Lecture 4: Attention Mechanism and Transformers - I

COMP 5801H/4900A: Generative AI and LLMs

2026-01-15

Sriram Subramanian
Assistant Professor & Canada Research Chair, Carleton University
Faculty Affiliate, Vector Institute for Artificial Intelligence
Faculty Affiliate, Schwartz Reisman Institute for Technology and Society

Carleton University

VECTOR INSTITUTE

SCHWARTZ REISMAN INSTITUTE FOR TECHNOLOGY AND SOCIETY

Lecture Outline

- **The Bottleneck Problem**
  - Why recurrence fails for long-range dependencies
  - The "Single Sticky Note" problem (Fixed-size context vectors)
- **Neural Machine Translation (NMT) with Attention**
  - The Bahdanau breakthrough: Scoring and Alignment
  - Query (Q), Key (K), and Value (V) analogies
- **From RNNs to Self-Attention**
  - The "Attention is All You Need" philosophy
  - Parallelization vs. Sequential Processing
- **The Transformer Architecture**
  - Positional Encoding, Multi-Head Attention, and Layer Norm
- **Scalability and the Generative Era**
  - How Attention enabled the rise of Large Language Models (LLMs)

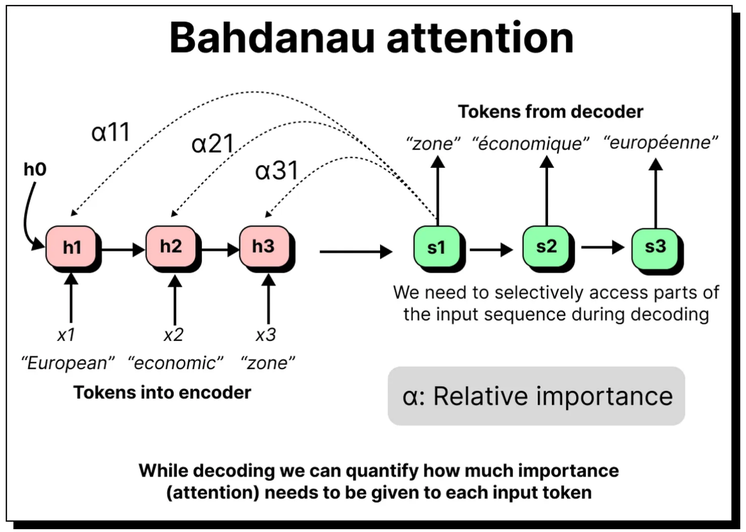
Information/Computation Bottleneck

- **Sequential Compression**
  - Entire input sequence $(x_1, x_2, \ldots, x_n)$ must be compressed into a **single, fixed-length hidden state** (h_n)
  - Final hidden state has **fixed capacity**
  - Information Decay: Recent words dominate the hidden state
- **Long-Range Dependency (Distance vs Signal problem)**
- **The Computational Bottleneck**
  - Serial Constraint: Cannot compute h_t until h_{t-1} is finished
  - GPU Underutilization: Cannot use parallel matrix multiplication features

The Bahdanau Breakthrough (2014)

- **Content-Based Addressing**: Instead of a single context vector (c), the model calculates a **different context vector** (c_t) for every single generated word
- At each step, the model "looks" at all hidden states of the encoder ($h_1, \ldots, h_n$) and decides which ones are relevant
- The model learns to align words across languages (e.g., learning that the 4th word in English might "align" with the 2nd word in French)

The Bahdanau Breakthrough (2014): Architecture



Credit: https://www.vizuaranewsletter.com/p/from-rnns-to-transformers-the-evolution

- Each input token is encoded into a hidden state
- Decoder generates a token by computing **alignment scores between** s_i **and** $(h_0, \ldots, h_t)$
- Decoder forms a context vector as a weighted sum of the encoder states, guided by the attention scores
- This context vector, combined with the decoder's hidden state, produces the next output token

