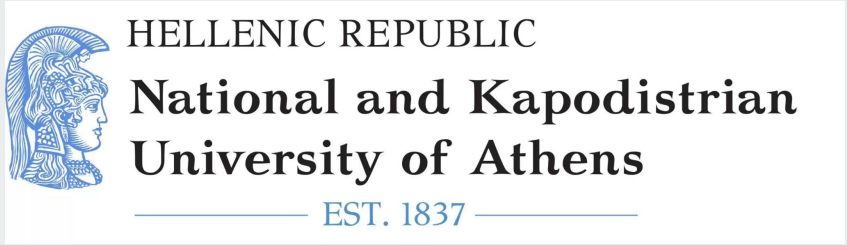
Computational Learning Theory - 2025
Lab for Applications with LLMs

B.Sc. Informatics & Telecommunications

Department of Informatics & Telecommunications
National & Kapodistrian University of Athens

Yorgos Pantis - pantisyorgos@gmail.com



(1/3) Introduction

Data:

- Transformer architecture was originally introduced for sequential text data (e.g., natural language)
- Later adapted to other data types such as: Images (e.g., Vision Transformers, ViT), Audio, Video etc

Transformer:

- A specific kind of neural network architecture based on self-attention mechanisms, see “Attention is All You Need” by Vaswani et al. (2017)

Unlike RNNs or feedforward networks:

- Processes input sequences in parallel
- Focuses dynamically on different parts of the input
- Captures relationships between distant elements in the input

(2/3) Introduction

LLMs (Large Language Models):

- Examples: ChatGPT, Gemini, DeepSeek, Claude
- Built around Transformer architecture as the core building block.
- Use additional techniques on top, such as:
    - Reinforcement Learning (RL) (e.g., RLHF — Reinforcement Learning from Human Feedback)
    - Supervised fine-tuning
    - Prompt tuning and other optimization strategies

Transformers allow LLMs to:

- Handle very large amounts of data
- Generate long pieces of text
- Show complex behaviors like reasoning, summarizing, and coding

(3/3) Introduction

Positives:

- Parallelizable — Faster training compared to RNNs
- Scalable — Performance improves predictably with model and data size
- Flexible — Works across text, images, audio, video, and code
- Captures Long-Range Dependencies: Easily relates distant parts of the input without memory loss, unlike RNNs

Negatives:

- Computationally Expensive — Training requires large compute resources
- Data Hungry — Needs massive datasets for best performance
- Less Interpretable — Hard to understand decision-making internally
- Training Instability — Sensitive to initialization, learning rate, etc

(1/3) How to Construct a Model Like ChatGPT from Scratch

1. Gathering and Preparing the Data:
    - To train a model like ChatGPT, huge amounts of data are needed.
    - Public text sources like books, websites, Wikipedia, etc.
    - Curated datasets specific to conversational data, dialogue, and human interactions.
    - Data preprocessing includes removing noise, tokenize data, etc
2. Designing the Model Architecture (Transformers):
    - Self-attention layers to understand relationships between words, even far apart in a sentence
    - Positional encoding to retain word order, since Transformers process words in parallel (not sequentially like RNNs)
    - Feedforward layers and multi-head attention for scaling and capturing complex patterns

(2/3) How to Construct a Model Like ChatGPT from Scratch

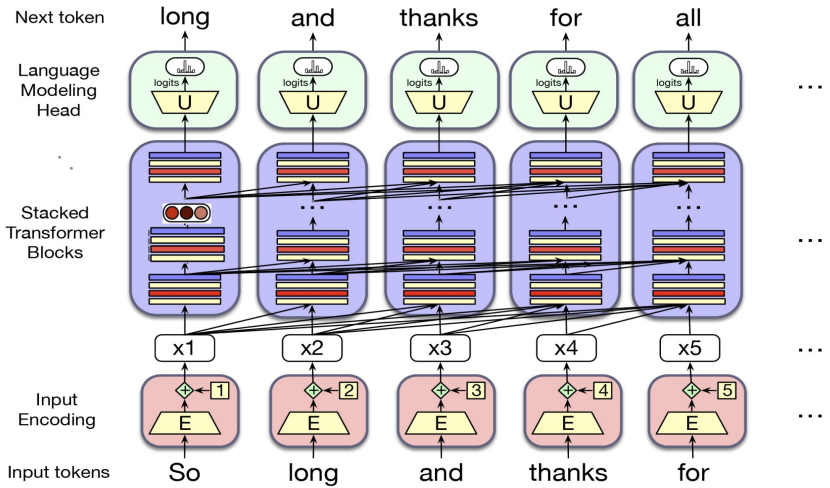
3. Model Pre-training:
    - The model learns to predict the next word in a sequence of text, based on its context. This stage is unsupervised and focuses purely on learning language patterns, such as grammar, sentence structure, facts, and some reasoning ability
    - Minimize the next token prediction errors across large text corpora (books, websites, etc.)
    - The model learns to handle vast amounts of data in a scalable way, capturing general knowledge from the training material
4. Supervised Fine-tuning (SFT):
    - After pre-training, the model needs to be fine-tuned for specific tasks like answering questions, summarizing, or having a conversation
    - SFT uses labeled datasets where humans provide high-quality examples of how the model should respond
    - It focuses on teaching the model to follow instructions and give correct, relevant, and useful answers

