WHAT HAPPENS INSIDE GPT

GPT 안에서 일어나는 일

LLM 이론 집중 강좌

이나베슌타로 지음

수식 없이, GPT 속에서 무슨 일이 일어나는가

가중치·학습·추론부터 어텐션·트랜스포머·RAG까지 — 직관만으로 완주하는 LLM 이론

쉽게 · 깊게 · 재밌게 — 전 8장 완전 수록

GPT 안에서 일어나는 일
LLM 이론 집중 강좌 — 수식 없이, GPT 속에서 무슨 일이 일어나는가, 끝까지
(전자판 제목: [쉽깊잼] LLM 집중이론)

초판 발행: 2026년 6월
지은이: 이나베슌타로

본서는 『모델의 본체와 추론의 원리』를 한 권으로 완결시킨, 독립된 해설서입니다.
본문의 정보는 2026년 4~5월 시점 기준입니다. LLM 분야는 변화가 빠르므로,
최신 모델 사양·가격은 각 공식 문서(Anthropic, OpenAI, Google 등)를 참고하십시오.

본문에서 인용한 모든 외부 출처는 부록 B에 정리했습니다.

FROM THE AUTHOR

저자의 말

이 책은, LLM을 배우다가 「여기서 막혔다」 싶었던 자리를 중심으로, 수식에 기대지 않고 되도록 쉽게 풀어낸 책입니다.

이 책이 약속하는 것은 하나입니다. 쉽게 들어가고, 깊게 파고, 재밌게 끝까지. 어려운 개념일수록 가장 쉬운 문 하나를 찾아 먼저 열고, 깊이는 본문에서 끝까지 책임지며, 그 길에 작은 재미 한 조각을 남깁니다. 권위로 누르는 책이 아니라, 같이 끝까지 걷자고 손을 내미는 책입니다.

여기서 가장 크게 효과를 본 것은, 최신 기법 그 자체보다도, 이 책에서 다루는 기초를 얼마나 정확하게 「내 것」으로 만들었는가였습니다. 기초가 단단하면 그 위에 올라가는 새 기법은 이름만 바뀐 같은 이야기로 보이기 시작합니다.

처음 LLM의 속을 들여다봤던 그 무렵, 저도 여러 자리에서 오래 멈춰 섰습니다. ChatGPT의 답이 한 글자씩 돌아오는 건 도대체 왜인지. 모르는 것까지 어떻게 그렇게 자신만만하게 지어내는지. 애초에 컴퓨터는 말의 의미를 어떻게 숫자로 바꾸는지. 흔히 듣는 「가중치(weight)」란 결국 무엇인지. 정리하고 다시 봐도, 그 자리는 누구나 한 번은 멈춰 서는 곳이었습니다. 해설이 친절해서 그냥 지나칠 수 있었더라도, 돌아보면 그 자리에서 한 번은 멈췄어야 했습니다.

이 강좌는 그 「멈춰 서는 자리」를 다시 한 번 찬찬히 되짚으며, 『여기서 한 번, 멈추자』고 분명히 표시해 둔 책입니다. 처음엔 가중치·벡터·학습·추론 넷이면 정리될 줄 알았습니다. 그런데 진행하다 보면 문맥 임베딩(embedding)과 어텐션(attention)까지 한 줄로 이어집니다. 그래서 하나의 이야기로 풀어냈습니다. 어려운 수식 뒤로 숨지 않고, 직관으로 끝까지 갑니다.

각 장의 절, Q&A, 질문의 흐름은 「독자는 분명 여기서 멈출 것이다」를 전제로 일부러 배열했습니다. 어디도 생략하지 않았고, 걸려 넘어지기 쉬운 자리일수록 두껍게 썼습니다.

이 책의 여덟 개 장은——

- 제1장 LLM의 구조 — 가중치·벡터·토큰·임베딩의 정체
- 제2장 학습 — 산을 내려가기·오차역전파·자기지도학습
- 제3장 추론 — KV 캐시·VRAM·Softmax·샘플링·Temperature·BPE
- 제4장 추론의 진화 — 답하기 전에 「혼잣말」을 하는 Reasoning 모델

