The Attention Mechanism

Lecture 12 · ES 667: Deep Learning

Prof. Nipun Batra
IIT Gandhinagar · Aug 2026

Learning outcomes

By the end of this lecture you will be able to:

1. Explain attention as **differentiable dictionary lookup**.
2. Derive **scaled dot-product** and justify the √d_k.
3. Distinguish **Bahdanau (additive)** from **Luong (multiplicative)**.
4. Implement **QKV self-attention** in PyTorch.
5. Apply **causal masking** to get a GPT-style decoder.
6. State the **$O(n^2)$ complexity** wall and its consequences.

Recap · where we are

Module 6 opens. The previous lecture ended on a cliff-hanger:

- **Seq2Seq** works for short sentences.
- The **fixed-size context vector** can't hold everything for long ones.
- **BLEU collapses** past ~30 tokens.

Today we fix it.

> REFERENCE
>
> Today maps to **Prince Ch 12** (early sections). This is the single most influential idea in deep learning between backprop and diffusion.

The one-line fix

KEY IDEA

Don't force the decoder to read from one fixed vector. Let it **peek at every encoder state** and decide which ones matter for the current step.

Bahdanau et al. 2014 · "Neural Machine Translation by Jointly Learning to Align and Translate."

Why one vector can't hold everything

Think about translating a 40-word English sentence into French. A Seq2Seq encoder has to compress:

- every word's identity
- the full syntactic structure
- the tense, number, gender, mood of every verb and noun
- the coreference chains ("it" refers to "the box")
- the sentiment

...into a **single 512-dim vector**. One number for every 0.3 bits of meaning.

> WATCH OUT
>
> No matter how big you make that vector, there's always a sentence that exceeds it. The information-theoretic problem is **fundamental**, not an engineering issue.

The shift in viewpoint

Old · push mode

Encoder *pushes* a summary forward. Decoder takes whatever fits.

- One-shot summarization.
- Lossy — compression is mandatory.

New · pull mode

Decoder *pulls* information on demand. Encoder keeps everything around.

- No compression required.
- Decoder chooses what's relevant *per step*.

Attention is the mechanism for the *pull* — a differentiable version of "look up the word I need right now."

Four questions

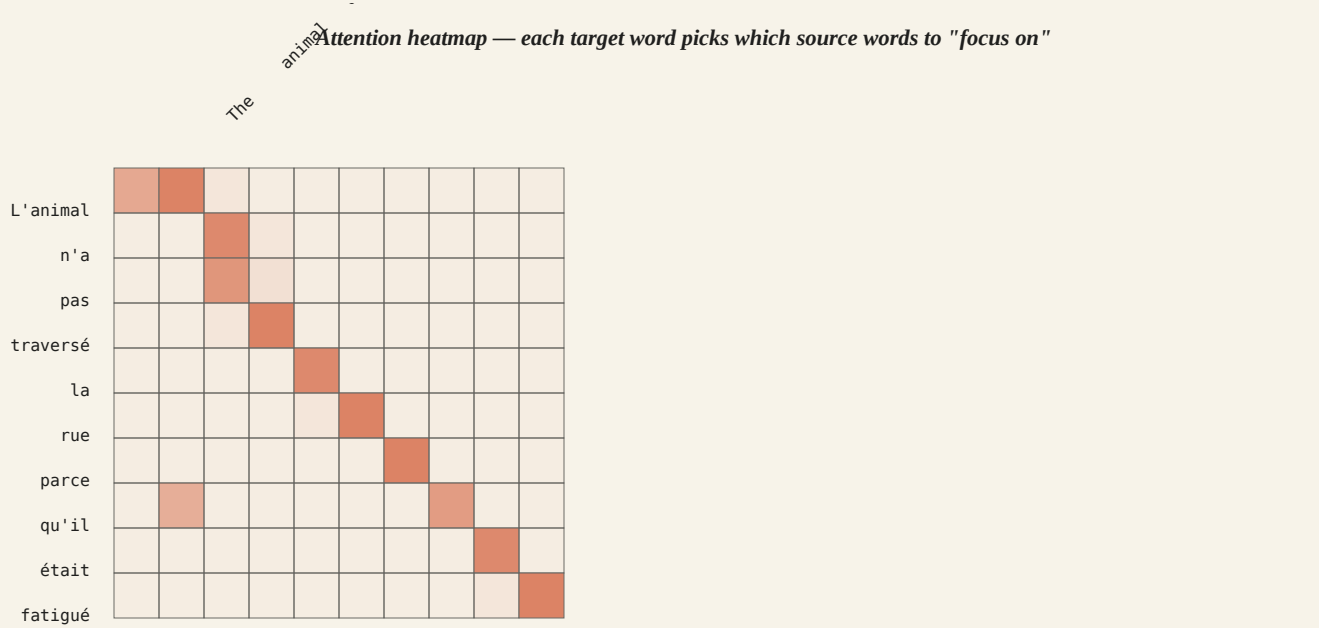
1. What does attention look like — literally, as a heatmap?
2. What are Q, K, V and why "retrieval"?
3. Why do we divide by $\sqrt{d_k}$?
4. What is **self**-attention and how does it differ from cross-attention?

PART 1

Attention as soft alignment

The heatmap view

What attention looks like



Notice the **off-diagonal entry** · "qu'il" attends to BOTH "it" AND "animal"
— the model learned to resolve coreference on its own.