(3/3) How to Construct a Model Like ChatGPT from Scratch

5. Reinforcement Learning from Human Feedback (RLHF):
    - To take the model even further, human feedback is used to align it with human preferences.
    - Multiple responses are generated for the same prompt, and human reviewers rank these responses based on quality and helpfulness.
    - A reward model is trained on these rankings, and the model is further optimized using Reinforcement Learning (usually Proximal Policy Optimization, PPO).
    - This ensures the model doesn't just produce fluent responses, but responses that are safe, helpful, and aligned with user needs.

Word Embeddings

Instead of jumping to the full Transformer architecture, let's first examine how an individual word is represented at a single layer



The Problem with Static Embeddings (e.g., word2vec)

- Static embeddings assign one fixed vector to each word, regardless of context
- But word meanings change based on context
- Example: **The chicken didn't cross the road because it was too tired**

  — What does "it" refer to here? A static embedding can't tell.

Contextual Embeddings: Meaning Shaped by Context

- Key idea: a word's representation should adapt to its surrounding words
- Contextual embeddings create different vectors for the same word depending on the sentence
- How do we compute contextual embeddings? → Attention mechanisms

Example of Contextual Embeddings

The chicken didn't cross the road because it...

Question: What properties should the representation of "it" have at this point?

It should remain ambiguous, able to refer to either the chicken or the road.

Later context clarifies:

- "...because it was too tired" → "it" refers to the chicken
- "...because it was too wide" → "it" refers to the road

Key takeaway: A good embedding must adapt dynamically based on the evolving sentence context.

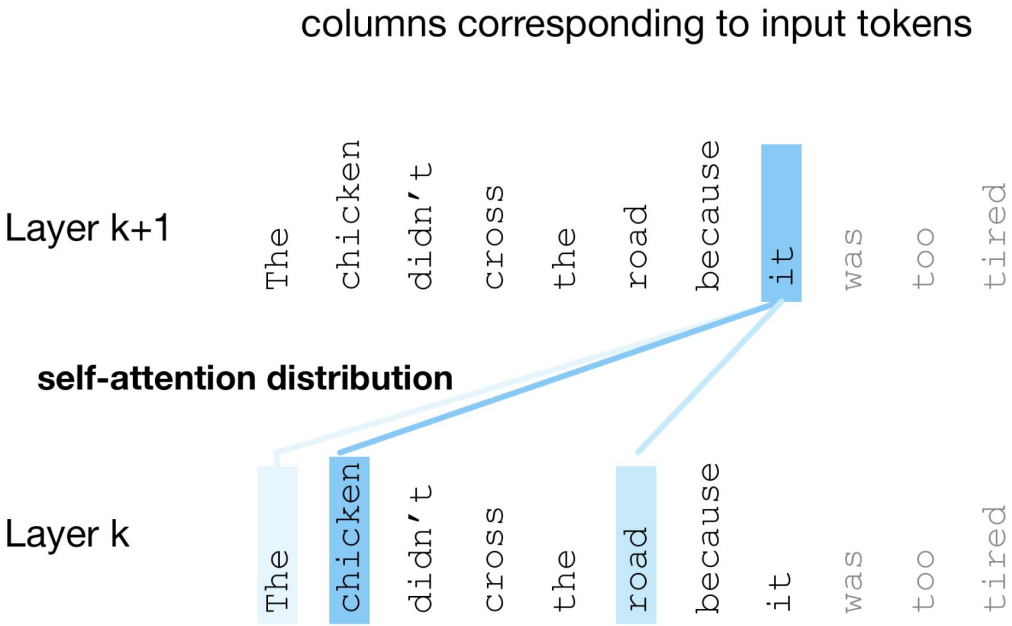
(1/3) Intuition of Attention

Contextual embeddings are built by selectively integrating information from neighboring words.

A word "attends to" certain neighboring words more than others, based on their relevance.

Attention allows the model to focus on important words for context while ignoring less relevant ones.