· 제5장 디코딩 — 1토큰을 1문장으로 잇기·Beam·Min-P·Stop·JSON 강제

· 제6장 어텐션 — 「그 말이 누구를 가리키는가」를 숫자로 알아내기·Q·K·V

· 제7장 트랜스포머 — 어텐션을 쌓아 올려 ChatGPT가 되기·Multi-Head·RoPE

· 제8장 RAG와 LLM의 한계 — 클라우드/로컬·할루시네이션·검색으로 보완

가중치에서 시작해, 학습, 추론, 추론의 진화, 디코딩, 어텐션, 트랜스포머, 그리고 마지막 장에서는 LLM의 한계와 그것을 검색으로 보완하는 RAG까지. 이 여덟 개 장으로, 『모델의 본체』에서 『그것을 현장에서 쓸 수 있는 도구로 바꾸는 자리』까지가, 끝까지 한 줄로 이어져 보입니다. 수식의 벽 뒤로 도망치지 않고, 제가 배우면서 「여기서 한 번 멈추자」고 느낀 자리를, 그대로 이정표로 남겨 두었습니다.

읽다가 막히면, 그 자리에서 멈춰 서십시오. 먼저 나아가도, 뒤의 장은 앞 장을 전제로 합니다. 걸린 자리를 두 번 읽는 편이, 다음 장을 열 페이지 훑어 읽는 것보다 훨씬 빠릅니다. 각 장 끝에 「내 말로 답해 보기」라는 짧은 점검 질문을 두었습니다. 거기서 답이 막히면, 그 장을 한 번 더 읽고 다음으로 넘어가십시오.

AI 공부는, 진도를 빨리 빼는 것보다, 내 말로 설명할 수 있는가가 훨씬 중요한 분야입니다.

2026년 5월 이나베슌타로

TABLE OF CONTENTS

목차

PROLOGUE

CHAPTERS

프롤로그 — 왜 이 강좌를 시작했는가

> 한 번 끝까지 배우고, 정리하며 다시 봐도, 이 자리는 누구나 한 번은 멈춰 서는 곳입니다. 그래서 「여기서 멈추자」를 분명히 표시해 둔 책입니다.
>
> — 본서의 한 줄

1. 다시 한 번 정리한 이유

어느 날, 또다시 가중치·벡터·토큰의 정체가 궁금해졌습니다. 한 번은 끝까지 배운 적이 있는데도, 몇몇 자리는 떠올리며 다시 봐도 「아, 여기, 처음 봤을 때 막혔지」가 그대로 되살아납니다. 그 자리는 해설이 친절해서 그냥 지나칠 수 있었더라도, 돌아보면 한 번은 멈춰 섰어야 할 자리였습니다.

처음엔 가중치·벡터·학습·추론 넷이면 정리될 줄 알았습니다. 그런데 진행하다 보면, 문맥 임베딩(embedding)·어텐션(attention)·트랜스포머(Transformer)까지 함께 가야 할 필요가 생깁니다. 그래서 하나의 이야기로 풀어냈습니다. 제1장에서는 가중치와 벡터의 영역을 끝까지, 제2장은 학습, 제3장은 추론, 제4장은 추론의 진화(Reasoning)까지 — 우선 이 네 장이 출발점이었습니다.

이 강좌는, 정리하며 「여기서 한 번, 멈추자」를 분명히 표시한 책입니다. 어려운 수식 뒤로 도망치지 않고, 직관으로 끝까지 갑니다. 각 장의 절·Q&A·질문의 흐름은, 「독자는 분명 여기서 멈출 것이다」를 전제로 일부러 배열했습니다. 어디도 생략하지 않고, 그대로 옮겼습니다.

본서의 약속

이 책은 「한 번 더 읽으면, 막혔던 자리가 풀리는」 책입니다. 어려운 수식으로 나아가지 않고, 한 번 끝까지 걸어 본 사람이 「여기서 멈추자」고 놓아 둔 이정표를 그대로 남겼습니다. 어떤 장에서 막혔다고 느껴지면, 거기를 한 번 더 읽고 나아가십시오. 그게, 다음 장으로 빨리 닿는 길입니다.

2. 이 책의 여덟 개 장

이 책에는, 가중치에서 시작해 학습·추론·추론의 진화·디코딩·어텐션·트랜스포머, 그리고 마지막 장의 LLM의 한계와 RAG로 이어지는 여덟 개 장이 들어갑니다. 한 호흡으로 읽으면, 모델의 본체에서, 그것을 현장에서 쓸 수 있는 도구로 바꾸는 자리까지, 끝까지 심이 이어져 보입니다.

제1장 · LLM의 구조
스팸 모델에서, LLM의 80억 개 가중치까지. 같은 식, 다른 규모. 토큰 → 어휘 ID → 임베딩 벡터. 어휘집과의 대조로 끝나는 출력층.