The Bahdanau Breakthrough (2014): Math

- **A"Score" Function** is used to decide how much attention to pay to hidden (encoder) state h_i, we compare it with the previous decoder state s_{t-1}
- Alignment Score: $e_{t,i} = a(s_{t-1}, h_i) = v_a^\top \tanh(W_a s_{i-1} + U_a h_j)$
- The score is calculated using a small feed-forward neural network that is trained jointly with the rest of the model

- This "score" is passed through a Softmax function to create the Attention Weights ($\alpha_{t,i}$), which sum to 1
- Context Vector: $c_t = \sum \alpha_{t,i} h_i$

The Bahdanau Breakthrough (2014): Math - II

- In the Bahdanau architecture, the "next" hidden state of the decoder (s_i) is calculated using the **context vector, the previous state, and the previously generated word**

$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$

- Where:
  - s_{i-1}: The previous decoder hidden state
  - y_{i-1}: The word the model just predicted (or the ground truth word during training)
  - c_i: The context vector we just calculated
  - f: Usually a GRU or LSTM cell

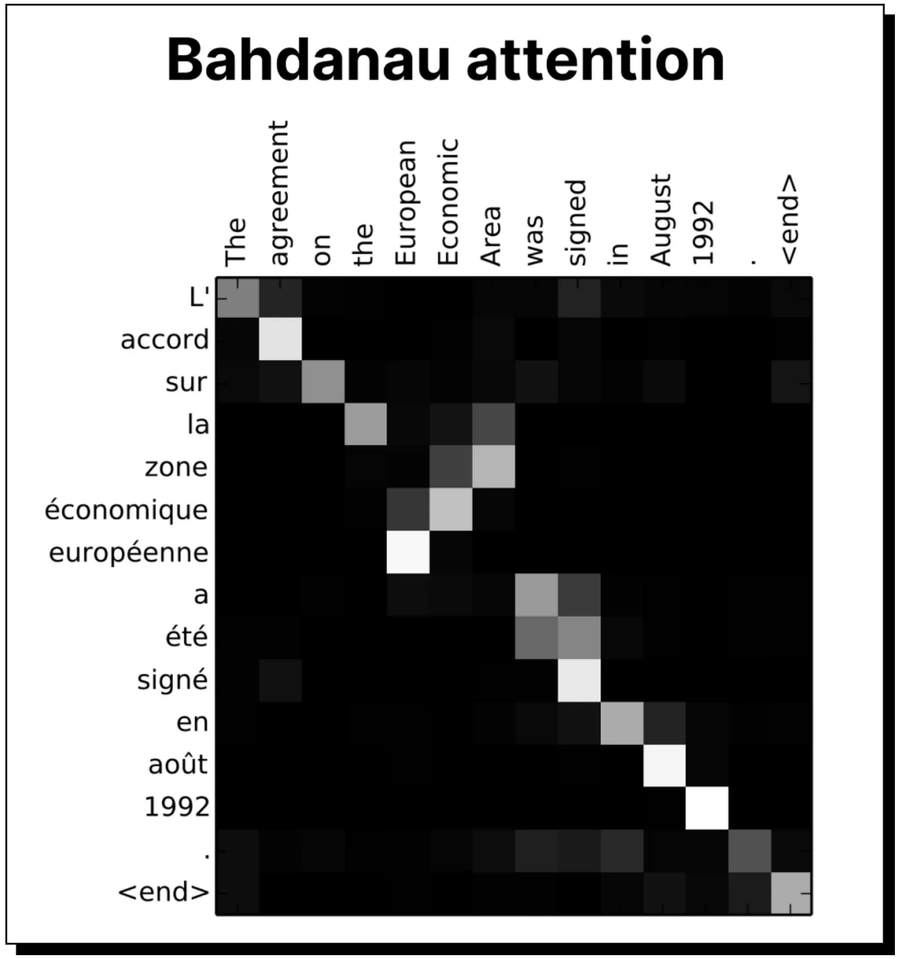
The Bahdanau Breakthrough (2014): Math - III

- Once we have the new hidden state s_i, **we pass it through a linear layer** and **a Softmax** to get the probability of every word in our vocabulary:

$$P(y_i \mid y_1, \ldots, y_{i-1}, X) = \text{softmax}(g(s_i, y_{i-1}, c_i))$$

- Typically, g is a simple feed-forward network

Visualizing the Alignment (Interpretability)



Credit: https://www.vizuaranewsletter.com/p/from-rnns-to-transformers-the-evolution

Question: In Bahdanau attention, we calculate the similarity between the decoder state and every encoder state. If our input sentence is 1,000 words long, how many 'similarity scores' must we calculate to produce just a single translated word? Does this solve our computational bottleneck?

