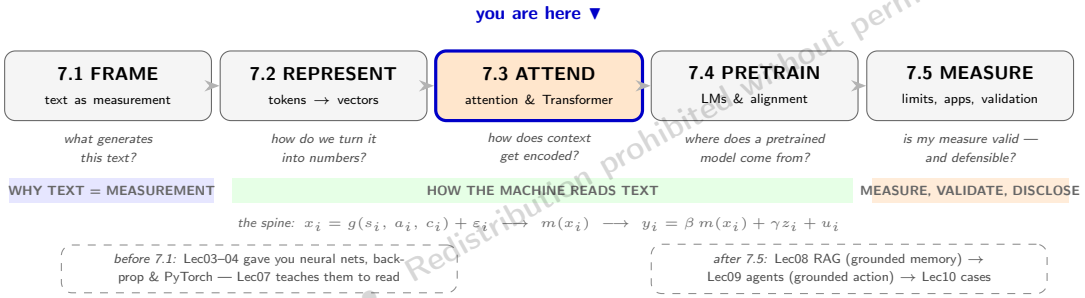
AI for Economic Research: Dynamic Models, Language, and Agents
Lecture 7.3a: The Attention Mechanism

Zhigang Feng

2026-07-09 06:15:34

Lecture 7 in One Map



Route: one worked example → the goal (a context vector per token) → the approach (Q/K/V) → where the matrices come from (the training objective) → the mechanics, step by step → multi-head; part 2/2 (T3b) wraps the head into the full Transformer layer.

The Goal, Through One Example

One ambiguous word — and the context-aware vector we want for it

Start Here: One Word, One Problem

A sentence from a policy report: "The central bank raised interest rates amid rising inflation."

Tokenization (9 word tokens, numbered left to right):

$$\underbrace{\text{The}}_{1}\ \underbrace{\text{central}}_{2}\ \underbrace{\text{bank}}_{3}\ \underbrace{\text{raised}}_{4}\ \underbrace{\text{interest}}_{5}\ \underbrace{\text{rates}}_{6}\ \underbrace{\text{amid}}_{7}\ \underbrace{\text{rising}}_{8}\ \underbrace{\text{inflation}}_{9}$$

- **The problem.** In isolation, *interest* (position 5) is ambiguous — monetary-policy *rates*, personal *curiosity*, or a financial *stake*? We want a vector $e'^{(5)}$ encoding how it is used *in this sentence*.
- **What a reader does.** Seeing *interest*, you ask "interest in *what*?" — your eye jumps to *rates* and *inflation*, the words that fix its meaning.
- **What attention does (previewed).** Exactly that, numerically. Of all the attention *interest* pays across the sentence:

  $$\text{attention on } \textit{rates} \approx 0.50, \quad \text{on } \textit{inflation} \approx 0.40, \quad \text{on all else} \approx 0.10,$$

  then it *blends their content* into $e'^{(5)}$ — landing it squarely in monetary-policy territory.

Those weights were never hand-coded — no dictionary said "interest goes with rates." How the model *finds* them, and what $0.50/0.40$ computes, is the rest of this deck.

What We're Trying to Achieve

- Zoom out from one word. A Transformer is a function

$$\mathcal{T}\colon \left(\mathbb{R}^d\right)^n \longrightarrow \left(\mathbb{R}^d\right)^n, \quad \left[e^{(1)}, \ldots, e^{(n)}\right] \mapsto \left[e'^{(1)}, \ldots, e'^{(n)}\right],$$

  mapping n input vectors to n contextualized outputs of the same dimension. (Inputs are token embeddings + positional encodings; details in T3b.)

- **The single conceptual jump: static → contextual.** Word2Vec gave *one vector per word type* — *bank* in "river bank" and "central bank" share it. A Transformer gives *one vector per token occurrence*: the two *bank*s separate. Our $e'^{(5)}$ for *interest* is exactly such a vector.

- **What "contextualized" packs in.** Each $e'^{(i)}$ encodes at once (a) the identity of $t^{(i)}$, (b) its role in *this* sequence, (c) a weighted blend of every other token's contribution.

- **Not the final answer.** These are an *intermediate representation*. A small *task head* sits on top: project $e'^{(n)}$ onto the vocabulary (generation) or $e'^{([\mathrm{CLS}])}$ onto labels (classification).

The Approach: Query, Key, Value

Score with queries and keys, read with values — a soft database lookup

The Approach: Score, Normalize, Blend

- For each position i, ask: *"which other positions are most relevant to i?"* Then build i's new representation as a *weighted sum* over all positions:

$$\tilde{e}^{(i)} \;=\; \sum_j \alpha_j^{(i)}\,(\text{content of } j), \qquad \alpha_j^{(i)} \geq 0, \quad \sum_j \alpha_j^{(i)} = 1.$$

  The weights $\alpha^{(5)} = (\ldots, 0.50, \ldots, 0.40, \ldots)$ from the previous slide are exactly these.