제2장 · 학습
안개 낀 산을 내려가기 — gradient descent. 오차역전파 = 회사 부서로의 책임 추궁. 미니배치·학습률·검증 데이터까지.

제3장 · 추론
한 글자마다 32개 층을 다시 도는 이유 — autoregressive. KV 캐시·VRAM·메모리 대역폭·양자화·Softmax·샘플링·Temperature·BPE.

제4장 · 추론의 진화
답하기 전에 「혼잣말」을 하는 모델. 23×47조차 못 풀던 LLM이, Erdős의 수학 난제를 80분 만에 푼 사건. Test-time compute·o1·Extended Thinking.

제5장 · 디코딩
1토큰을 매번 best로 골라도, 1문장이 best라는 보장은 없다. 빔서치·Min-P·Repetition Penalty·Stop·JSON 강제(Constrained Decoding).

제6장 · 어텐션
「울었다」의 주어가 고양이라는 걸 어떻게 아는가. 내적·Q·K·V·softmax를, 고양이 한 문장으로 숫자까지 따라간다.

제7장 · 트랜스포머
어텐션을 쌓아 올리면, 어째서 ChatGPT가 되는가. Multi-Head·위치 인코딩(RoPE)·FFN·Residual/LayerNorm·Encoder/Decoder·FlashAttention까지.

제8장 · RAG와 LLM의 한계
인터넷 없는 현장에서. 클라우드/로컬 LLM·할루시네이션·다섯 가지 한계·AGI 논쟁. 검색으로 보완하는 RAG(벡터·내적)·파인튜닝·시스템 프롬프트·Llama.

이 여덟 개 장으로, 『모델의 본체』에서 『그것을 현장에서 쓸 수 있는 도구로 바꾸는 자리』까지가 이어집니다. 가중치에서 시작해, 학습, 추론, 추론의 진화, 디코딩, 어텐션, 트랜스포머, 그리고 한계와 RAG까지. 한 줄로 읽으면, 모델의 속과 「답이 나오는 구조」, 그리고 「그 답을 어떻게 보완하는가」의 심까지, 끝까지 보입니다.

3. 추천하는 읽는 법

처음부터 끝까지 순서대로 읽는 것이, 가장 확실합니다. 뒤로 갈수록, 앞 장이 전제가 됩니다. 그렇다고 누구나 그만한 시간을 낼 수 있는 건 아닙니다. 세 가지 루트를 마련했습니다.

루트 A · 완전 처음

제1장부터 제8장까지 순서대로. 누적 대략 8~9시간. 주에 한 장씩 페이스라면, 두 달쯤이면 한 권을 다 읽을 수 있습니다. 가장 확실한 루트.

루트 B · 추론·KV·VRAM만 빠르게

제1장을 슥 훑고 → 제3장(추론)으로 직행 → 제4장(Reasoning). 대략 2시간. 다만 가중치·임베딩 감각이 약하면, 제3장 중간의 양자화에서 막힙니다. 그때는 제1장으로 돌아가면 됩니다.

루트 C · Reasoning만

제4장으로 직행. 대략 30분. 다만 「테스트 시점 compute」의 정체와, 「진짜 추론인가」라는 질문의 무게가 옅어지기 쉽습니다. 가능하면 제3장의 일부(Softmax / 샘플링)만이라도 먼저.

4. 딱 하나, 약속

KEY INSIGHT

어떤 장에서 이해가 애매하면, 그 장에서 멈춰 서십시오. 먼저 나아가도, 그 뒤의 장은 앞 장을 전제로 합니다. 막힌 장을 두 번 읽는 편이, 다음 두 장을 훑어 읽는 것보다, 훨씬 빠릅니다.

각 장 끝에는 「내 말로 답해 보기」라는 짧은 점검 질문이 들어갑니다. 거기서 답이 막히면, 그 장을 한 번 더 읽고 다음으로 넘어가는 걸 권합니다. AI 공부는, 진도를 빨리 빼는 것보다, 내 말로 설명할 수 있는가가 훨씬 중요한 분야입니다.

여담 — 한 장의 AI 칩에, 세계가 담겨 있다

본편에 들어가기 전에, 잠깐만 옆길로. (LLM의 이론만 급한 분은, 이 여담을 건너뛰고, 다음 「그럼, 시작합시다」로 넘어가도 괜찮습니다.) 이 책은 LLM의 「속」을 푸는 책이지만, 그 속을 실제로 돌리고 있는 것

은, 세계 곳곳의 회사가 한 장에 담긴 「칩」입니다. 이 지도를 먼저 펼쳐 두면, AI 뉴스나 경제의 흐름까지, 한 줄로 이어 읽을 수 있게 됩니다.