QKV Analogy (Search Engine)

To understand attention, think of it as a **retrieval system** similar to a library or a search engine

- **Query (Q)**: What you are looking for? (e.g., "What is the subject of this sentence?”)
- **Key (K)**: The labels/tags on the information (e.g., "I am a noun," "I am a verb.”)
- **Value (V)**: The actual information contained in the word (e.g., "The meaning of 'Apple'.")

QKV Analogy: Mathematical Interaction

- **Match ($Q \cdot K$)**: Compare your **Query** against all possible **Keys** to see how well they match
- **Filter (Softmax)**: Normalize these matches into a set of weights that sum to 1
- **Retrieve (Weight $\cdot$ V)**: Extract the information (Values) from the items that had the best match

Scaled Dot-Product Attention

- The entire process above is expressed in one elegant matrix equation:

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

- Why the $\sqrt{d_k}$? (The "Scaled" part)
  - As the dimension (d_k) of the vectors grows, the magnitude of the dot product (QK^T) can grow very large
  - Leads to **numerical instability** due to floating-point precision limitations
  - Dividing by the square root of the dimension "scales" the scores back down, ensuring stable training

Why is QKV better than Bahdanau? Intuition

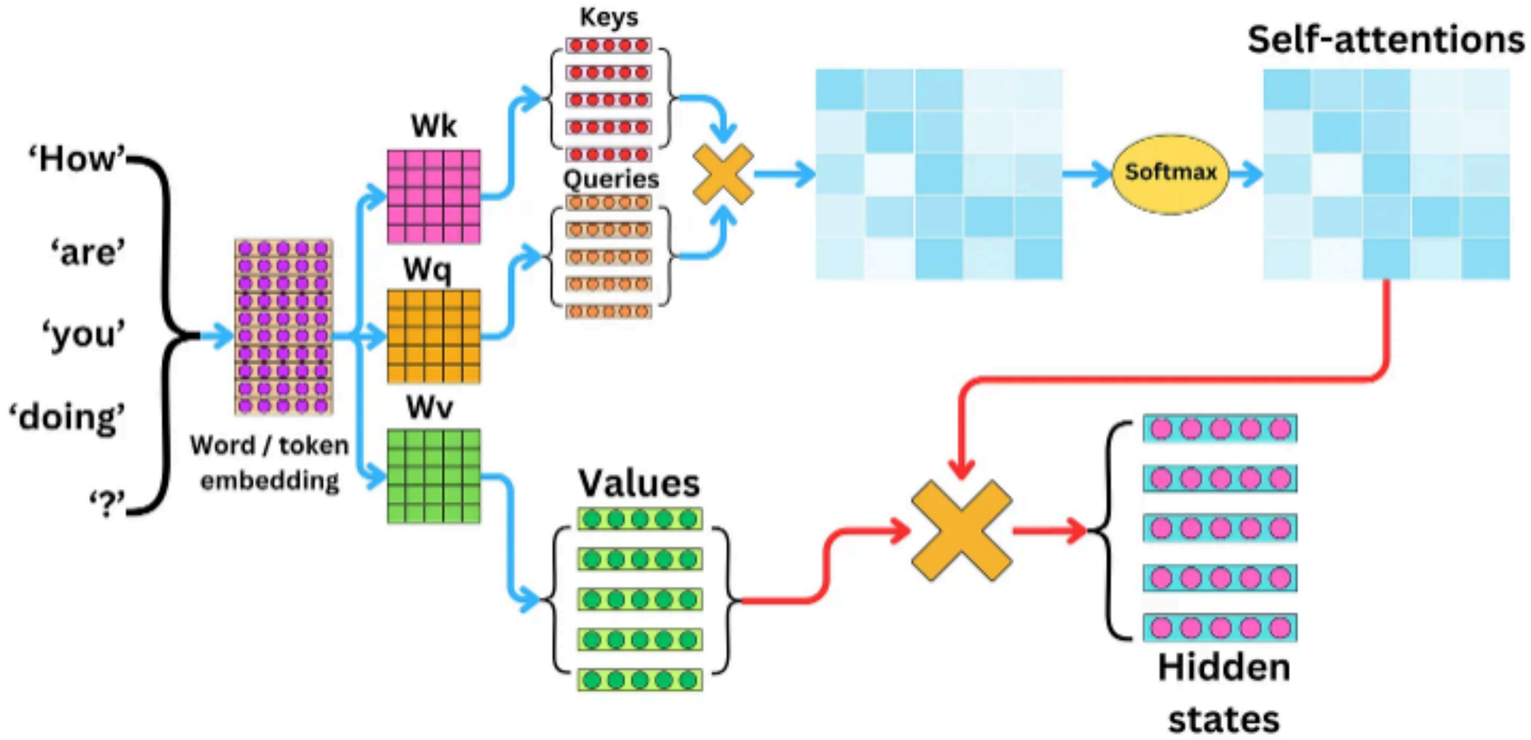
- In Bahdanau's RNN, Q, K, and V were all tied to the hidden states of the RNN
- In the QKV framework, we learn linear projections (W_Q, W_K, W_V) to transform the raw inputs into these three specialized spaces
- This gives the model the freedom to learn how to search for information, rather than being stuck with whatever the RNN provides