(2/3) Intuition of Attention

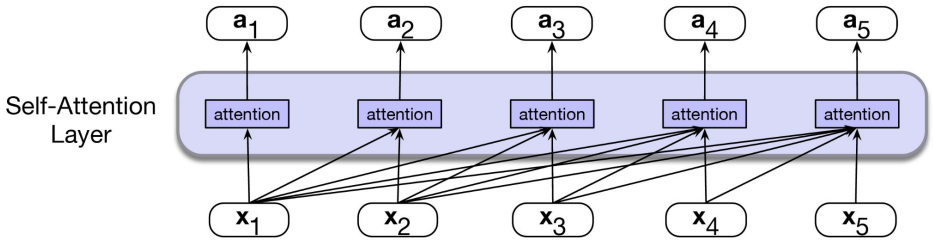


(3/3) Intuition of Attention

- Contextual embeddings are built by selectively integrating information from neighboring words
- A word "attends to" certain neighboring words more than others, based on their relevance
- Attention allows the model to focus on important words for context while ignoring less relevant ones

Attention in LLMs

- Attention is not inherently left-to-right. The direction depends on the architecture
- BERT uses bidirectional attention, meaning it can consider both past and future words to form the context for each word.
- GPT uses causal (left-to-right) attention, where each word can only attend to previous words (or itself)
- Attention direction is flexible — bidirectional in BERT and causal in GPT (and other autoregressive models)



(1/2) Simplified Version of Attention

Given a sequence of token embeddings

$$\mathbf{x}_1 \quad \mathbf{x}_2 \quad \mathbf{x}_3 \quad \mathbf{x}_4 \quad \mathbf{x}_5 \quad \mathbf{x}_6 \quad \mathbf{x}_7 \quad \mathbf{x}_i$$

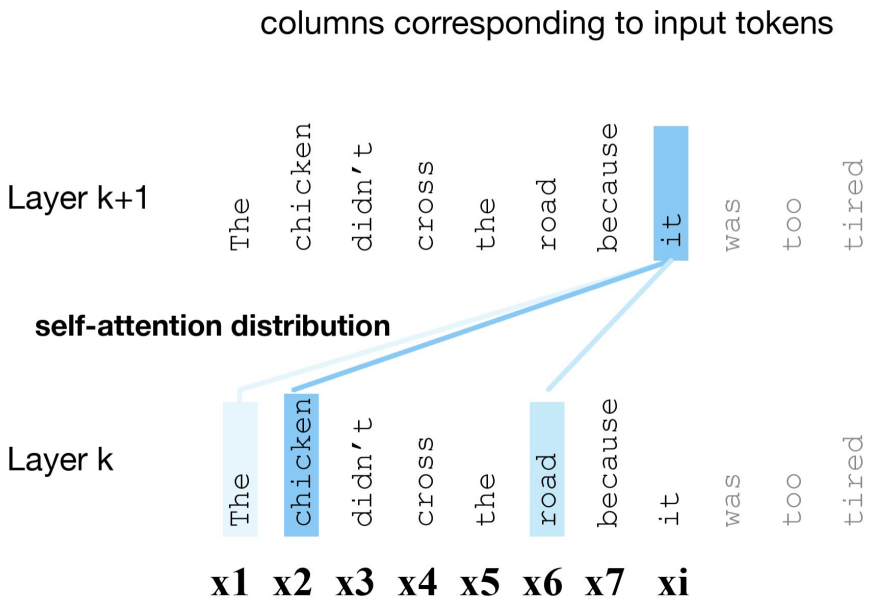
Compute the following

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$$

$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \ \forall j \leq i$$

$$\mathbf{a}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{x}_j$$

(2/2) Simplified Version of Attention



Advanced Version of Attention

Instead of using vectors directly, we represent three separate roles each vector plays:

Query:

- The current element being compared to preceding inputs
- Represents the focus of attention.

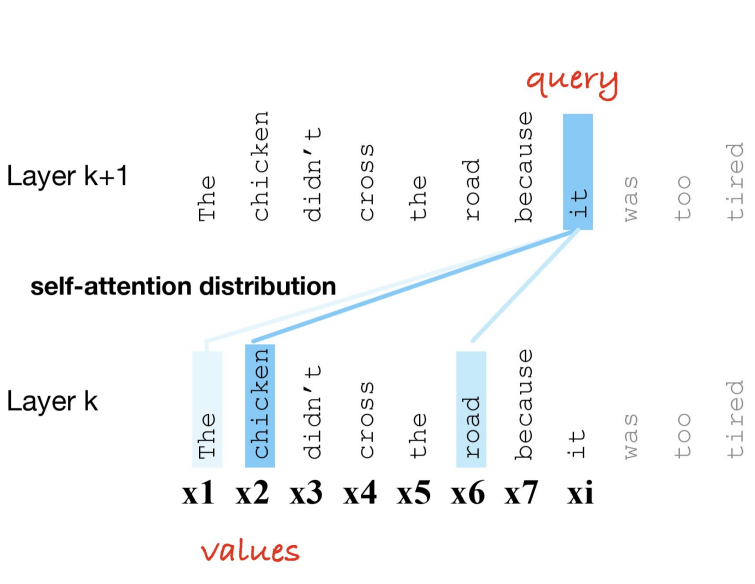
Key:

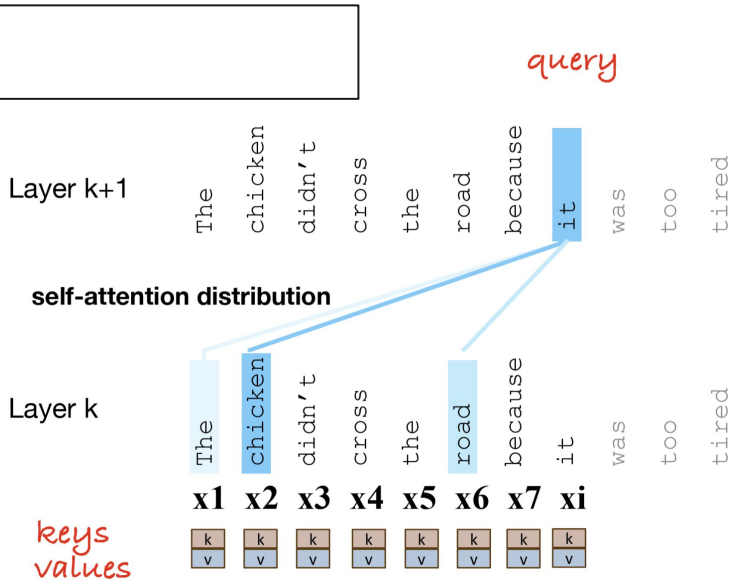
- A preceding input that is compared to the current element to compute similarity
- Helps determine how much attention should be paid to other tokens

Value:

- The value of a preceding element that gets weighted and summed to contribute to the final output
- Determines the actual information passed to the next layer.

Advanced Version of Attention





(1/2) Three Roles in Attention

These weight matrices project each input vector xi into its respective role:

- Query: $q_i = xiW^q$
- Key: $k_i = x_iW^k$
- Value $V_i = x_iW^v$

These projections allow the model to differentiate
the roles of each token and calculate attention effectively

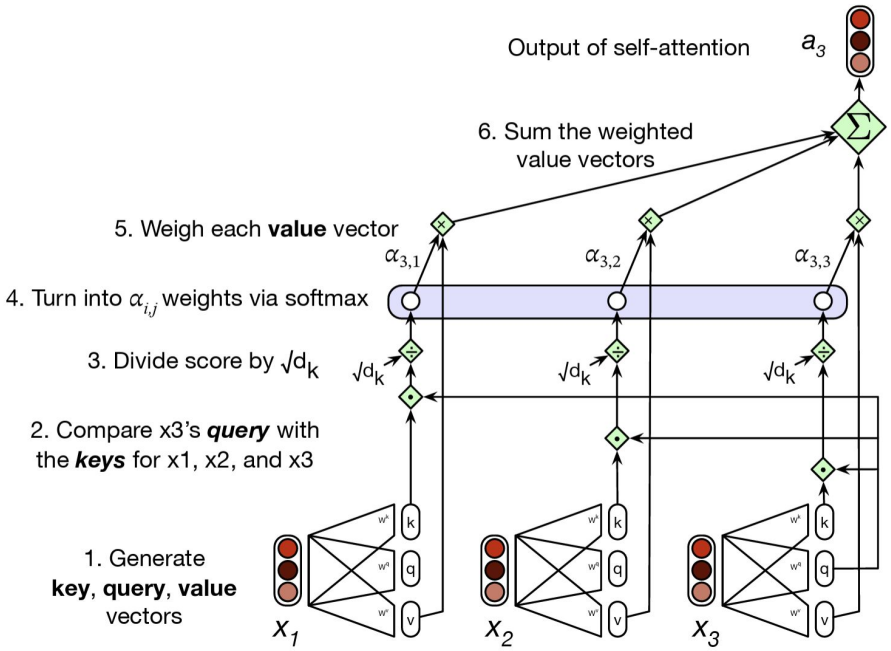
$$\mathbf{q}_i = \mathbf{x}_i\mathbf{W}^{\mathbf{Q}};\quad \mathbf{k}_j = \mathbf{x}_j\mathbf{W}^{\mathbf{K}};\quad \mathbf{v}_j = \mathbf{x}_j\mathbf{W}^{\mathbf{V}}$$

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}$$

$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j))\ \ \forall j \leq i$$

$$\mathbf{a}_i = \sum_{j \leq i} \alpha_{ij}\mathbf{v}_j$$

(2/2) Three Roles in Attention



(1/4) Multihead Attention

Attention Mechanism:

- In self-attention, the model computes a score for each pair of tokens in the input, which determines how much attention each token should pay to every other token
- An attention head is a single instance of this process. It computes its own set of attention scores and produces its own weighted sum of values

Multiple Heads:

- Instead of using a single attention mechanism, transformers use multiple heads (e.g., 8 or 16 heads)
- Each head operates independently, learning different aspects of the relationships between tokens in the sequence

(2/4) Multihead Attention

Why Multiple Heads?

Different heads can focus on different types of information. For example:

- One head might focus on syntactic relationships (e.g., subject-verb agreement)
- Another head might focus on semantic relationships (e.g., recognizing synonyms)
- Other heads might capture other types of dependencies, like co-reference or long-range dependencies

Process in Multi-Head Attention, for each attention head:

- Compute queries, keys, and values using different sets of learned weights
- Calculate attention scores, apply softmax, and get the weighted sum of the values
- After all heads have processed the information, the outputs of all heads are concatenated and linearly transformed to form the final result

(3/4) Multihead Attention

Imagine you're processing the sentence

"The cat sat on the mat."

