Institute for Advanced Studies
in Basic Sciences
Gava Zang, Zanjan, Iran

Understanding Large Language Models:

A Functional Overview with Mathematical Intuition

Majid Ramezani
ramezani@iasbs.ac.ir

Lecture Outline

Chapter 1. Introduction to Language and Language Models

Chapter 2. Types of LLM Architectures

Chapter 3. Core Mathematical & Statistical Concepts

Main Goals!

- **At the end of this session, we are going to know that:**

G1: What is the language!

G2: What is the language model!

G3: What are the types of language models and what are their use cases?

G4: What is the main mathematical/statistical concept behind language models?

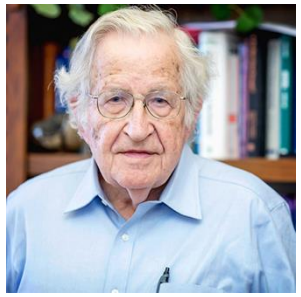
Lecture Outline

Chapter 1. Introduction to Language and Language Models

Chapter 2. Types of LLM Architectures

Chapter 3. Core Mathematical & Statistical Concepts

1-1 What is “Language”?

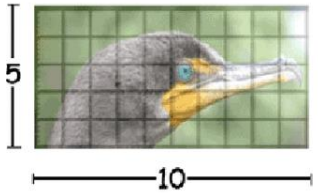


Chomsky “A **language** is a collection of sentences of finite length all constructed from a finite alphabet (or, where our concern is limited to syntax, a finite vocabulary) of symbols.”

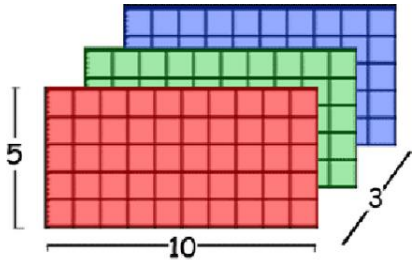
Question!!

Based on this definition, can we claim that many sequences could be considered a “*language*”?!

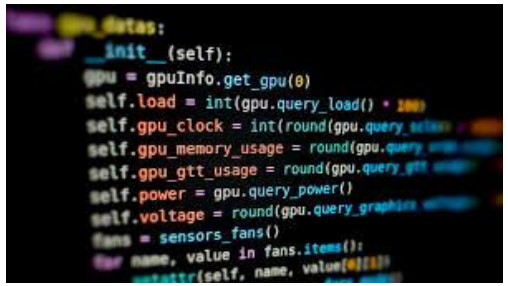
$$x^m = a \Rightarrow \log_x a = m$$
$$\log_a 1 = 0$$
$$\log_a a = 1$$
$$\log_a(xy) = \log_a x + \log_a y$$
$$\log_a \frac{x}{y} = \log_a x - \log_a y$$
$$\log_a(x^m) = m \log_a x$$
$$\log_a x = \frac{\log_c x}{\log_c a}$$
$$a^{\log_a x} = x$$



Original Color Image



Matlab RGB Matrix



1-2 Language Models

Definition and Purpose of Language Models

A probabilistic model that predicts the **next word/token** in a sequence.

- Core objective: Understanding and generating human-like text.
- Real-world applications: text generation tasks such as machine translation, text summarization, chatbots and virtual assistants, code generation, etc.