Interpretation

Every target word computes a **distribution over source words**. Darker cell = more attention mass.

Three things to notice:

1. **Rough diagonal** — most alignments are monotonic.
2. **"traversé" attends to "cross"** — semantic alignment across a language barrier.
3. **"qu'il" attends to BOTH "it" AND "animal"** — coreference resolution, emergent from training.

> INTUITION
>
> No one told the model what "it" refers to. It learned this by minimizing translation loss. Attention made linguistic structure visible for the first time.

> IN PRACTICE
>
> ▶️
>
> Interactive: hover over source tokens, see attention weights in real time — attention.

PART 2

From Bahdanau to QKV

Two parameterizations · one abstraction

A 3×3 worked example · before the math

Source · "the cat slept" Target step · decoding French for "cat" → *"chat"*

DERIVATION

Encoder states (toy 2-d): $h_1 = [1, 0]$ (the), $h_2 = [0, 1]$ (cat), $h_3 = [0.2, 0.1]$ (slept)

Decoder state: $s_t = [0.1, 0.9]$ (about to emit *chat*)

Raw scores (dot products): $s_t \cdot h_1 = 0.1$, $s_t \cdot h_2 = 0.9$, $s_t \cdot h_3 = 0.11$

Softmax: $[0.24, 0.54, 0.22]$ — *"cat"* gets the largest weight.

Context: $c_t = 0.24h_1 + 0.54h_2 + 0.22h_3 = [0.29, 0.56]$

The decoder reads a **weighted mixture**, dominated by the relevant word. That's one step of attention.

Bahdanau (additive) attention · 2014

INTUITION

Analogy · the compatibility test. Plain dot product compares profiles directly. A *learned* score function projects both into a shared compatibility space (W_1, W_2), combines (`tanh`), and an "expert" $\mathbf{v}$ reads out a single 8/10 score. More expressive than raw dot product when vectors aren't aligned.

Score between decoder state s_t and encoder state h_i:

$$e_{t,i} = \mathbf{v}^\top \tanh(W_1 h_i + W_2 s_t), \quad \alpha_{t,i} = \mathrm{softmax}_i(e_{t,i}), \quad c_t = \sum_i \alpha_{t,i}\, h_i$$

Bahdanau · worked numeric, term-by-term

Toy 2-d. $h_i = [2, 1]^\top$, $s_t = [1, 3]^\top$.

$$W_1 = \begin{pmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{pmatrix}, \quad W_2 = \begin{pmatrix} 0.5 & 0.6 \\ 0.7 & 0.8 \end{pmatrix}, \quad \mathbf{v} = \begin{pmatrix} 0.9 \\ 0.1 \end{pmatrix}$$

Step 1 · project.

- $W_1 h_i = [0.2 + 0.2,\ 0.6 + 0.4]^\top = [0.4, 1.0]^\top$
- $W_2 s_t = [0.5 + 1.8,\ 0.7 + 2.4]^\top = [2.3, 3.1]^\top$

Step 2 · add + non-linearity.

$U = [0.4 + 2.3,\ 1.0 + 3.1]^\top = [2.7, 4.1]^\top$

$\tanh U \approx [0.99, 1.00]^\top$

Step 3 · final dot product with $\mathbf{v}$.

$e_{t,i} = 0.9 \cdot 0.99 + 0.1 \cdot 1.00 = 0.891 + 0.100 = \mathbf{0.991}$

This single number is the alignment score for **one** (s_t, h_i) pair. Compute for all i, then softmax.

Luong (multiplicative) attention · 2015

A simpler score, no learned MLP:

$$e_{t,i} = \boldsymbol{s}_t^\top h_i$$

- **Faster** — one matrix multiply instead of a two-layer MLP.
- **Equivalent in expressiveness** when you have enough data (the projections in QKV absorb Bahdanau's W_1, W_2).

  This is the version Vaswani et al. kept for the Transformer in 2017, with one small but crucial addition: the $\sqrt{d_k}$ scaling.

Why multiplicative won

Two reasons the ML community moved from Bahdanau to Luong:

1. **Hardware** — a dot product is one matrix multiply; GPUs love it. Bahdanau's MLP has element-wise tanh, which is slower per op and harder to batch.
2. **Enough capacity elsewhere** — once we added learned W_Q, W_K projections (next section), we no longer needed the $\tanh(W_1 h + W_2 s)$ to *learn* the similarity. The dot product of projections already gives it.