- One attention head might focus on how "cat" and "sat" are related syntactically (e.g., subject-verb agreement)
- Another head might focus on the relationship between "mat" and "on", understanding spatial relationships
- Yet another head might focus on broader semantic dependencies, understanding that "cat" and "mat" are related objects in a typical setting

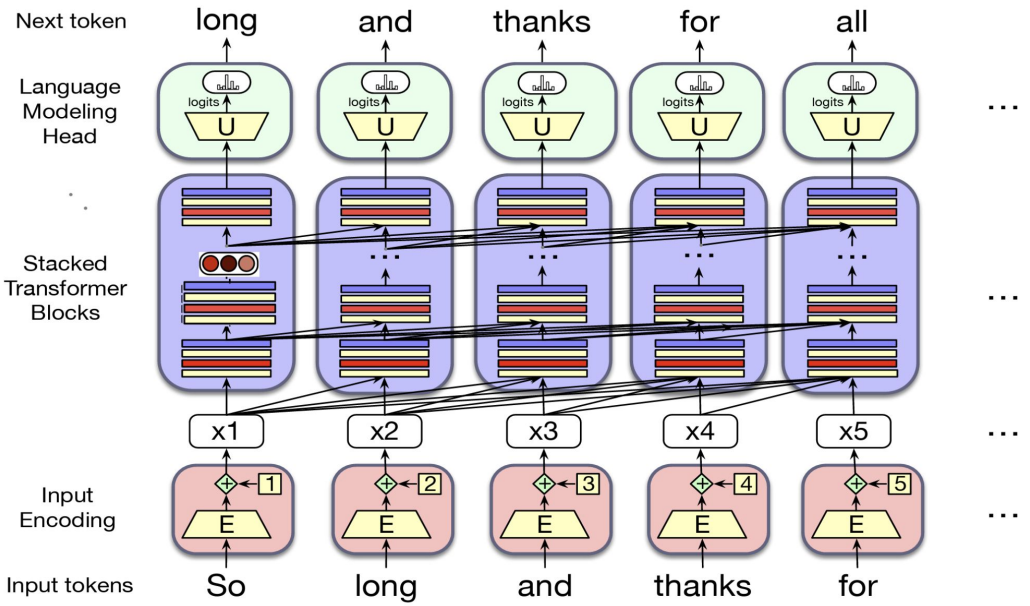
(4/4) Multihead Attention

$$
\begin{aligned}
\mathbf{q}_i^c = \mathbf{x}_i \mathbf{W}^{\mathbf{Qc}};\quad \mathbf{k}_j^c = \mathbf{x}_j \mathbf{W}^{\mathbf{Kc}};\quad \mathbf{v}_j^c &= \mathbf{x}_j \mathbf{W}^{\mathbf{Vc}};\quad \forall\, c \quad 1 \le c \le h \\
\text{score}^c(\mathbf{x}_i, \mathbf{x}_j) &= \frac{\mathbf{q}_i^c \cdot \mathbf{k}_j^c}{\sqrt{d_k}} \\
\alpha_{ij}^c &= \text{softmax}(\text{score}^c(\mathbf{x}_i, \mathbf{x}_j))\ \ \forall j \le i \\
\mathbf{head}_i^c &= \sum_{j \le i} \alpha_{ij}^c \mathbf{v}_j^c \\
\mathbf{a}_i &= (\mathbf{head}^1 \oplus \mathbf{head}^2 \ldots \oplus \mathbf{head}^h)\mathbf{W}^O \\
\text{MultiHeadAttention}(\mathbf{x}_i, [\mathbf{x}_1, \cdots, \mathbf{x}_N]) &= \mathbf{a}_i
\end{aligned}
$$

Summary of Attention

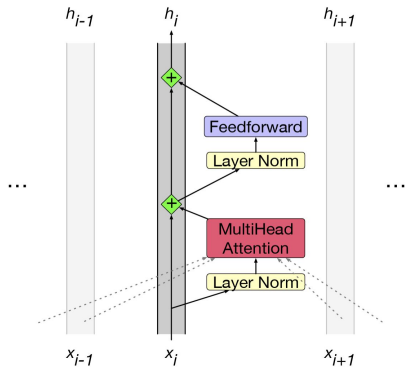
- Attention: A method to enrich the representation of a token by incorporating contextual information from other tokens
- Result: The embedding for each word will vary depending on its context
- Contextual Embeddings: A representation of a word's meaning that changes based on its surrounding words
- Attention can also be viewed as a way to transfer information from one token to another

Transformers Architecture



Residual Stream

- The term residual stream refers to a concept in deep learning where the output of each layer in a neural network is combined with the input of that layer before being passed to the next layer
- Each token is passed through the model and continuously modified at each layer
- The token's representation is updated and refined through each stage of processing