- **Crucial properties:**
  - Every token attends to every other in *one* step — no recurrence.
  - Weights are computed *dynamically* from the input, not fixed.
  - All positions are processed in *parallel* $\Rightarrow$ trains efficiently on GPUs.

- Repeat this "everyone looks at everyone" operation L times (e.g. $L = 96$ in GPT-3) with different learned weights, and the model builds deeply contextualized representations.

- **Engineering name:** scaled dot-product self-attention. Three questions remain — *what* gets scored, *how*, and *where the weights come from*. The answer to the first is Q, K, V.

The Three Roles — Query, Key, Value

- To score and blend, each token computes *three* projections of its embedding $e^{(i)} \in \mathbb{R}^d$:

$$\begin{aligned} Q_i &= W_Q\, e^{(i)} && \text{(query)} \\ K_i &= W_K\, e^{(i)} && \text{(key)} \\ V_i &= W_V\, e^{(i)} && \text{(value)} \end{aligned}$$

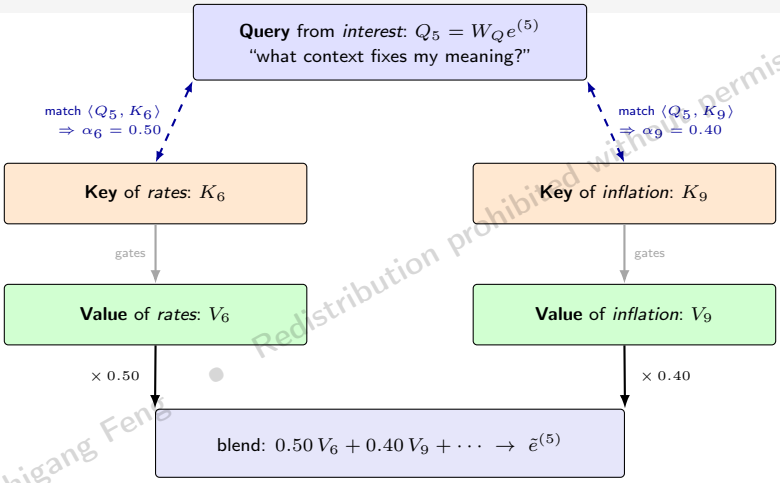
  where $W_Q, W_K, W_V \in \mathbb{R}^{k \times d}$ are learned matrices and typically $k \leq d$.

- **The three roles** (information-retrieval mnemonic):
  - Q_i — the *question* token i asks: "what context do I need?"
  - K_j — the *advertisement* token j posts: "here's what I'm about."
  - V_j — the *content* token j delivers if it is matched.

- **Why three separate projections?** Separation enables *asymmetric* attention: *interest* can ask a question (Q_5) that matches *rates*'s advertisement (K_6) without *rates* having to ask the same question back. A single shared vector could not express "i needs j but not vice versa."

- The match $\langle Q_i, K_j \rangle$ sets *how much* to read; the value V_j is *what* gets read. The next slide shows this split visually.

Q/K/V at a Glance — Match Sets the Weight, the Weight Reads the Value



Three roles, kept separate. The *query–key* match (dashed) only decides *how much* to read; what gets read is the *value*. Asymmetry matters: *interest* queries *rates* without *rates* querying back. W_Q, W_K, W_V are all learned from the loss — nobody hand-codes "attend to rates."

The Retrieval Analogy: A Soft Database Lookup

Attention is a search engine that returns a weighted blend instead of one hit.

Retrieval system	Self-attention	At position i
search query	query vector	$Q_i = W_Q e^{(i)}$: "what do I need?"
document tags / index	key vectors	$K_j = W_K e^{(j)}$: "what I am about"
document contents	value vectors	$V_j = W_V e^{(j)}$: what is actually read
relevance ranking	scaled dot products	$\langle Q_i, K_j \rangle / \sqrt{k}$
click the top result	softmax blend	read *all* results, weighted

- Every position plays all three roles at once: it queries the others, advertises itself, and delivers content when matched.
- The "soft" part — a convex combination rather than a hard top-1 pick — keeps the whole lookup differentiable, so it can be *learned* end-to-end by gradient descent.
- Preview: the same query–key logic powers document retrieval in RAG (Lec08) — there the "keys" are your corpus.

Where Do Q, K, V Come From?

The projection matrices are learned — by one global objective

Where the Projection Matrices Come From

The $0.50/0.40$ pattern is not in the code — it is learned. Here is the objective that produces it.