Question: In a Self-Attention layer, where does the Q, the K, and the V come from?

The "Attention Is All You Need" Philosophy

- **The Paradigm Shift**: Before 2017, **the consensus** was that **Recurrence was mandatory** for processing sequences.
- The "Attention Is All You Need" (Vaswani et al.) paper challenged this by:
  - Claiming **Attention is sufficient**: You don't need "loops" or "convolutions" to understand time or order
  - **Throwing away the RNN**: Hence solving the two biggest problems in AI: ***Vanishing Gradients and Hardware Inefficiency***
- **Why "All You Need"?**
  - Usually, attention was used as a 'helper' for an RNN
  - Here, it becomes the entire engine
  - Move from "How do we design a better memory cell?" to "How do we design a better communication network between words?"

Self-Attention: All-to-All Interaction



Credit: https://newsletter.theaiedge.io/p/understanding-the-self-attention

Self-Attention

- **Global Context**: To understand a single word, the model looks at the entire sentence simultaneously
- **Relationship Mapping**: The model calculates a "relevance score" between all pairs of words
- **"Self" Attention**: The Queries (Q), Keys (K), and Values (V) all originate from the same source
- **Example:** "The bank was flooded after the river overflowed"
  - The word "bank" attends heavily to "river" and “flooded"
  - "bank" refers to a geography feature

Self-Attention - QKV

- The process starts with a sequence of input vectors X
- To get $Q, K,$ and V, the model multiplies the input X by three different weight matrices (W^Q, W^K, W^V). These matrices are parameters that the model learns during training

$$Q = X \cdot W^Q$$

$$K = X \cdot W^K$$

$$V = X \cdot W^V$$

Benefits of All-to-All Interaction

- **Resolution of Ambiguity**: Pronouns (like "it" or "they") can find their **referents** regardless of how far apart they are in the text

- **Non-Linear Relationships**: It captures dependencies that don't follow a simple left-to-right order (e.g., a verb at the end of a sentence relating to a subject at the beginning)

- **Parallelism**: Because every word looks at every other word at the same time, the GPU doesn't have to wait for the previous word to finish processing

Self-Attention vs. Recurrence

- **The Structural Shift**
  - **Recurrence (RNN/LSTM)**: To know word 10, you must pass through words 1 to 9
  - **Self-Attention**: Every word looks at every other word simultaneously to determine its own context
- **Path Length & Long-Range Dependencies**
  - **RNN Path**: To connect word 1 to word 100, the gradient must travel 100 steps through time
  - **Attention Path**: The "distance" between any two words is always exactly 1. Word 1 can "see" Word 100 just as easily as it sees Word 2