Nonlinearities in Transformer

FeedForward Network:

- $\text{FFN}(\mathbf{x}_i) = \text{ReLU}(\mathbf{x}_i\mathbf{W}_1 + b_1)\mathbf{W}_2 + b_2$

Layer Normalization:

- Layer Norm: A variation of the z-score from statistics, applied to a single vector in a hidden layer
- It standardizes the values within the layer to improve training stability and convergence

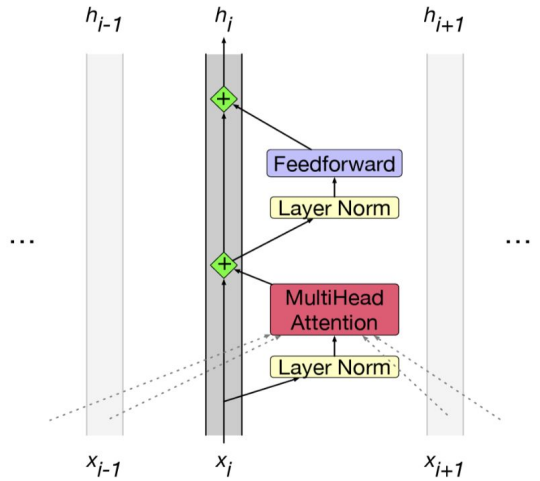
$$\mu = \frac{1}{d}\sum_{i=1}^{d} x_i$$

$$\sigma = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(x_i - \mu)^2}$$

$$\hat{\mathbf{x}} = \frac{(\mathbf{x} - \mu)}{\sigma}$$

$$\text{LayerNorm}(\mathbf{x}) = \gamma\,\frac{(\mathbf{x} - \mu)}{\sigma} + \beta$$

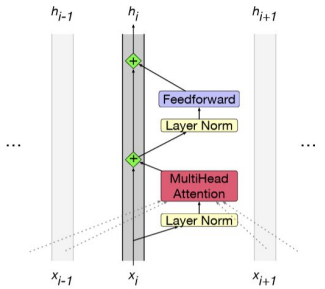
Simple Transformer Block



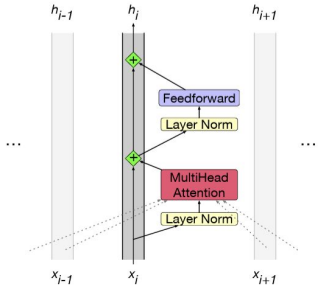
$$
\begin{aligned}
\mathbf{t}_i^1 &= \text{LayerNorm}(\mathbf{x}_i) \\
\mathbf{t}_i^2 &= \text{MultiHeadAttention}(\mathbf{t}_i^1, [\mathbf{x}_1^1, \cdots, \mathbf{x}_N^1]) \\
\mathbf{t}_i^3 &= \mathbf{t}_i^2 + \mathbf{x}_i \\
\mathbf{t}_i^4 &= \text{LayerNorm}(\mathbf{t}_i^3) \\
\mathbf{t}_i^5 &= \text{FFN}(\mathbf{t}_i^4) \\
\mathbf{h}_i &= \mathbf{t}_i^5 + \mathbf{t}_i^3
\end{aligned}
$$

Multiply Transformers Blocks

Block 2



Block 1



Note

Residual Stream:

- Every part of the Transformer (like feedforward layers, layer normalization, etc.) operates on a single token's residual stream. This means each token is updated independently in terms of its own context

Attention

- Attention is different because it doesn't just process a token in isolation. Instead, attention takes information from other tokens in the sequence to update the representation of the current token
- In simple terms, attention allows a token to gather information from its neighbors (or surrounding tokens) based on their relevance

(1/2) Parallelization in Transformers

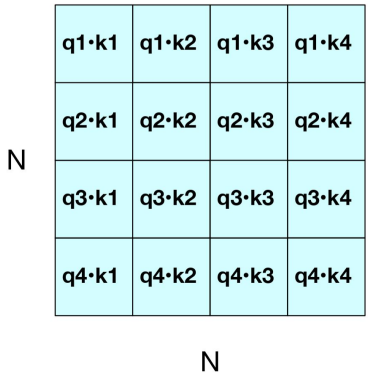
- Previously, computation was done for a single output at a single time step i in a single residual stream
- Instead, we can pack all N tokens of the input sequence into a single matrix X of size [N×d]
- Each row of X is the embedding of one input token

Examine the process for a single attention head before extending to multiple heads and the full transformer block. Calculate the following:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^{\mathbf{Q}};\ \ \mathbf{K} = \mathbf{X}\mathbf{W}^{\mathbf{K}};\ \ \mathbf{V} = \mathbf{X}\mathbf{W}^{\mathbf{V}}$$

(2/2) Parallelization in Transformers

Simply matrix multiply to combine Q and K^T:



Attention vector for each input token:

$$\mathbf{A} = \text{softmax}\left(\frac{\mathbf{QK}^T}{\sqrt{d_k}}\right)\mathbf{V}$$

(1/2) Mask in Transformers

Mask the future: $$\mathbf{A} = \text{softmax}\left(\text{mask}\left(\frac{\mathbf{Q}\mathbf{K}^\intercal}{\sqrt{d_k}}\right)\right)\mathbf{V}$$

Problem with Self-Attention:

- In the standard self-attention computation, the calculation of QK^T results in a score for each query-token with respect to every key-token, including those after the query
- This can be problematic in language models, where the goal is to predict the next token without peeking at future tokens
- Guessing the next token is easy if you already know it

Solution:

- The mask function ensures that attention is only computed for tokens that occur before or at the current token in the sequence, maintaining the autoregressive nature of the model

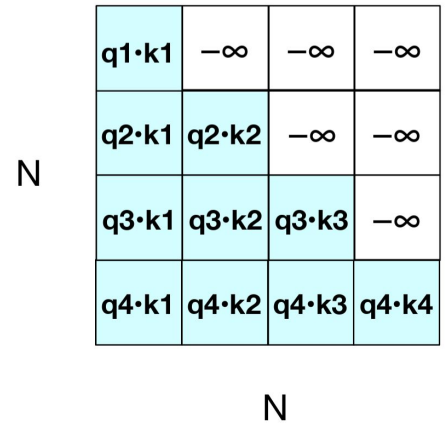
(2/2) Mask in Transformers

Masking Future Tokens:

- To prevent attending to future tokens, we mask the scores for future tokens by setting them to negative infinity (-∞)
- Softmax of -∞ effectively becomes 0
- This ensures that the future tokens' attention scores contribute no

Effect on the Attention Matrix:

- The attention matrix is a square matrix of shape [N × N], where eac token in the sequence
- For token i, we mask all future tokens j > i (those above the diagona upper triangle of the matrix



Token and Positional Embeddings

The matrix X has shape [N × d], where:

- N is the number of tokens in the input sequence
- d is the dimensionality of the embedding (model dimension)
- Each row of matrix X represents the embedding of a word in the context

The embedding for each token is created by adding two distinct embeddings:

- Token Embedding: The embedding for the actual word or token in the input sequence
- Positional Embedding: A unique embedding that encodes the position of each token in the sequence

(1/2) Token Embeddings

Embedding Matrix E has shape [|V| × d], where:

- |V| is the size of the vocabulary (number of unique tokens)
- d is the dimensionality of each embedding vector

Each row of the matrix represents the embedding for a specific token in the vocabulary

(2/2) Token Embeddings

Example: Given the input string "Thanks for all the"

Step 1: Tokenize with Byte Pair Encoding (BPE) and convert it into vocabulary indices: w = [5, 4000, 10532, 2224] (These numbers represent the token indices in the vocabulary)

Step 2: Select the corresponding rows from the embedding matrix E:

- The embedding for token "Thanks" is in row 5
- The embedding for token "for" is in row 4000
- The embedding for token "all" is in row 10532
- The embedding for token "the" is in row 2224.

Each row corresponds to a d-dimensional embedding vector for that specific token.

(1/2) Positional Embeddings

Goal: Learn a position embedding matrix Epos of shape [1 × N], where:

- N is the maximum sequence length (e.g., 512 tokens)
- Each position in the sequence (like position 1, 2, 3, ...) has its own learned embedding

(2/2) Positional Embeddings

Example:

Step 1: Initialize a random position embedding matrix Epos where each position (e.g., position 1, 2, 3) corresponds to a unique embedding

Step 2: For a token at position 3 in the sequence:

- Use the embedding corresponding to position 3 from the position embedding matrix.

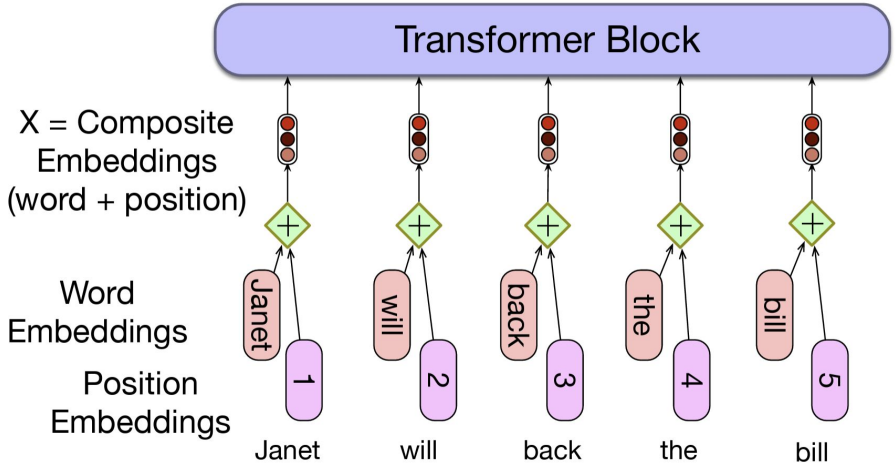
Step 3: For a token at position 17:

- Use the embedding corresponding to position 17 from the position embedding matrix

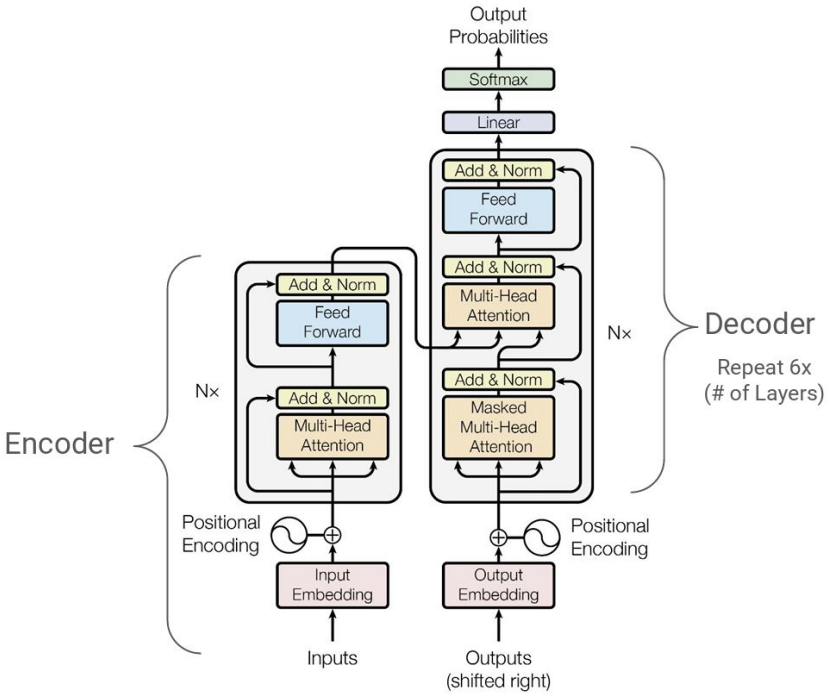
Learning: Just like word embeddings, these position embeddings are learned during training

Summary of Token and Positional Embeddings

Each X is just the sum of word and position embeddings



Putting all Together



(1/2) Alignment, Prompting, and In-Context Learning

Post-training and Model Alignment:

- Post-training: Fine-tuning a model after its initial large-scale pretraining to specialize it for specific tasks or behaviors
- Model alignment: Adjusting models so their outputs align with human intentions, ethics, and societal values
- Techniques:
    - Supervised Fine-Tuning (SFT): Training on labeled datasets with human-written examples
    - Reinforcement Learning from Human Feedback (RLHF): Using human feedback to guide model improvements

(2/2) Alignment, Prompting, and In-Context Learning

Prompting:

- Definition: Directly steering model behavior by crafting input prompts
- Zero-shot, one-shot, and few-shot prompting techniques
- Importance of prompt design for performance and generalization

Chain-of-Thought Prompting:

- Strategy: Encourage the model to reason step-by-step
- Key to solving complex tasks like arithmetic, logic puzzles
- Boosts performance in tasks requiring multi-step reasoning

Automatic Prompt Optimization:

- Automating the search for effective prompts
- Techniques: Prompt tuning, soft prompts, learned embeddings
- Benefits: Reduces reliance on manual prompt engineering

(1/2) Conclusion and Discussion

- Attention mechanisms solve the the context vector limitation in sequence models by producing dynamically derived context vectors
    - They allow the model to focus selectively on relevant parts of the input, improving handling of long-range dependencies
- Transformers build on attention mechanisms and introduce several key innovations:
    - Self-attention layers capture relationships between all tokens in a sequence, regardless of their distance
    - Feedforward layers apply non-linear transformations independently to each token's representation
    - Parallelization is possible because token computations are independent (unlike RNNs), speeding up training significantly
    - Positional encoding is added to input embeddings to provide information about the position/order of tokens, as self-attention alone is position-agnostic

(2/2) Conclusion and Discussion

- Other important components of Transformer architecture:
  - Multi-head attention: Multiple attention heads allow the model to jointly attend to information from different representation subspaces
  - Residual connections: Each sub-layer (attention or feedforward) has a skip connection, helping with gradient flow and enabling deeper models
  - Layer normalization: Applied after residual connections to stabilize and speed up training
  - Masked self-attention (in the decoder): Ensures that each output token prediction depends only on previous known outputs during training
  - The encoder processes the input sequence into a set of continuous representations
  - The decoder generates the output sequence step-by-step using the encoder's outputs and its own previously generated tokens

Suggested Material

1. Jurafsky, D., & Martin, J. H. (2025). Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition (3rd ed., draft). Retrieved from https://web.stanford.edu/~jurafsky/slp3/
2. Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015).
3. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. Advances in Neural Information Processing Systems (NeurIPS), 30.
4. Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2019), 1, 4171–4186.