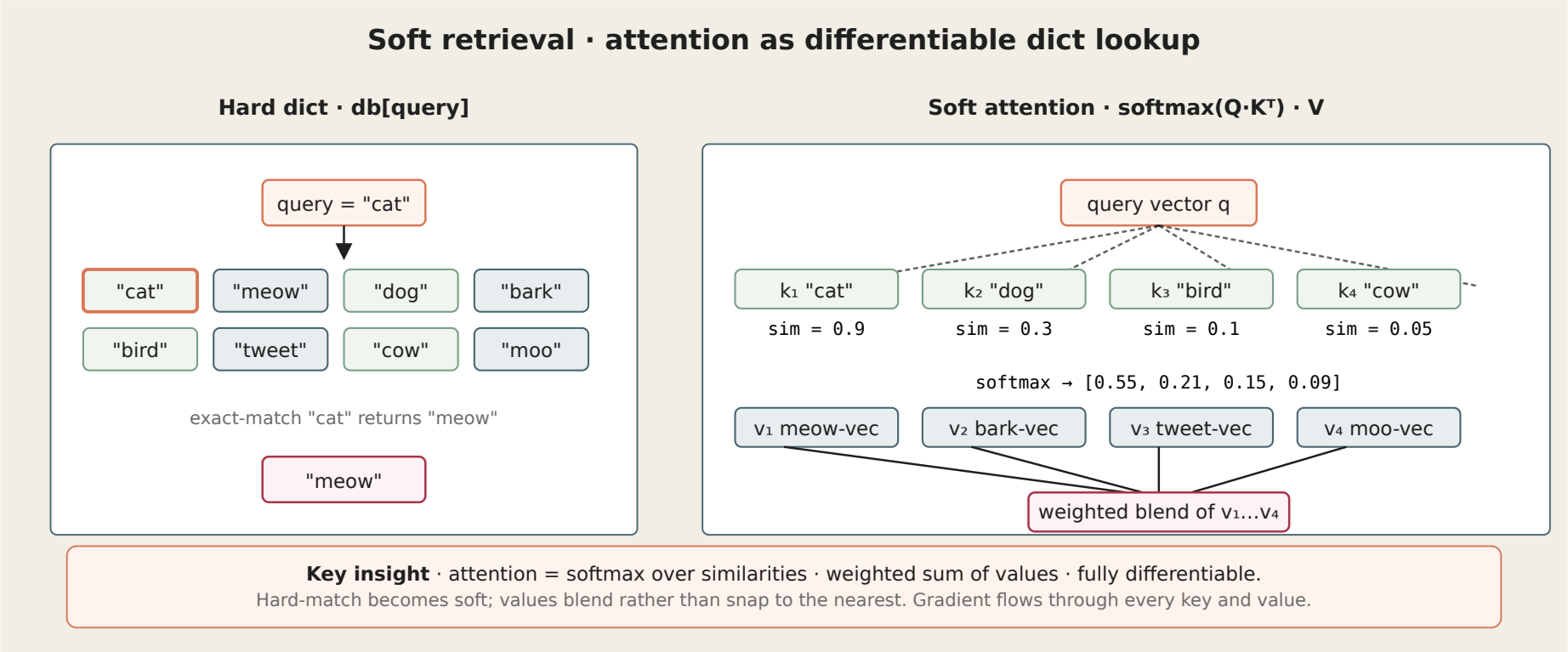
> INTUITION
>
> Pattern in DL · keep the core operation small + fast, push learning into the linear layers around it. This is also why attention beat CNN for sequences — CNN's inductive bias was baked in; attention's bias is *learned*.

PART 3

Q, K, V · the clean abstraction

Attention as database retrieval

Soft retrieval · picture



Why Q, K, V are three different things · the library analogy

KEY IDEA

You walk into a library with a question.

- **Query** · "I need info on the 2008 financial crisis."
- **Key** · the title on each book's spine ("The 2008 Meltdown", "The Great Depression"...).
- **Value** · the actual *content* inside each book.

You use the query to **match against keys**. The match score decides how much of each book's **value** (content) you blend into your final answer.

That's it · attention is a soft library lookup. The next slides formalize the math; the analogy is the whole intuition.

The retrieval metaphor

Imagine a Python dictionary lookup:

```
db = {"cat": "meow", "dog": "bark"}
query = "cat"
result = db[query]      # returns "meow"
```

Attention is the **soft** version of this:

- **Query** — what you're looking for (from the decoder).
- **Key** — what each encoder state announces itself as.
- **Value** — what each encoder state actually *contains*.

  Score keys against the query → softmax → use weights to blend values.

Soft retrieval · why three roles not one

You could imagine an attention mechanism where K and V are the same thing. Early models did exactly this (Luong 2015). So why separate them?

KEY IDEA

Keys say "what I am"; values say "what I contribute."

A word like "bank" in a sentence should be *found* by the query "financial", but *contribute* the full contextual embedding. Keys for retrieval, values for content.

- K optimized for similarity with plausible queries.
- V optimized to carry whatever downstream layers need.

Separating them doubles the parameter count of one head but roughly doubles the expressiveness too.