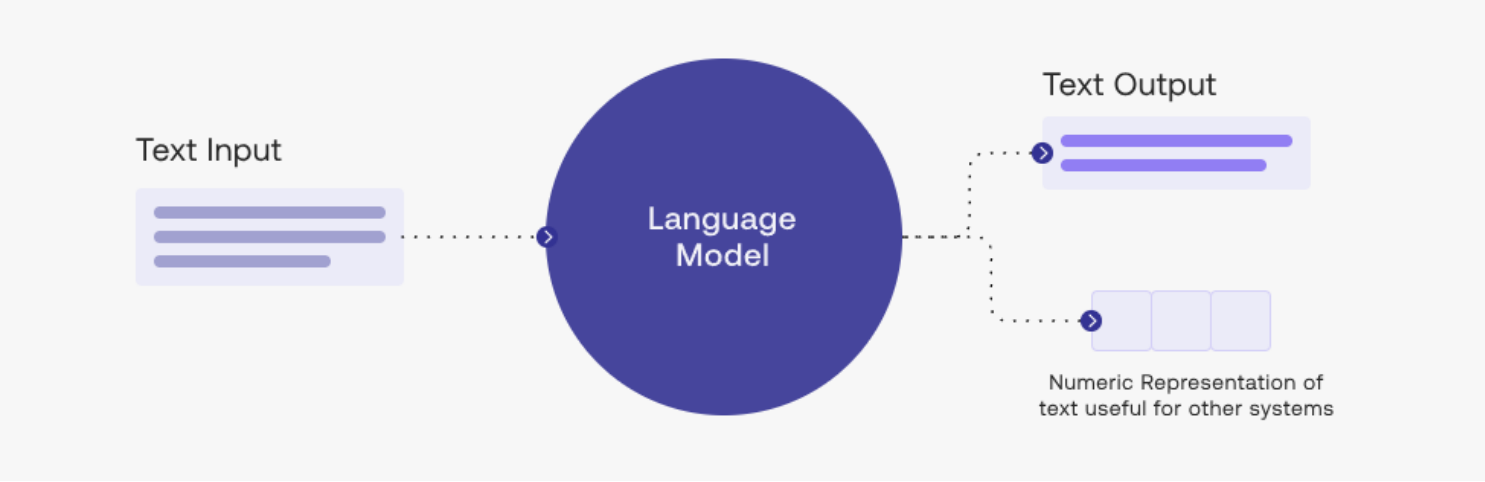
1-2 Language Models

Definition and Purpose of Language Models (continued….)

How Language Models Work?

- Probability Estimation: Predicting the likelihood of word sequences.
- Mathematical formulation: Given a sequence of words $W = (w_1, w_2, \ldots, w_n)$, a language model estimates $\mathrm{P}(W) = P(w_1, w_2, \ldots, w_n)$

Chain rule decomposition: $\mathrm{P}(W) = P(w_1)P(w_2|w_1)P(w_3|w_1w_2)\, P(w_n|\, w_1, \ldots, w_{n-1})$



1-2 Language Models

Statistical vs. Neural Language Models

- Statistical Language Models (SLMs): A Statistical Language Model (SLM) is a probabilistic framework that assigns a probability to a sequence of words based on statistical patterns observed in a corpus. It estimates the likelihood of word sequences using mathematical techniques such as **N-grams**, **Hidden Markov Models (HMMs)**, and **Bayesian networks**. These models rely on the assumption that the probability of a word depends on a fixed number of previous words, making them computationally efficient but limited in capturing long-range dependencies.

- Neural Language Models (NLMs): A Neural Language Model (NLM) is a type of language model that uses neural networks to learn the probability distribution of word sequences. Unlike traditional statistical language models (e.g., N-gram models), which rely on explicit probability tables, NLMs leverage dense word embeddings and deep learning architectures (such as RNNs, LSTMs, and Transformers) to capture complex syntactic and semantic relationships in text.

  Key Features of NLMs:

  - Learn **context-aware** word representations.
  - Handle **long-range dependencies** better than statistical models.
  - **Reduce data sparsity issues** by using continuous vector representations.

1-2 Language Models

Applications of Language Models

- Text Generation: LLMs generate human-like text for creative writing, content creation, and storytelling (e.g., ChatGPT, Bard).
- Machine Translation: LLMs improve automatic translation by understanding context and nuances (e.g., Google Translate, DeepL).
- Summarization: LLMs condense long documents into concise summaries while preserving key information (e.g., academic papers, news articles).
- Chatbots & Virtual Assistants: AI-powered conversational agents assist users in answering queries and automating tasks (e.g., Siri, Alexa, customer support bots).
- Code Generation & Assistance: LLMs help developers by writing, explaining, and debugging code (e.g., GitHub Copilot, Code Llama).
- Information Retrieval & Question Answering: LLMs extract relevant answers from vast datasets and documents (e.g., Google Search, AI-powered FAQs).
- Medical & Legal Text Processing: LLMs assist professionals by analyzing documents, summarizing cases, and extracting insights (e.g., AI in medical diagnosis, legal contract analysis).
- etc.