고성능 LLM을 돌리려면, 고성능 GPU가 필요합니다. 그 GPU에서 압도적인 회사가, 미국의 엔비디아(Nvidia)입니다. 2025년 10월 말, 엔비디아는 시가총액 5조 달러를 역사상 처음으로 넘긴, 인류 최초의 기업이 되었습니다. 2026년 5월 시점에도 약 5.2조 달러로 세계 1위. AI용 데이터센터 칩의 약 8할을 쥐고 있습니다.

그런데 재미있게도, 그 엔비디아는 「설계」만 합니다. 반도체를 실제로 「제조」하는 것은, 대만의 TSMC(대만적체전로제조)입니다. 엔비디아는 설계도만 넘기고, 실물은 TSMC가 만든다——이를 팹리스(fabless)라고 부릅니다. 그리고 세계 최첨단의 「연산용」 반도체는, 그 대부분이 이 TSMC에서 나옵니다(위탁 제조=파운드리에서 약 7할, 최첨단에서는 약 9할).

그런데 반도체란 무엇일까요. 금속처럼 전기를 잘 통하는 것을 「도체」, 고무나 플라스틱처럼 통하지 않는 것을 「절연체」라고 합니다. 반도체는 그 딱 중간으로, 조건에 따라 전기를 통하게 하거나, 멈추거나 할 수 있습니다. 게다가, 그 조건을 사람이 제어할 수 있습니다. 이를 1초에 수십억 번 전환해서, 전기가 흐르면 1, 멈추면 0, 이라는 신호로 만듭니다. 컴퓨터는 0과 1만으로 온갖 계산을 하므로(이진법), 이 「통하고·멈추기를 고속으로 전환할 수 있는」 반도체가, 연산의 심장부가 됩니다.

그것을 만드는 곳이 TSMC입니다. 반도체 한 장을 위해, 매년 「조」 단위로 연구개발과 설비를 쏟아붓는——하나의 구멍만을, 누구보다 깊이 파 내려가는 회사이기에, 다른 회사는 좀처럼 따라잡지 못합니다.

그 TSMC에도, 「이게 없으면 안 돌아가는」 급소가 있습니다. 네덜란드의 ASML입니다. 최첨단 칩을 새겨 넣는 노광장치(EUV)를, 세계에서 9할 이상, 단 한 회사가 독점하고 있습니다. ASML의 기계가 없으면, TSMC조차 최신 칩을 만들지 못합니다. 또 설계의 「틀」에서는, 영국의 ARM도 빼놓을 수 없습니다——스마트폰 CPU의 99%가, 이 ARM의 설계를 쓰고 있습니다.

그리고, 엔비디아의 연산 칩 「바로 옆」에 없어서는 안 되는 것이, 메모리. 특히 초고속인 HBM입니다. 이건 한국의 삼성전자와 SK하이닉스가 세계의 대부분을 쥐고 있습니다(자세히는 제3장에서). 즉 한 장의 AI 칩은, 연산 칩(대만 TSMC 제)과, 메모리(한국 제)를, 하나의 토대 위에 나란히 붙여 만든 「합작」인 것입니다.

역할	어디의 누구
설계	미국 — Nvidia
연산 칩의 제조	대만 — TSMC
그 제조장치(EUV)	네덜란드 — ASML
설계의 틀(IP)	영국 — ARM
메모리(HBM)	한국 — 삼성·SK하이닉스
빌려주는 곳(클라우드)	미국 — Amazon·Microsoft·Google

이 지도에서 한국은? — 메모리라는 급소

이 지도에서 한국이 선 자리를 보면, 한 곳이 또렷합니다. 메모리입니다. 칩을 「설계」하는 주역은 미국, 「제조」의 주역은 대만이지만, 엔비디아의 연산 칩 바로 옆에 반드시 붙는 초고속 메모리 HBM은, 삼성전자와 SK하이닉스 두 회사가 세계의 대부분을 쥐고 있습니다. 화려한 무대 한가운데는 아니어도, 「이게 없으면 최신 AI 칩이 완성되지 않는」 자리입니다.

왜 메모리가 그토록 결정적인지는 제3장에서 끝까지 풉니다. 여기서는 한 줄만. AI 칩은 연산만으로 돌지 않습니다. 그 연산이 다룰 80억 개의 가중치를, 바로 옆에서 빠르게 먹여 주는 메모리가 없으면, 아무리 빠른 연산 칩도 굶습니다. 그 「먹여 주는 속도」를 쥔 것이 HBM이고, 그 HBM의 패권을 쥔 것이 한국의 두 회사라는 이야기입니다.