The Python-dict mental model · extended

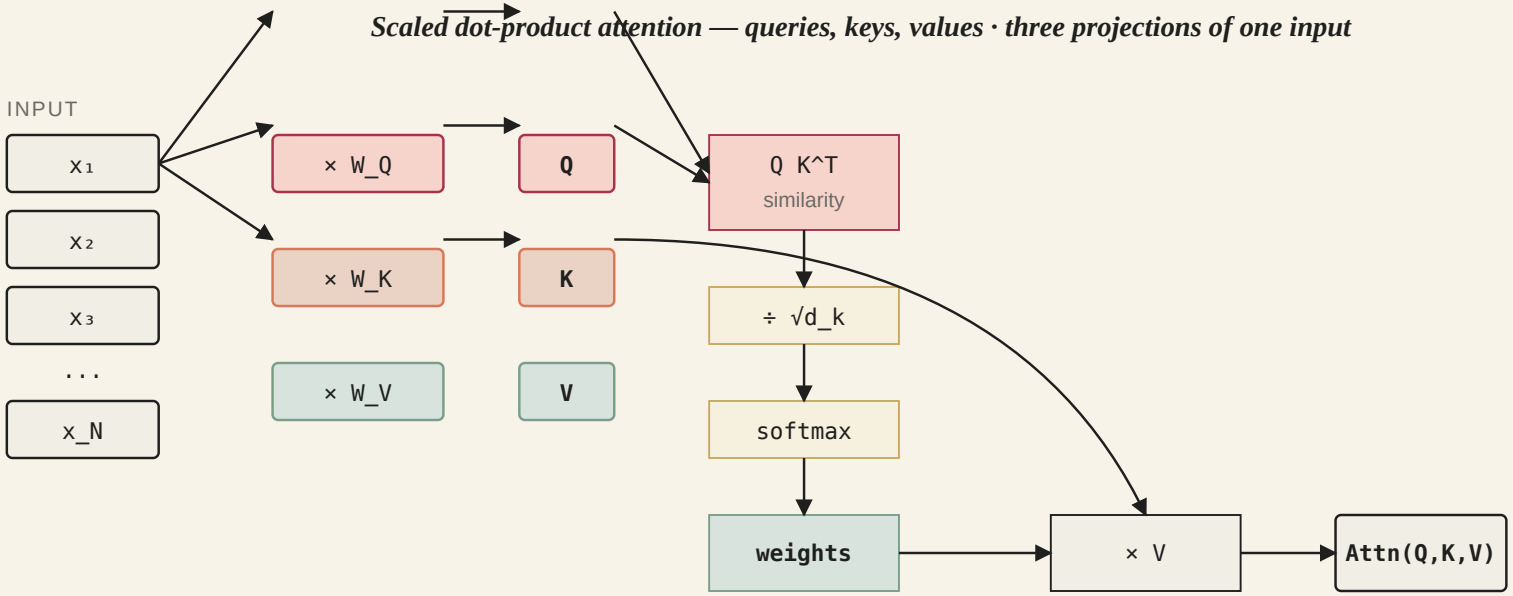
```
# Hard retrieval
db = {k1: v1, k2: v2, k3: v3}
out = db[query]                      # exact match → one value

# Soft retrieval (attention)
scores  = [sim(query, k) for k in [k1, k2, k3]]   # similarities
weights = softmax(scores)                          # probabilities
out     = weights[0]*v1 + weights[1]*v2 + weights[2]*v3
```

INTUITION

Attention is **differentiable dictionary lookup**. The network's parameters shape what "sim" means and what each key/value represents. Everything else is the soft version of `db[query]`.

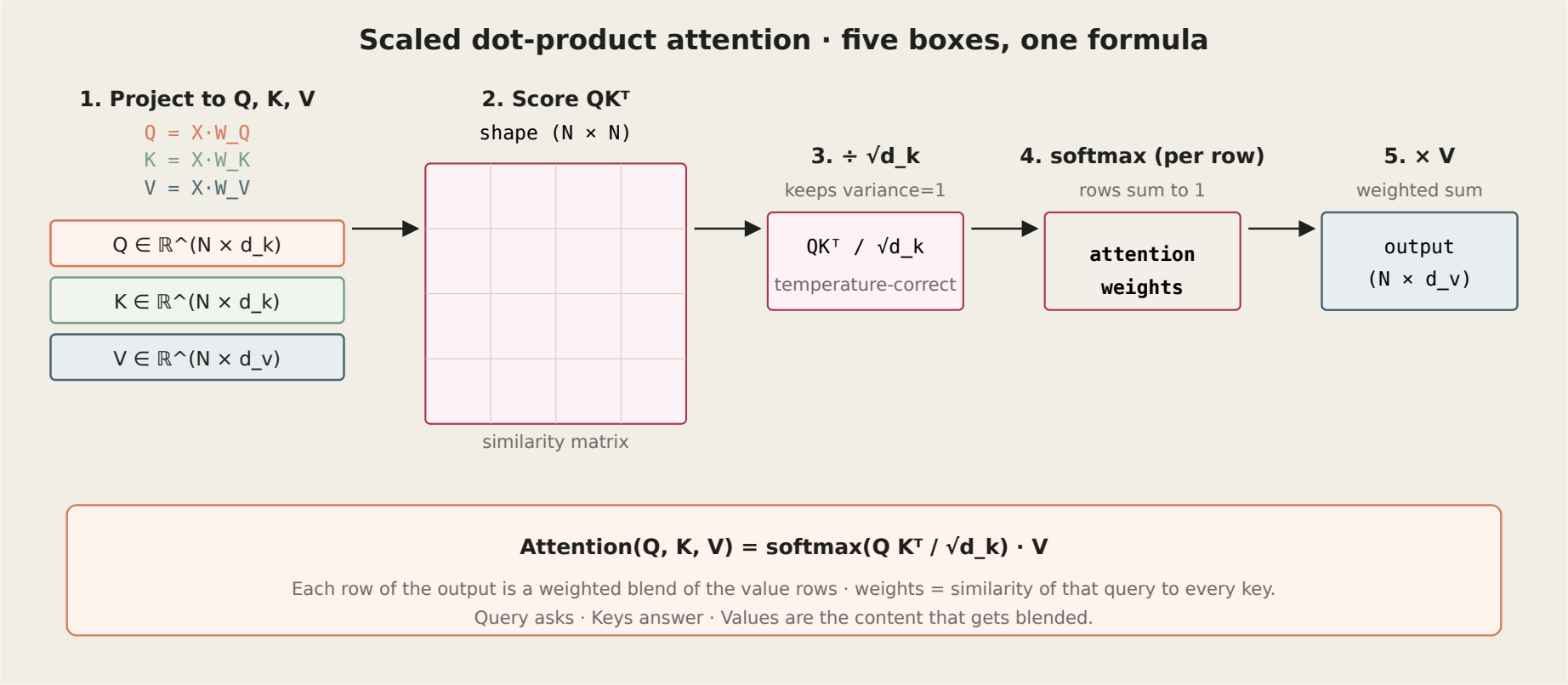
QKV · the computation



$$\text{Attention}(Q, K, V) = \text{softmax}(Q K^T / \sqrt{d_k}) \cdot V$$

database retrieval · Q asks, K answers, V pays

Scaled dot-product · step by step



Scaled dot-product · worked numeric (4 steps)