- **What is trainable (θ).** In every layer W_Q, W_K, W_V, W_O, plus the FFN, LayerNorm, token-embedding ($\mathcal{E}$) and output-head (W_{vocab}) weights. All start *random*, updated by gradient descent on a **single global loss**.
- **The objective** (GPT-style autoregressive LM) — predict the next token from the past:

$$\mathcal{L}(\theta) \;=\; -\sum_{s}\sum_{i} \log P_\theta\big(t_s^{(i+1)} \mid t_s^{(1:i)}\big).$$

  (BERT-style *masked* LM predicts hidden tokens from both sides; fine-tuning swaps in a task label.)
- **How that shapes W_Q, W_K, W_V.** One backward pass sends gradients through softmax, the dot products, *and* the projections together. They move so that $\langle Q_i, K_j \rangle$ is *high exactly where attending $i \to j$ lowers the loss*. Nobody codes "attend to neighbours" — the model *discovers* which pairings help predict text.
- **Back to interest.** Reading it against *rates* and *inflation* helped predict the surrounding words of millions of policy sentences — *that* payoff is why W_Q, W_K, W_V end up producing $\alpha_6 \approx 0.50,\ \alpha_9 \approx 0.40$. Nobody wrote the rule.

The Mechanics, Step by Step

We have Q, K, V — now score, normalize, blend, and project back

Mechanics I — Scaled Dot-Product Scores

We already have Q, K, V. Now score every candidate position against interest's query.

- Query–key compatibility for pair (i, j) is the scaled dot product $s_j^{(i)} = \langle Q_i, K_j \rangle / \sqrt{k}$.
- **Why divide by $\sqrt{k}$?** Larger k inflates dot products, pushing softmax into saturation (one term dominates, gradients vanish); dividing by $\sqrt{k}$ holds $\mathrm{Var}\big(s_j^{(i)}\big) \approx 1$.

Worked example, $i = 5$ (*interest*) — the nine scaled scores:

token j	rates$_6$	infl.$_9$	raised$_4$	int.$_5$	bank$_3$	rising$_8$	central$_2$	amid$_7$	the$_1$
$s_j^{(5)}$	2.3	2.1	-0.8	-1.0	-1.1	-1.2	-1.4	-1.6	-1.8

Only the two policy words score high; function and distant words score low.

- **Direction.** This illustration is *bidirectional* (encoder-style: every token may see every other), so *interest* can look ahead to *rates* / *inflation*. A generator would instead apply a **causal mask** — next slide.

Mechanics II — Softmax to Attention Weights

- Convert the scores into a probability distribution over positions:

$$\alpha_j^{(i)} = \frac{\exp\big(s_j^{(i)}\big)}{\sum_{r=1}^{n} \exp\big(s_r^{(i)}\big)}, \qquad \alpha_j^{(i)} \geq 0, \;\; \sum_j \alpha_j^{(i)} = 1.$$

- **Our example.** Exponentiate the nine scores and normalize (try it: $e^{2.3}/\sum_r e^{s_r} \approx 9.97/20.2 \approx 0.49$):

$$\alpha_6^{(5)} \approx 0.50 \text{ (\textit{rates})}, \qquad \alpha_9^{(5)} \approx 0.40 \text{ (\textit{inflation})}, \qquad \sum_{\text{other 7}} \alpha_j^{(5)} \approx 0.10.$$

  This is the $0.50/0.40$ split previewed on slide 1 — now *derived*, not asserted.

- **Why softmax?** It turns any real score vector into a *convex* weighting (each ≥ 0, summing to 1) and is smooth $\Rightarrow$ differentiable end-to-end. Concentrated $\alpha \Rightarrow$ "hard" attention; diffuse $\alpha \Rightarrow$ "soft."

- **Causal masking (decoder variant).** A generator masks $s_j^{(i)} = -\infty$ for $j > i$ before softmax (no peeking ahead) — that would hide positions 6–9 from *interest*. Our bidirectional (encoder) reading keeps them.

Mechanics III — Weighted Sum of Values

- Given the weights $\alpha_j^{(i)}$, the attention output for position i is the weighted sum of *value* vectors:

$$\text{AttnVec}_i \;=\; \sum_{j=1}^{n} \alpha_j^{(i)}\, V_j \;\in\; \mathbb{R}^k.$$

- **Our example:**

$$\text{AttnVec}_5 \;\approx\; 0.50\, V_6 \;+\; 0.40\, V_9 \;+\; 0.10 \times (\text{the rest}).$$

interest "reads" mostly *rates*'s and *inflation*'s content. Recall the split of labour: K_j decided *how much*; V_j is *what* is delivered.

- **Why a blend, not a hard top-1 pick?** The output mixes $V_{\text{rates}}, V_{\text{inflation}}, \ldots$ into one vector reflecting all relevant contexts at once — richer than any single lookup, and differentiable so it can be learned.