마지막의 「빌려주는 곳」에도 한 줄. GPT나 Claude를 학습시키고, 돌리려면, 이 GPU가 수만 대 필요합니다. 한 장에 수백만 원입니다. 직접 갖출 수 있는 사람은 많지 않습니다. 그래서, 거대한 자본으로 엔비디아에서 GPU를 수만~수십만 대 사들여, 「데이터센터」라는 건물에 늘어놓고, 필요한 사람에게 빌려주는 회사가 있습니다. Amazon(평소엔 쇼핑), Microsoft(Windows나 Office), Google(YouTube나 Gmail)——이런 얼굴로 알려져 있지만, 이들이 가장 강한 것은, 사실 이 「클라우드」, 즉 GPU 대여업입니다. ChatGPT는 Microsoft의 클라우드 위에서, Claude는 Amazon·Google의 클라우드 위에서 돌고 있습니다.

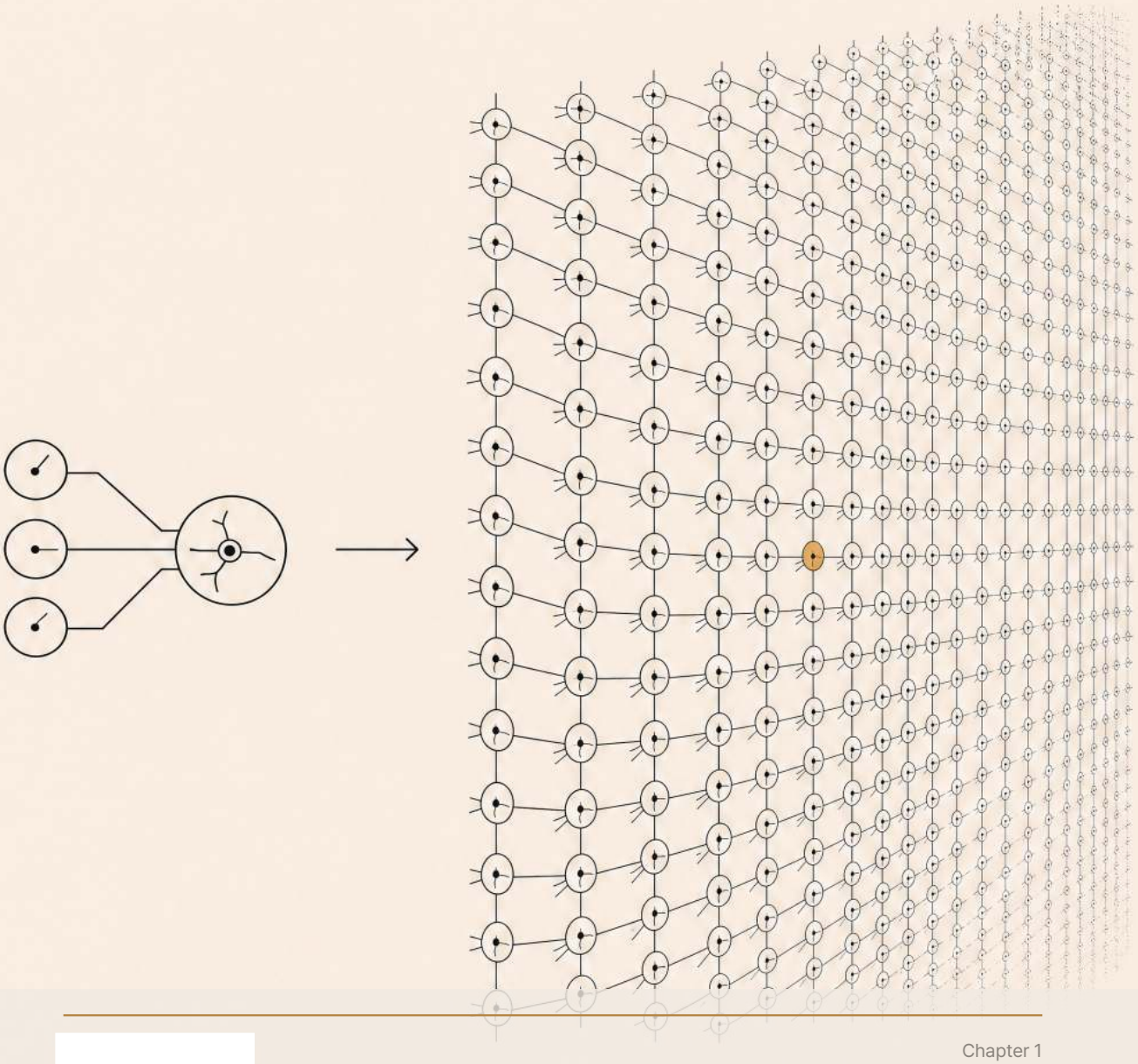
그래서 AI 뉴스는, 엔비디아 한 회사의 이야기가 아닙니다. 설계 → 제조 → 메모리 → 클라우드, 라고 강물처럼 흘러갑니다. 이 한 장의 지도를 머리 한구석에 놓아 두면, 「돈이 다음에 어디로 향할지」까지 읽을