Tiny example · 2 tokens, $d_k = 2$.

$$Q = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, K = \begin{pmatrix} 1 & 1 \\ 2 & 0 \end{pmatrix}, V = \begin{pmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{pmatrix}, \sqrt{d_k} = \sqrt{2} \approx 1.414.$$

Step 1 · scores.

$QK^\top = \begin{pmatrix} 1 & 2 \\ 1 & 0 \end{pmatrix}$ — token 1's query matches token 2 better (score 2 > 1).

Step 2 · scale.

$$QK^\top/\sqrt{2} = \begin{pmatrix} 0.707 & 1.414 \\ 0.707 & 0 \end{pmatrix}$$

Step 3 · row-wise softmax.

Row 1: $\mathrm{softmax}([0.707, 1.414]) = \left[\frac{2.03}{6.14}, \frac{4.11}{6.14}\right] \approx [0.33, 0.67]$

Row 2: $\mathrm{softmax}([0.707, 0]) \approx [0.67, 0.33]$

$$A = \begin{pmatrix} 0.33 & 0.67 \\ 0.67 & 0.33 \end{pmatrix}$$

Step 4 · weighted sum.

$$AV = \begin{pmatrix} 0.33 \cdot 0.1 + 0.67 \cdot 0.3 & 0.33 \cdot 0.2 + 0.67 \cdot 0.4 \end{pmatrix} = \begin{pmatrix} 0.234 & 0.334 \end{pmatrix}$$

One actor, three roles

INTUITION

Analogy. Eddie Murphy in *The Nutty Professor* — same person (X), different costumes and makeup (W_Q, W_K, W_V), three characters.

- **Query** role · "I'm the hero — what do I need?"
- **Key** role · "Here's what I am."
- **Value** role · "Here's the info I have to offer."

The network learns the costumes (W matrices) that make attention work.

QKV · projection worked example

Input vector for "cat" · $x = [1, 2, 3, 4]^\top$ ($d = 4$). Project to $d_k = 2$ via three 4×2 matrices.

$$W_Q = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \\ 0 & 0 \end{pmatrix}, \quad W_K = \begin{pmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 0 \\ 1 & 1 \end{pmatrix}, \quad W_V = \begin{pmatrix} 1 & 1 \\ 0 & 0 \\ 0 & 1 \\ 1 & 0 \end{pmatrix}$$

- $q = x^\top W_Q = [1 \cdot 1 + 3 \cdot 1,\ 2 \cdot 1 + 3 \cdot 1] = [4, 5]$
- $k = x^\top W_K = [2 + 4,\ 1 + 4] = [6, 5]$
- $v = x^\top W_V = [1 + 4,\ 1 + 3] = [5, 4]$

The single input $[1, 2, 3, 4]$ now plays **three roles**: query $[4, 5]$, key $[6, 5]$, value $[5, 4]$.

The full Q, K, V matrices are this calculation repeated for every token. PyTorch:

```
class AttentionHead(nn.Module):
```


```
class AttentionHead(nn.Module):
```

PART 4

Why $\sqrt{d_k}$?

The scaling that makes attention work

Why softmax can get "spiky"

INTUITION

Analogy · grading on a curve.

- Scores 1–10 · 7 vs 8 are close. Curve is smooth.
- Scores 1–1000 · 700 vs 800 are worlds apart. The 801 wins; everyone else gets ~0%. Curve is **spiky**.

Unscaled dot products behave like the second case as d_k grows. Scaling factor brings us back to the first.

Variance of unscaled dot products

For Q_i, K_j with i.i.d. $\mathcal{N}(0,1)$ entries:

$$S = Q_i^\top K_j = \sum_{k=1}^{d_k} q_k k_k$$

Variance of a sum of independent terms = sum of variances:

$$\mathrm{Var}(S) = \sum_{k=1}^{d_k} \mathrm{Var}(q_k k_k) = d_k$$

(For two independent zero-mean unit-variance variables, $\mathrm{Var}(qk) = 1$.)

Standard deviation $\sqrt{d_k}$ → typical magnitudes scale **like** $\sqrt{d_k}$.

With $d_k = 512$ raw scores are $\sim \pm 22$. Softmax of $[22, -22, 22, \ldots]$ → nearly one-hot. Scale by $1/\sqrt{d_k}$ → variance back to 1.

Worked numeric · the scaling factor in action

$d_k = 256 \Rightarrow \sqrt{d_k} = 16$. A typical raw-score vector for a query: $[15.5, -16.1, 2.3]$.

Without scaling.
$\exp(15.5) \approx 5.4 \times 10^6$. Softmax $\approx [0.999995,\ 0.000000,\ 0.000005]$ — essentially **one-hot** → gradient ≈ 0 → learning stalls.