- **Contextualization, made concrete.** Swap the sentence to "The discount *rates* applied to pension funds were revised" and the *same* machinery shifts its reading: *discount* and *pension* become the high-weight sources, and the vector lands in discounting / pensions territory instead. Same machinery, different sentence, different meaning.

Mechanics IV — Project Back, and the Full Recipe

- AttnVec lives in $\mathbb{R}^k$; the next layer needs $\mathbb{R}^d$. Project back with a learned $W_O \in \mathbb{R}^{d\times k}$:

$$\tilde{e}^{(i)} \;=\; W_O\,\text{AttnVec}_i.$$

  For *interest*, $\tilde{e}^{(5)}$ is the context-aware vector we set out to build on the very first slide.

- **Why project back?** In multi-head attention each head works in a smaller subspace $\mathbb{R}^k$ ($k = d/H$); W_O restores the common dimension d after concatenating heads, so the result flows into the next layer with no shape change.

- **The full recipe (one head):**
  1. **Project:** $Q_i, K_i, V_i = W_Q e^{(i)},\ W_K e^{(i)},\ W_V e^{(i)}$
  2. **Score:** $s_j^{(i)} = \langle Q_i, K_j\rangle/\sqrt{k}$
  3. **Softmax:** $\alpha_j^{(i)}$
  4. **Blend:** $\text{AttnVec}_i = \sum_j \alpha_j^{(i)} V_j$
  5. **Project back:** $\tilde{e}^{(i)} = W_O\,\text{AttnVec}_i$

- All four matrices W_Q, W_K, W_V, W_O are learned by backpropagation from the task loss (previous section) — and the whole computation runs in parallel for every position i at once.

Multi-Head Attention

- One (Q, K, V) trio captures only one "view" of the sequence. **Multi-head attention** runs H such trios in parallel and concatenates their outputs.
- Each head h has its own projection matrices $W_Q^{(h)}, W_K^{(h)}, W_V^{(h)}$, all of dimension $k = d/H$.
- After running H attention computations independently, concatenate and project:

$$\text{MultiHead}_i \;=\; W_O^{*}\Big[\,\text{AttnVec}_i^{(1)};\ldots;\,\text{AttnVec}_i^{(H)}\Big].$$

- **Do heads have different objectives?** *No — all H heads share the same training loss* (e.g. next-token cross-entropy). What differs is their random initialization. Specialization is **emergent**: under one global loss, H parallel feature detectors diversify because identical copies of the same head would be redundant.
- **Observed specializations in financial text:**
  - Head 1: monetary-policy relations (*inflation* ↔ *interest*)
  - Head 2: temporal phrases (*Q1*, *next month*, *by year-end*)
  - Head 3: hawkish vs. dovish stance markers

  Nobody told the heads what to look for — they were not assigned separate objectives.
- Total parameter cost is essentially the same as a single-head attention with the full dimension d.

Thinkbox: Attention Entropy as a Measurement Diagnostic

The weight distribution itself carries information about measurement quality.

- Position i's weights $\boldsymbol{\alpha}^{(i)}$ form a probability distribution; its Shannon entropy $H(\boldsymbol{\alpha}^{(i)}) = -\sum_j \alpha_j^{(i)} \log \alpha_j^{(i)}$ measures how *focused* the reading is.
- $H \approx 0$: meaning pinned down by a few strong context words — 2022-style tightening statements, where *inflation* attends to *raised*, *persistent*, *mandate*.
- $H \approx \log n$: attention spread thin — the contextual vector approaches a global average. 2019-style "crosscurrents" statements: *inflation* attends diffusely to *employment*, *growth*, *uncertainty*, ...

High-entropy passages produce noisier stance scores — measurement-error variance that *moves with the state of the economy*. If you regress on Transformer-based stance measures, attention entropy is a candidate flag for heteroskedastic measurement error. A PBOC wrinkle to discuss: boilerplate political formulas share every passage with the substantive policy words — does that push the attention entropy of terms like "inflation" systematically higher in PBOC reports than in FOMC statements?

Caveat: attention weights are intermediate computations, not structural parameters — diagnostics and interpretation aids, never "effects."

Bridge: One Head Is a Building Block

We can now compute a context-aware vector — once.

- One attention pass mixes information *across* positions. On its own it is still linear in the values, blind to word order, and single-pass.
- **T3b wraps the head into a layer:** positional encodings (order), residual connections + LayerNorm (trainability at depth), a feed-forward network (nonlinearity — and, it turns out, the model's stored knowledge) — then stacks the layer L times and chooses among three architectural variants, ending with a full research case on 10-K filings.

Arc: T1 FRAME → T2 REPRESENT → **T3 ATTEND** *→ T4 PRETRAIN → T5 MEASURE.*