변 길이를 추가하면 어떻게 달라질까?

3-3. Zero-shot vs. Few-shot

LLM에게 작업을 요청할 때 예시를 제공할 수도 있다.

Zero-shot Prompt

Classify the sentiment as Positive or Negative.
Text: I really enjoyed this movie.

Few-shot Prompt

I love this movie. → Positive
This movie was boring. → Negative
The acting was excellent. → ?

핵심

Zero-shot은 예시 없이 작업을 요청한다.
Few-shot은 몇 개의 예시를 Context에 제공하여 원하는 출력 Pattern을 유도한다.

3-4. Does Prompt Engineering Change the Model?

Prompt Engineering을 사용하면 Model의 Weight가 변경될까?

Prompt A → LLM → Output A

Prompt B → Same LLM → Output B

핵심

일반적인 Prompt Engineering은 Model Parameter를 다시 학습하지 않는다.
Input Context를 변경하여 Model의 출력을 유도하는 방법이다.

4. Can an LLM Know Everything?

LLM은 Training Data를 통해 많은 정보를 학습할 수 있다.

하지만 다음과 같은 정보는 정확하게 알지 못할 수 있다.

- Training 이후에 발생한 사건
- 공개되지 않은 회사 내부 문서
- 개인이 작성한 새로운 보고서
- 최신 일정이나 변경된 규정

Example

오늘 새롭게 작성한 연구보고서의 내용을 LLM에게 질문한다면 어떻게 될까?

핵심

LLM의 학습된 지식만으로 모든 질문에 정확히 답할 수는 없다.

4-1. Hallucination and External Knowledge

LLM은 모르는 정보에 대해서도 그럴듯한 답변을 생성할 수 있다.

이러한 문제를 Hallucination이라고 한다.

그렇다면 새로운 정보를 활용하기 위해 항상 Model을 다시 Training해야 할까?

Question + Relevant Documents

↓

LLM → Answer

핵심

외부 문서를 Context로 제공하면 Model Weight를 변경하지 않고도 새로운 정보를 답변에 활용할 수 있다.

5. What Is RAG?

RAG는 Retrieval-Augmented Generation의 약자이다.

질문과 관련된 외부 정보를 검색하여 LLM의 답변 생성에 활용하는 방법이다.

User Question

↓

Retrieve Relevant Documents

↓

Question + Retrieved Context

↓

LLM → Answer

핵심

RAG는 외부 Knowledge를 검색하여 LLM의 Input Context에 추가하는 방식이다.

5-1. Three Steps of RAG

RAG는 크게 세 단계로 구분할 수 있다.

Step	Description
Retrieval	질문과 관련된 Document를 검색한다.
Augmentation	검색된 Document를 Prompt에 추가한다.
Generation	LLM이 Context를 이용하여 답변을 생성한다.

핵심

RAG의 핵심은 LLM이 답변을 생성하기 전에 관련된 외부 정보를 제공하는 것이다.

5-2. Example: University Information

어떤 대학의 새로운 학사 규정을 질문한다고 생각해보자.

User Question

What is the minimum attendance requirement for this course?

Retrieved Document

Course Policy:
Students must attend at least 75% of scheduled classes.

LLM은 검색된 Document를 참고하여 답변을 생성할 수 있다.

핵심

LLM이 사전에 해당 규정을 학습하지 않았더라도, 관련 Document를 제공하면 답변에 활용할 수 있다.

5-3. How Do We Retrieve Documents?

RAG에서는 질문과 관련된 문서를 찾는 과정이 중요하다.

대표적인 Retrieval 방법은 다음과 같다.

Method	Description
Keyword Search	질문과 문서의 단어 일치 정도를 활용한다.
Semantic Search	질문과 문서의 의미적 유사도를 활용한다.

핵심

Retrieval은 LLM에 제공할 Relevant Context를 선택하는 과정이다.
검색 품질은 최종 답변의 품질에 영향을 줄 수 있다.