수 있게 됩니다. 그리고 이 책이 앞으로 파 내려갈 것은, 그 강의 가장 위쪽 상류——LLM의 「속」 그 자체입니다.

그럼, 시작합시다

첫 장은, 가장 단순한 모델 — 스팸메일 판별기 — 부터 시작합니다. 그 모델의 입력 3개와 가중치 3개가, LLM의 80억 개로 넓어져 가는 「같은 식, 다른 규모」가, 이 책 전체의 출발점입니다.

이 한 줄이 머리에 들어오면, 이어서 나오는 가중치·토큰·임베딩·학습·추론이, 모두 같은 「어마어마하게 큰 행렬곱」 위에 얹힌 다른 단계라는 게 보이기 시작합니다.



1

LLM의 구조

가중치·토큰·임베딩의 정체

01

LLM의 구조

이 장에서 붙잡고 갈 것

(1) 가중치란 무엇인가——스팸 모델에서 LLM의 80억 개까지. (2) 자연어를 어떻게 숫자로 바꾸는가——토큰화와 임베딩. (3) 벡터·차원·문맥 임베딩의 정체. (4) 동음이의어(눈·배·밤·말)는 같은 토큰인가? (5) 마지막 출력층은 결국, 어휘집과의 대조.

1. 가장 작은 모델부터——스팸메일 판별기

복잡한 LLM 이야기로 들어가기 전에, 단순한 모델부터 시작합시다. 메일이 스팸인지 아닌지 판단하는 모델이 있다고 합시다.

이 모델이 보는 것은, 딱 3개뿐입니다.

- 「당첨」이라는 단어가 몇 번 나왔는가
- 「회의」라는 단어가 몇 번 나왔는가
- 느낌표가 몇 개 있는가

각 항목의 중요도를 직관적으로 정해 보면, 이렇게 됩니다.

- 「당첨」→ 많이 나오면 스팸 확률↑(양수, +2.0)
- 「회의」→ 많이 나오면 오히려 정상 메일(음수, -1.5)
- 느낌표 → 조금 스팸 같음(작은 양수, +0.3)

이 +2.0, -1.5, +0.3이야말로 가중치입니다. 영어로는 weight.

뉴런이 하는 일은, 한 줄로 끝납니다.

```
출력 = (x1 × w1) + (x2 × w2) + (x3 × w3)
```

각 입력에 가중치를 곱해서 더한다. 정말 이게 전부입니다.

예를 들어 어떤 메일에 「당첨」이 3번, 「회의」가 0번, 느낌표가 5개 있으면.

```
출력 = (3 × 2.0) + (0 × -1.5) + (5 × 0.3)
      = 6.0 + 0 + 1.5
   = 7.5  → 스팸 확률 높음!
```

LLM은 결국, 이걸 어마어마한 규모로 하고 있을 뿐. 이 장에서는 먼저, 이 단순한 식을 머리에 새기고 나아갑시다.

2. 가중치의 부호와 크기

여기서 조금 헷갈리는 데가 있습니다. 「+2.0이 -1.5보다 더 중요한 거 아니야?」 라고 생각하기 쉽거든요.

결론부터 말하면, 둘 다 똑같이 중요합니다. 부호와 크기는, 다른 것을 뜻합니다.

- 부호 (+/-) : 입력이 결과에 어느 방향으로 영향을 주는가
- 절댓값　　: 얼마나 강하게 영향을 주는가

그래서 +5.0도 -5.0도, 똑같이 중요합니다. 그저 방향이 반대일 뿐. 정말 안 중요한 입력은, 가중치가 0에 가까운 입력입니다. 0에 가까우면, 곱해 봤자 결과에 영향이 거의 없으니까요.