The Power of Parallelization

- In an RNN, the GPU is **mostly idle** because it has to wait for h_{t-1} to finish
- In Self-Attention, the Q, K, and V matrices for the entire sentence are calculated in a **single massive matrix multiplication**
- We can utilize **thousands of GPU cores** at once
  - We can train on billions of words in days, rather than months
- **Computational Complexity** (n = sequence length and d = representation dimension)
  - **RNN**: $O(n \cdot d^2)$ for complexity per layer and $O(n)$ for operations
  - **Self-Attention**: $O(n^2 \cdot d)$ for complexity per layer and $O(1)$ for operations

Need for Multi-Head Attention

The "Single Head" Limitation

- In a standard Self-Attention layer, the model calculates one set of attention weights
- However, a word can have multiple different types of relationships with other words simultaneously
- **Linguistic Ambiguity**: One "head" might focus on grammar, while another needs to focus on semantic meaning
- **The "Averaging" Problem**: Model is forced to "average" all these different relationships into a single score, which can dilute important signals

Multi-Head Attention

- **The Solution: Parallel Heads**
  - Instead of performing a single attention function, we split the Query, Key, and Value vectors into multiple low-dimensional spaces
  - Each head has its own set of learnable weights (W_Q, W_K, W_V)
  - Each head performs Self-Attention independently
  - **Concatenation**: The results from all heads are "glued" back together and projected into the final output dimension
- **What do the Heads actually learn?**
  - **Head 1**: Might focus on the immediate neighbour (local context)
  - **Head 2**: Might focus on the relationship between a Verb and its Object
  - **Head 3**: Might focus on coreference (matching "it" to "the dog”)

The "Committee of Experts" Analogy:

"Imagine you are trying to understand a complex movie. If you watch it alone, you might miss the historical context while focusing on the dialogue. Multi-Head Attention is like having a committee of 8 experts watching the same movie simultaneously.

One expert is a historian, one is a linguist, one is a cinematographer, and one is a psychologist. They each 'attend' to different details. At the end, they all sit down, combine their notes, and give you a complete, multi-faceted understanding of the scene. That is what Multi-Head Attention does for every word in a sentence."

Discussion: "If we have 100 heads, would the model be 100 times smarter? Is there a downside to having too many heads?”

Positional Encoding — Restoration of Order

The Problem: Permutation Invariance

- **Order-less Attention**: In a pure Self-Attention mechanism, the relationship between words is based only on their content (Q, K, V)
- **Scrambled Meaning**: Without a sense of position, the model treats "The dog bit the man" and "The man bit the dog" as identical sets of vectors
- **RNN vs. Transformer**: In an RNN, position is implicit (time-step t). In a Transformer, we must explicitly add position information to the input embeddings

Positional Encoding - Solution

The Solution: Sinusoidal Encodings

- We add a unique position vector PE to each input embedding E
- Formula

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

- Why Sinusoids? It allows the model to attend by relative positions, as PE_{pos+k} can be represented as a linear function of PE_{pos}

Understanding the formula

- Think about how we represent numbers in binary:

  0: 000

  1: 001

  2: 010

  3: 011

- Notice that the "least significant bit" (the rightmost column) toggles every step, the next column toggles every two steps, and the next every four. We are essentially using different frequencies to encode position. Sinusoidal encoding is the continuous version of this binary counter

Understanding the formula - II

- We define the encoding for a word at a specific position (pos) across its embedding dimensions (i):

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

- pos: The index of the word in the sentence (e.g.,$0,1,2,\ldots$)
- i: The index of the dimension in the embedding vector. If your embedding is 512 dimensions, i ranges from 0 to 255 (since we use $2i$ for sine and $2i + 1$ for cosine)
- d_{model}: The total dimensionality of the embedding (e.g., 512)