5-4. Keyword Search vs. Semantic Search

다음 질문을 생각해보자.

Question

How can I reset my password?

다음 두 문서가 있다고 하자.

1. How to reset your password
2. I forgot my password. How do I change it?

Keyword Search는 단어의 일치를 활용한다.

Semantic Search는 표현이 달라도 의미가 비슷한 문서를 찾을 수 있다.

핵심

Semantic Search는 문장의 의미적 유사성을 활용하여 관련 문서를 검색한다.

5-5. Embeddings for Semantic Search

Week 09에서는 Token Embedding을 배웠다.

Semantic Search에서는 문장이나 문서를 Vector로 표현할 수 있다.

$$\text{Question} \rightarrow \mathbf{q} \in \mathbb{R}^d$$

$$\text{Document} \rightarrow \mathbf{d} \in \mathbb{R}^d$$

두 Vector 사이의 유사도를 계산한다.

$$\text{sim}(\mathbf{q}, \mathbf{d}) = \frac{\mathbf{q}^\top \mathbf{d}}{\|\mathbf{q}\| \|\mathbf{d}\|}$$

핵심

Cosine Similarity는 두 Vector의 방향이 얼마나 비슷한지를 측정한다.
유사도가 높은 Document를 검색 결과로 선택할 수 있다.

5-6. From Documents to Context

관련 문서를 찾았다면 LLM의 Prompt에 추가한다.

Prompt with Retrieved Context

Context:
Students must attend at least 75% of scheduled classes.

Question:
What is the minimum attendance requirement?

Instruction:
Answer using the provided context.

핵심

Augmentation은 검색된 정보를 LLM의 Input Context에 추가하는 과정이다.

5-7. RAG vs. Fine-tuning

외부 정보를 활용하는 방법이 RAG만 있는 것은 아니다.

Method	Main Idea
RAG	외부 정보를 검색하여 Context에 추가한다.
Fine-tuning	추가 Training으로 Model Parameter를 변경한다.

핵심

RAG는 일반적으로 Model Weight를 변경하지 않는다.
Fine-tuning은 추가적인 Training을 통해 Model Weight를 조정하는 방법이다.

5-8. Does RAG Eliminate Hallucination?

RAG를 사용하면 Hallucination이 완전히 사라질까?

반드시 그렇지는 않다.

- 관련 없는 Document가 검색될 수 있다.
- 검색된 Document 자체가 틀릴 수 있다.
- 중요한 Document를 검색하지 못할 수 있다.
- LLM이 제공된 Context를 잘못 해석할 수 있다.

주의

RAG는 외부 근거를 제공하여 답변을 개선하는 데 도움이 될 수 있지만, 답변의 정확성을 보장하지는 않는다.

5-9. What We Learned

이번 Week에서는 LLM의 내부 구조와 활용 방법을 살펴보았다.

- Transformer는 Attention을 활용한다.
- Self-Attention은 Token 사이의 관계를 처리한다.
- Query와 Key로 관련성 Score를 계산한다.
- Softmax로 Attention Weight를 계산한다.
- Value의 Weighted Sum으로 정보를 결합한다.
- Prompt Engineering은 Input Context를 설계한다.
- RAG는 External Knowledge를 검색하여 활용한다.

핵심

LLM의 내부 구조를 이해하는 것과 LLM을 실제 시스템에 활용하는 것은 서로 연결되지만 다른 문제이다.

6. Next Week: From LLM to AI Agent

이번 Week에서는 LLM이 외부 문서를 활용하여 답변하는 방법을 배웠다.

그렇다면 다음과 같은 작업도 가능할까?

Next Question

LLM이 계산기를 사용하거나, 외부 도구를 실행하고, 여러 단계를 거쳐 목표를 수행하도록 만들 수 있을까?

다음 Week에서는 다음 내용을 살펴본다.

- AI Agent
- Tool Calling
- Reasoning and Action
- Observation and Feedback
- Agent Workflow

Next Week

From Language Models to Autonomous Agents

Thank You

Questions?