정리하면——

- 가중치 +5.0 = 결과를 위로 강하게 밀어 올린다
- 가중치 -5.0 = 결과를 아래로 강하게 끌어내린다
- 가중치 0.0 = 이 입력은 무시해도 된다
- 가중치 0.1 = 거의 영향 없음

3. 그럼 입력은? 단어를 어떻게 숫자로 바꾸는가

스팸 모델은 입력이 단순했습니다. 「당첨 횟수, 회의 횟수, 느낌표 개수」처럼, 수로 셀 수 있는 것이었으니까요.

그런데 LLM에는, 사용자가 「오늘 날씨 어때?」 같은 자연어를 그대로 넣습니다. 이걸 어떻게 숫자로 바꿔야, 모델이 처리할 수 있을까요.

여기서 2단계가 필요합니다.

- 1단계: 토큰화 (Tokenization)——문장을 작은 조각으로 나눈다
- 2단계: 임베딩 (Embedding)——각 조각을 수치 벡터로 변환한다

이 2단계는 자주 혼동됩니다. 같이 풀어 가 봅시다.

4. 토큰이 뭔가요?

토큰은, 모델이 처리할 수 있는 최소 단위까지 쪼갠 조각입니다. 한국어 LLM이 「나는 사과를 먹었다」를 받으면, 대략 이렇게 쪼갭니다.

```
"나" "는" "사과" "를" "먹었" "다"
```

(실제 쪼개는 방식은 모델마다 다릅니다. 어떤 모델은 「사과를」을 통째로 1토큰으로, 어떤 모델은 더 잘게 쪼갭니다.)

여기서의 토큰은, 아직 숫자가 아닙니다. 그냥 텍스트 조각입니다. 모델이 실제로 처리하려면, 이 토큰을 숫자로 바꿔야 합니다.

방법은 2단계입니다.

4-1. 먼저 토큰 ID로 변환

모델은 어휘집 (vocabulary)이라는 걸 갖고 있습니다. 마치 사전처럼, 모든 토큰에 번호가 매겨져 있습니다.

```
"나"    → ID: 1024
"는"    → ID: 47
"사과"  → ID: 8932
"를"    → ID: 89
```

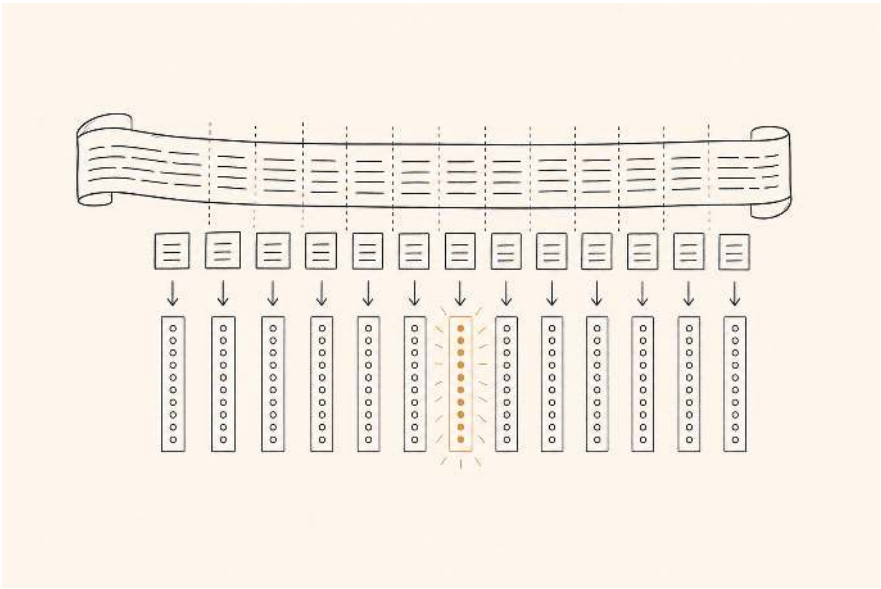
4-2. 다음으로 ID로 임베딩 벡터를 꺼낸다

어휘집의 각 ID마다, 임베딩 벡터가 미리 저장돼 있습니다.

ID 1024 → [0.12, -0.45, 0.78, ...] ← 이게 "나"의 임베딩
ID 8932 → [0.34, -0.51, 0.82, ...] ← 이게 "사과"의 임베딩

여기만은 붙잡고 가자

토큰 = 텍스트의 조각(「사과」). 토큰 ID = 그 조각의 번호(8932). 임베딩 벡터 = 그 토큰의 의미를 담은 숫자 묶음([0.34, -0.51, ...]). 토큰은 글자고, 벡터는 그 토큰의 의미를 표현한 것입니다. 같은 게 아닙니다.