With scaling. Divide by 16: $[0.97, -1.01, 0.14]$.
$\exp \approx [2.64, 0.36, 1.15]$, sum ≈ 4.15. Softmax $\approx [0.64, 0.09, 0.27]$ — soft, gradients flow.

Numeric demo · softmax at different scales

DERIVATION

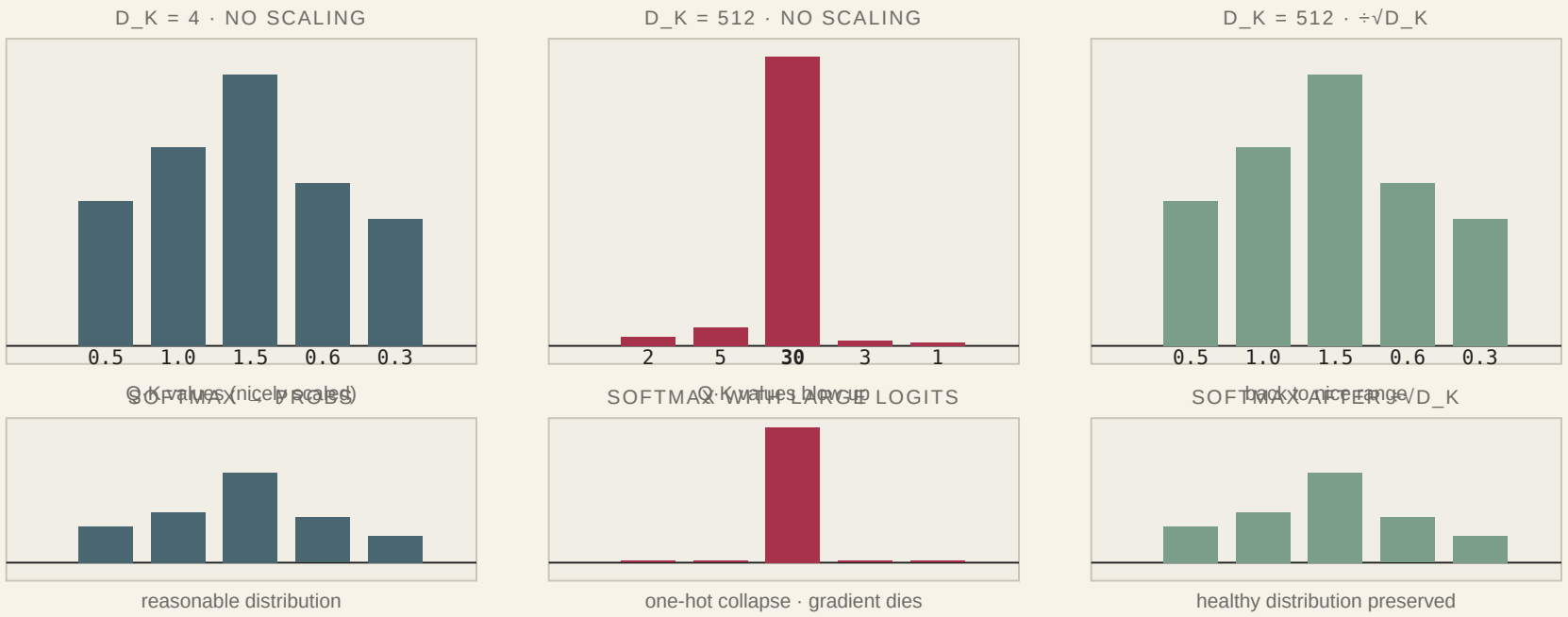
Raw logits $[s_1, s_2, s_3] = [2.0, 1.0, 0.5]$. Softmax behaves nicely:

TEMPERATURE	SOFTMAX
/1	(0.58, 0.21, 0.13, …) — soft
/4	(0.40, 0.30, 0.26, …) — very soft
×10	(0.9999, 4e-5, 2e-7) — one-hot

Dot products without scaling behave like the bottom row — **effectively one-hot**. Divide by $\sqrt{d_k}$ and you land back on the top row. The scaling is doing exactly the role of a temperature denominator, derived from variance analysis rather than tuned by hand.

In pictures

Why divide by √d_k — without it, softmax collapses to one-hot



Why one-hot is bad

If softmax outputs are (nearly) one-hot, then attention picks **one** encoder state and ignores the rest.

Two consequences:

1. Information from other positions is discarded.
2. Gradient through the softmax is near zero (saturation) → training stalls.

The fix — divide scores by $\sqrt{d_k}$:

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

Scores stay in a healthy range → softmax stays soft → gradients keep flowing.

⚠️

optional · The $\sqrt{d_k}$ derivation in three lines

Assume Q, K entries are i.i.d. with zero mean and unit variance.

$$\mathbb{E}[Q^\top K] = 0, \quad \mathrm{Var}[Q^\top K] = \sum_{k=1}^{d_k} \mathrm{Var}[Q_k K_k] = d_k$$

So $Q^\top K \sim \mathcal{N}(0, d_k)$. Dividing by $\sqrt{d_k}$ makes the variance **1** — independent of dimension.

KEY IDEA

Why this matters · the same attention block can be used at $d_k = 64$ or $d_k = 4096$ without retuning temperatures. The scaling is *dimension-invariant* by construction.