Understanding the formula - III

- **Why the 10000 Constant?**
  - The term $10000^{2i/d_{model}}$ is the wavelength.
  - In the first few dimensions ($i = 0$), the denominator is small, meaning the sine wave has a very high frequency (it changes rapidly from word to word).
  - In the deep dimensions ($i = 255$), the denominator is very large, meaning the wave has a very low frequency (it changes very slowly across the sentence).
- **Why Sine AND Cosine?**
  - The authors chose pairs of sine and cosine because of the Trigonometric Addition Theorem:

    $$\sin(\alpha + \beta) = \sin\alpha\cos\beta + \cos\alpha\sin\beta$$

    $$\cos(\alpha + \beta) = \cos\alpha\cos\beta - \sin\alpha\sin\beta$$

  - Because of this property, for any fixed offset k, PE_{pos+k} can be represented as a linear function of PE_{pos}. This makes it mathematically easy for the model to learn to attend to "the word 3 places to the left," regardless of where in the sentence that word is.

How the Dimensions are Used

The choice of embedding dimension is a balance between expressivity and computational cost:

- **Information Capacity**: Each of the 512 dimensions can learn to represent a different "feature" of the word (e.g., Is it a verb? Is it plural? Is it related to technology?).
- **Multi-Head Split**: 512 is a popular number—it is easily divisible
- **The Positional "Clock"**: 512 dimensions are used to create the different "wavelengths."
  - Dimension 0: Very fast frequency (changes significantly between word 1 and word 2).
  - Dimension 511: Very slow frequency (barely changes across a sentence of 50 words).

Positional Encoding - Properties

- **Fixed Length**: Each position has a unique signature
- **Extrapolation**: Theoretically allows the model to handle sequences longer than those seen during training
- **Non-Destructive**: Because the encodings are added to the embeddings, the model learns to separate "meaning" from "position" during training

Masked Self-Attention — Preventing "Cheating" in the Decoder

The Problem: Looking into the Future

- **Parallel Training**: Unlike RNNs, which see one word at a time, Transformers see the entire sequence at once during training to stay fast
- **The Leakage Risk**: If the model is trying to predict the next word in the sequence "**The cat sat on the mat**," we cannot allow the attention mechanism for the word "**sat**" to "see" the word "**on**"
- **Auto-regressive Consistency**: At inference time, the future words don't exist yet. The training environment must match this reality

The Solution: The Masking Matrix

- We apply a mask to the attention scores before the softmax step
  - **Illegal Connections**: Any connection from a word at position i to a word at position j (where $j > i$) is set to $-\infty$
  - **The Result**: When the softmax is calculated, these "future" positions receive a weight of 0, effectively making them invisible to the current token
- **Mathematical Implementation**
  - Inside the Attention formula:

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

  - Where M is the mask matrix:
    - $M_{ij} = 0$ for $j \leq i$ (Visible)
    - $M_{ij} = -\infty$ for $j > i$ (Masked)

Layer Normalization — Stabilizing the Deep Architecture

The Problem: **Internal Covariate Shift**

- In deep networks like the Transformer, the distribution of inputs to each layer changes as the parameters of the previous layers change during training
- Vanishing/Exploding Gradients: Without normalization, activations can become very large or very small, slowing down training or causing it to fail
- Stacking layers can destabilize training

Layer Normalization - Solution

- **Batch Normalization** - Normalizes across a batch of data
- **Layer Normalization (LN)** - normalizes the inputs across the feature dimension (d_{model}) for each individual sequence element
- **The Formula:** For a vector x (the embedding of a word), we calculate the mean (μ) and variance (σ^2) of all 512 dimensions:

$$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} \cdot \gamma + \beta$$

  - μ, σ: Mean and standard deviation of the current word's features
  - γ, β: Learnable parameters that allow the model to "re-scale" the data if needed
  - ϵ: A tiny constant for numerical stability

Why Layer Norm for Transformers?

- **Independence from Batch Size**
  - LN performs the same whether your batch size is 1 or 1000
  - Crucial for large models where memory limits force small batches
- **Sequence Length Flexibility**:
  - LN works per-token
  - Ideal for the varying sequence lengths common in NLP
- **Training Speed**:
  - Allows for much higher learning rates
  - Significantly speeding up training process