토큰화와 임베딩——문장을 토큰으로 쪼개고, 각 토큰을 수백~수천 차원의 벡터로 변환한다.

5. 벡터, 별것 아닙니다

벡터라는 말은 막연하고 어려워 보이지만, 정체는 정말 별것 아닙니다.

> 벡터 = 숫자가 정해진 순서로 늘어선 묶음.
>
> — 벡터의 정의

이게 전부입니다.

- 2차원 벡터: [3, 5] → 숫자 2개
- 3차원 벡터: [3, 5, -2] → 숫자 3개
- 100차원 벡터: 숫자 100개

「차원」이라는 말은 SF 영화의 평행우주 같은 신비로운 울림이지만, 수학적으로는 그냥 벡터 안에 들어 있는 숫자의 개수입니다. 그게 다예요.

그럼, 왜 굳이 묶을까요. 단어 하나의 의미를 표현하는 데, 숫자 하나로는 부족합니다. 사고실험을 해 봅시다. 만약 「사과」의 의미를 숫자 하나로 표현한다면, 어떻게 할 수 있을까요. 0.5? 0.8? 어떤 숫자를 골라도, 사과의 색·모양·맛이 전부 사라져 버립니다. 숫자 하나로는 의미가 너무 빈약한 거예요.

그래서, 여러 측면을 묶어서 표현합니다.

```
사과 = [빨강:0.8, 둥굶:0.9, 단맛:0.7, 과일:1.0]
```

이렇게 4개의 측면으로 표현하면 4차원 벡터가 됩니다. 이게 그 단어의 「의미의 좌표」입니다.

재밌는 건, 의미가 비슷한 단어끼리는, 벡터 공간에서 가까이 위치한다는 것.

```
강아지 → [0.8, 0.2, 0.5]
개     → [0.79, 0.21, 0.51]  ← 강아지와 거의 같은 자리
자동차 → [0.1, 0.9, 0.3]     ← 멀리 떨어진다
```

6. 그럼 차원은, 대체 몇 개로 정하나?

여기서 자연스럽게 솟는 의문. 사과를 4차원으로 표현한다고 했는데, 왜 4개? 5개면 안 돼? 모든 단어가 똑같이 4차원이어야 하나?

답을 하나씩.

Q1. 차원 수는 누가 정하나?

모델 설계자가 정합니다. 정해진 정답은 없습니다. 모델을 설계할 때 정하는 하이퍼파라미터의 하나입니다.

실제 모델이 쓰는 차원 수를 보면, 이렇습니다.

- BERT base: 768차원
- BERT large: 1024차원
- GPT-3: 12288차원
- Llama 3: 4096차원

모델마다 다릅니다. 768차원은, 그냥 BERT가 이 값이라는 것뿐. 신성한 숫자가 아닙니다.

Q2. 차원이 많을수록 좋은가?

대체로 그렇습니다. 차원이 많을수록, 단어의 의미를 더 풍부하게 표현할 수 있으니까요. 4차원이면 사과를 4개 측면으로만 볼 수 있지만, 4096차원이면 4096개 측면으로 볼 수 있습니다.

다만 트레이드오프가 있습니다. 차원이 커질수록——

- 표현력은 풍부해지지만
- 계산량이 폭발적으로 늘고
- 메모리도 더 먹고
- 학습도 어려워집니다

그래서 작은 모델은 768차원쯤, 큰 모델은 12288차원 같은 식으로, 모델의 크기에 맞춰 정합니다.

Q3. 모든 단어가 같은 차원이어야 하나?

네. 한 모델 안에서는, 모든 토큰의 임베딩이 같은 차원입니다. BERT를 쓰면 사과도 768차원, 자동차도 768차원, 사람도 768차원. 그러지 않으면, 행렬곱 같은 계산이 일관되게 돌지 않거든요.

7. 충격적인 사실——각 차원이 무엇을 뜻하는지, 사람도 모른다

여기까지 저는 사과를 [빨강, 둥긂, 단맛, 과일]로 표현해 왔습니다. 사실 이건, 이해를 돕기 위한 설명용 거짓말입니다. 미안해요.

진짜 LLM의 임베딩 벡터는, 이렇게 됩니다.

```
사과 = [0.34, -0.51, 0.82, 0.11, -0.27, 0.65, ...] (4096개)
```