PART 5

Self-attention vs cross-attention

Same machinery · different sources for QKV

Cross-attention · the decoder reads the encoder

In a Seq2Seq + attention model:

- Queries come from the **decoder** (current target position).
- Keys and values come from the **encoder** (all source positions).

  This is what the first attention heatmap showed — one distribution per target step over source positions.

Self-attention · the "bank" disambiguation

KEY IDEA

How do you know "bank" means a financial institution in "the **bank** approved the **loan**"?

You look at the other words. "Loan" tells you which bank.

In "he sat on the river **bank**" · the word "river" tells you the *other* meaning.

Self-attention is exactly this · every word looks at every *other* word in the sentence to figure out its **context-specific meaning**. The famous "the animal didn't cross the street because **it** was too tired" example needs this · attention disambiguates "it" by attending to "animal."

Self-attention · a sequence attends to itself

Same operation. Now:

$$Q = X\,W_Q, \quad K = X\,W_K, \quad V = X\,W_V$$

All three come from **the same input sequence**.

KEY IDEA

Self-attention lets every position in a sequence aggregate information from every other position — **in parallel**, with no recurrence.

This is the idea that killed RNNs and gave us Transformers (L13).

Self-attention · 3 tokens, by hand

Sentence · "the cat slept" token embeddings $x_1, x_2, x_3 \in \mathbb{R}^d$.

After projecting: Q, K, V are each a $3 \times d_k$ matrix.

DERIVATION

$$QK^\top / \sqrt{d_k} = \begin{bmatrix} s_{11} & s_{12} & s_{13} \\ s_{21} & s_{22} & s_{23} \\ s_{31} & s_{32} & s_{33} \end{bmatrix}$$

Row i says: *how much does token i want to look at every other token?*

Softmax per row → 3×3 attention matrix A. Output AV is again $3 \times d_k$.

Every output row is a weighted blend of value rows — a contextualized embedding for that token.

Self-attention vs convolution · same goal, different bias

Convolution

- Each output depends on a **fixed** local window.
- Inductive bias: *locality*.
- Parameters: shared kernel.

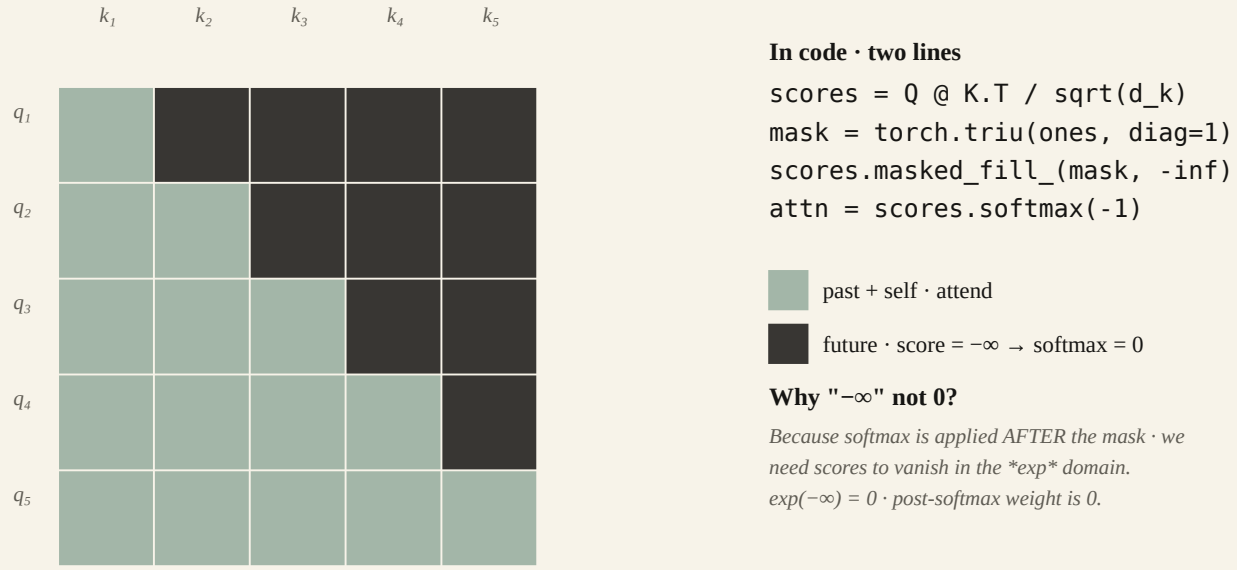
Self-attention

- Each output depends on **all** positions.
- Inductive bias: *learned* from data.
- Parameters: Q, K, V projections.

INTUITION

Convolution bakes in "nearby tokens matter"; self-attention lets the network decide from data whether nearby or far-away tokens matter. When you have lots of data, *learned* bias wins over *hand-designed* bias. That's the whole arc of the 2017–2025 vision revolution in one sentence.

Causal mask · 5×5 grid



In code · two lines

```
scores = Q @ K.T / sqrt(d_k)
mask = torch.triu(ones, diag=1)
scores.masked_fill_(mask, -inf)
attn = scores.softmax(-1)
```

past + self · attend

future · score = −∞ → softmax = 0

Why "−∞" not 0?

