Attention in Transformers, Step-by-Step

3Blue1Brown — Deep Learning, Chapter 6 Watch on YouTube

I. Recap: Embeddings & the Transformer's Goal

- The model takes in text and predicts the next word
- Input is broken into tokens, each associated with a high-dimensional embedding vector
- Directions in embedding space correspond to semantic meaning (e.g., gender)
- The transformer's job is to progressively adjust these embeddings to encode richer contextual meaning, not just individual words

II. Motivating Examples

- **"Mole"** — same word, three different meanings (animal, chemistry unit, skin growth); the initial embedding is identical in all cases, so context must refine it
- **"Tower"** — preceded by "Eiffel" or "miniature Eiffel," the embedding should shift toward more specific meanings
- **The mystery novel thought experiment:** the final vector in a long sequence must absorb information from the entire passage to predict "the murderer was ___"

III. The Attention Pattern (Queries & Keys)

- Running example: *"A fluffy blue creature roamed the verdant forest"* — adjectives should update their corresponding nouns
- **Query vectors (Q):** each token produces a query via a learned query matrix (W_Q); conceptually, nouns "ask" for relevant adjectives
- **Key vectors (K):** each token produces a key via a learned key matrix (W_K); adjectives "answer" the nouns' queries
- **Dot products:** keys and queries are compared via dot products to score relevance
- **Softmax normalization:** scores are normalized column-wise into a probability distribution
- **The attention pattern:** the resulting grid of weights, matching the compact formula from the original paper ($Q \cdot K^T / \sqrt{d_k}$, then softmax)

IV. Masking

- During training, the model predicts every next token simultaneously for efficiency
- Later tokens must not influence earlier ones (no "peeking" at the answer)

- Achieved by setting upper-triangle entries to −∞ before softmax, so they become 0 after normalization

V. Context Size

- The attention pattern is a square grid (context size × context size), so compute scales quadratically
- This is a major bottleneck; recent research explores ways to make it more scalable

VI. Values — Updating the Embeddings

- **Value vectors (V):** a third learned matrix maps each embedding to a value vector — "what information should this word pass along if it's relevant?"
- Weighted sums of value vectors (weighted by the attention pattern) produce a change (Δe) that is added to each token's embedding
- Result: refined embeddings that encode contextual meaning (e.g., "creature" → "fluffy blue creature")

VII. Counting Parameters

- Key and query matrices: ~1.5M parameters each (12,288 × 128)
- The value map is factored into two smaller matrices ("value down" and "value up") to keep parameter count efficient — a low-rank factorization
- Total per attention head: ~6.3 million parameters (using GPT-3 dimensions)

VIII. Cross-Attention (Side Note)

- Everything described so far is *self*-attention (one sequence attending to itself)
- *Cross*-attention: keys come from one data source, queries from another (e.g., translation, speech transcription); typically no masking

IX. Multi-Headed Attention

- A single attention head captures one type of contextual relationship
- Multi-headed attention runs many heads in parallel (GPT-3 uses 96 per block), each with its own Q, K, and V matrices
- Each head proposes a change; all changes are summed and added to the original embedding
- This gives the model capacity to learn many distinct types of contextual updating simultaneously

X. The Output Matrix

- In practice, the "value up" matrices from all heads are stapled together into one large *output matrix* for the entire block
- What papers call the "value matrix" per head is typically only the "value down" projection

XI. Going Deeper — Stacking Layers

- Transformers repeat attention blocks and MLP blocks many times (96 layers in GPT-3)
- Deeper layers encode increasingly abstract ideas: sentiment, tone, scientific relevance, etc.
- Total attention parameters across all layers: ~58 billion — about one-third of GPT-3's 175 billion total

XII. Closing: Why Attention Works

- A key advantage is *parallelizability* — massive computations run efficiently on GPUs
- Scale alone has driven huge qualitative improvements, and attention's architecture enables that scale
- Recommended further resources: Andrej Karpathy, Chris Olah, Vivek's history videos, and The Art of the Problem