*Because softmax is applied AFTER the mask · we need scores to vanish in the *exp* domain. exp(−∞) = 0 · post-softmax weight is 0.*

Causal self-attention · don't peek at the future

INTUITION

When writing the next word of "The quick brown fox jumps...", you can only use what you've already written. A **causal mask** is like covering the future with cardboard — for "fox", you see "The quick brown fox" but everything after is hidden.

```
scores = Q @ K.transpose(-2, -1) / math.sqrt(d_k)     # (n, n)
mask   = torch.triu(torch.ones_like(scores), diagonal=1).bool()
scores.masked_fill_(mask, float('-inf'))              # future → -inf
weights = scores.softmax(dim=-1)                      # rows still sum to 1
```

That's the only difference between BERT-style (bidirectional) and GPT-style (causal) attention. Same module, different mask.

Worked numeric · how the $-\infty$ mask works

Raw scores (3 tokens, $d_k = 1 \rightarrow$ no scaling):

$$S = \begin{pmatrix} 2.1 & 1.5 & 0.4 \\ 0.8 & 3.1 & 1.2 \\ 1.4 & 0.7 & 2.5 \end{pmatrix}$$

Step 1 · mask the upper triangle.

$$S' = \begin{pmatrix} 2.1 & -\infty & -\infty \\ 0.8 & 3.1 & -\infty \\ 1.4 & 0.7 & 2.5 \end{pmatrix}$$

Step 2 · row-wise softmax. Since $\exp(-\infty) = 0$:

- Row 1: $\mathrm{softmax}([2.1, -\infty, -\infty]) = [1.0,\ 0,\ 0]$
- Row 2: $\mathrm{softmax}([0.8, 3.1, -\infty]) \approx [0.09,\ 0.91,\ 0]$
- Row 3: $\mathrm{softmax}([1.4, 0.7, 2.5]) \approx [0.21,\ 0.10,\ 0.69]$

The $-\infty$ guarantees future tokens get **exact zero** weight after softmax. Token 1 sees only itself; token 2 sees 1–2; token 3 sees all.

Complexity · the $O(n^2)$ wall

Self-attention on a sequence of length n:

- $QK^\top$ builds an $n \times n$ matrix → $O(n^2)$ **memory** and $O(n^2 d)$ **compute**.
- Double the context → 4× the cost.

WATCH OUT

At $n = 8{,}192$ and $d = 4096$, one head's attention matrix is already **64 MB per layer**. Scaling context to 1M tokens naively would need 500 GB per layer. This is the wall that motivates:

- **FlashAttention** (L23) · recompute attention in tiles, avoiding the full $n \times n$ matrix.
- **Sparse / local / linear attention** (reading) · trade off quality for $O(n \log n)$ or $O(n)$.
- **KV caching** (L23) · don't redo the whole computation at every generation step.

The four kinds of attention you will meet

TYPE	Q COMES FROM	K, V COME FROM	USED IN
Encoder self-attn	encoder	encoder	Transformer encoder
Decoder self-attn (causal)	decoder (past only)	decoder (past only)	GPT, decoder of Transformer
Cross-attention	decoder	encoder	Seq2Seq decoder, translation
Masked/local attention	input	input (masked)	Longformer, Reformer, etc.

PART 6

What attention unlocked

One slide of consequences

Why attention was such a big deal

1. **Bottleneck solved** · no more "fit everything into 512 dims."
2. **Long-range dependencies** · every target step can see any source step.
3. **Parallelizable** · unlike RNNs, all attention scores can be computed at once.
4. **Interpretable** · attention heatmaps are the first DL visualization tool that's actually informative.
5. **Transfer** · attention blocks compose cleanly — stack them, mix them with cross-attention, make them multi-headed. The Transformer (L13) is exactly this.

> IN PRACTICE
>
> Without attention · no Transformer · no BERT · no GPT · no Claude · no diffusion text conditioning. A single 2014 paper seeded the next decade.

"Attention Is All You Need" · the 2017 pivot

Vaswani et al.'s one-page insight · drop the RNN entirely, use **only** attention plus FFNs plus positional encodings.

> REFERENCE
>
> Before: encoders and decoders were RNNs *with* attention as a helper. After: attention was the load-bearing operation; RNN was gone.

Consequence table:

AXIS	RNN+ATTENTION	TRANSFORMER
Sequential compute	yes (unrolled)	no (parallel)
Long-range path	$O(n)$ hops	$O(1)$ hops
Training throughput	slow	10–20× faster
Scaling	plateaus at ~1B	trained to 1T+

Every major model since 2018 (BERT, GPT-*, T5, Claude, Llama) is this architecture, plus or minus details. **L13 builds it from parts.**

Summary · Lecture 12 — summary

- **Attention** = soft retrieval · each query selects a weighted combination of values based on similarity to keys.
- **Bahdanau** (additive, 2014) and **Luong** (multiplicative, 2015) are two parameterizations — we use Luong's dot-product form in Transformers.
- **QKV abstraction** · Q, K, V are learned projections of the same input; the network decides what each role should be.
- $\sqrt{d_k}$ **scaling** — without it, softmax collapses to one-hot at large d_k and gradients die.
- **Self-attention** · Q, K, V from the same sequence; parallel, long-range, interpretable.
- **This unlocked the Transformer** — next lecture.

Read before Lecture 13

Prince Ch 12 mid-sections (Transformer block).

Next lecture

The Transformer — built live. Multi-head attention · positional encoding · residual + LayerNorm · the full encoder-decoder stack.

NOTEBOOK

Notebook 12 · `12-attention-nmt.ipynb` — add attention to Lecture 11's Seq2Seq; visualize attention heatmaps; watch BLEU improve on long sentences.