1-3 Transformer: the Foundation of Language Models

Original Transformer Architecture (encoder-decoder); Vanilla Transformer

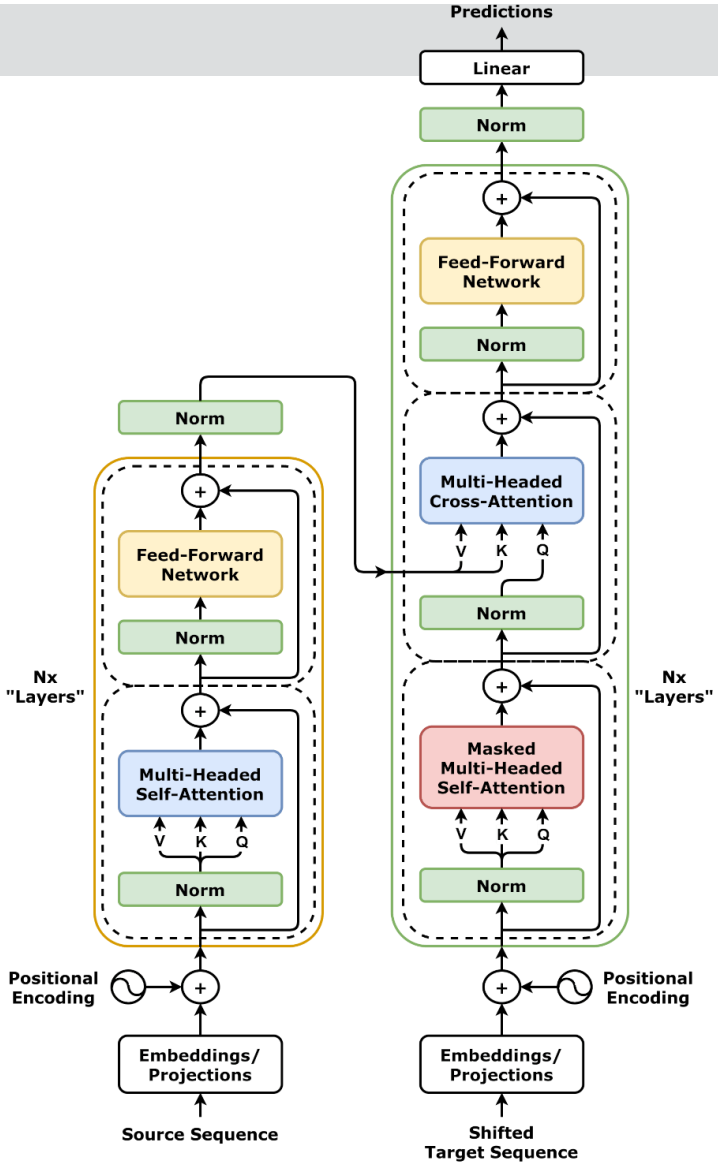
A deep learning architecture introduced in "Attention is All You Need" (Vaswani et al., 2017). Transformers are in many cases replacing convolutional and recurrent neural networks (CNNs and RNNs), as the most popular types of deep learning models just five years ago, enabling parallelization and long-range dependency modeling.

From RNNs to Transformers: Why the Shift?

- RNNs/LSTMs process sequences step-by-step, making them slow and hard to parallelize.
- They struggle with long-range dependencies and vanishing gradients.

⚡Motivation Behind Transformers

- Introduced self-attention to capture global context efficiently.
- Enabled parallel training (vs. sequential in RNNs).
- Addressed limitations of earlier models → better performance, scalability, and generalization.



1-3 Transformer: the Foundation of Language Models

Original Transformer Architecture (encoder-decoder); Vanilla Transformer (continued….)

Why Transformers Matter in NLP:

- Enable scalable training on large corpora.
- Excel in understanding contextual relationships between words.
- Serve as the foundation of modern LLMs like BERT, GPT, T5, etc.

A (original) **Transformer Language Model** is a type of neural network that uses the transformer architecture (primarily built on self-attention mechanisms) to process and understand language. It:

- Handles sequential data without relying on recurrence.
- Captures contextual relationships between words regardless of their position.
- Can be trained to perform various tasks like text generation, classification, translation, and more.

1-3 Transformer: the Foundation of Language Models

Original Transformer Architecture (encoder-decoder) – Vanilla Transformer (continued….)

How the original transformer language models works?

The original Transformer model, introduced in *"Attention is All You Need" (2017)*, uses an encoder-decoder architecture designed for sequence-to-sequence tasks like translation. Practically, it comprises 2 steps:

♠ **Encoder:**

- Takes in the input sentence (e.g., English).
- Processes and transforms it into a context-rich representation using self-attention and feedforward layers.

♠ **Decoder:**

- Takes the encoder's output and generates the target sentence (e.g., French).
- Uses self-attention to focus on previously generated words and cross-attention to focus on encoder outputs.

💡 Example:

Input: "How are you?" (to translate)
Encoder: Understands the full sentence.
Decoder: Generates "Comment vas-tu?" word-by-word, attending to both the input and what it has already produced.

Lecture Outline

Chapter 1. Introduction to Language and Language Models

Chapter 2. Types of LLM Architectures

Chapter 3. Core Mathematical & Statistical Concepts

2-1 Types of Language Models

There are **three** main types of Transformer models:

1. Encoder-Only Models

- Examples: BERT, RoBERTa, DistilBERT
- Architecture: Uses only the encoder stack of the Transformer.
- Purpose: designed for understanding tasks (e.g., classification, named entity recognition, sentence similarity).

2. Decoder-Only Models

- Examples: GPT, GPT-2, GPT-3, GPT-4
- Architecture: uses only the decoder stack.
- Purpose: Designed for text generation and autoregressive tasks.

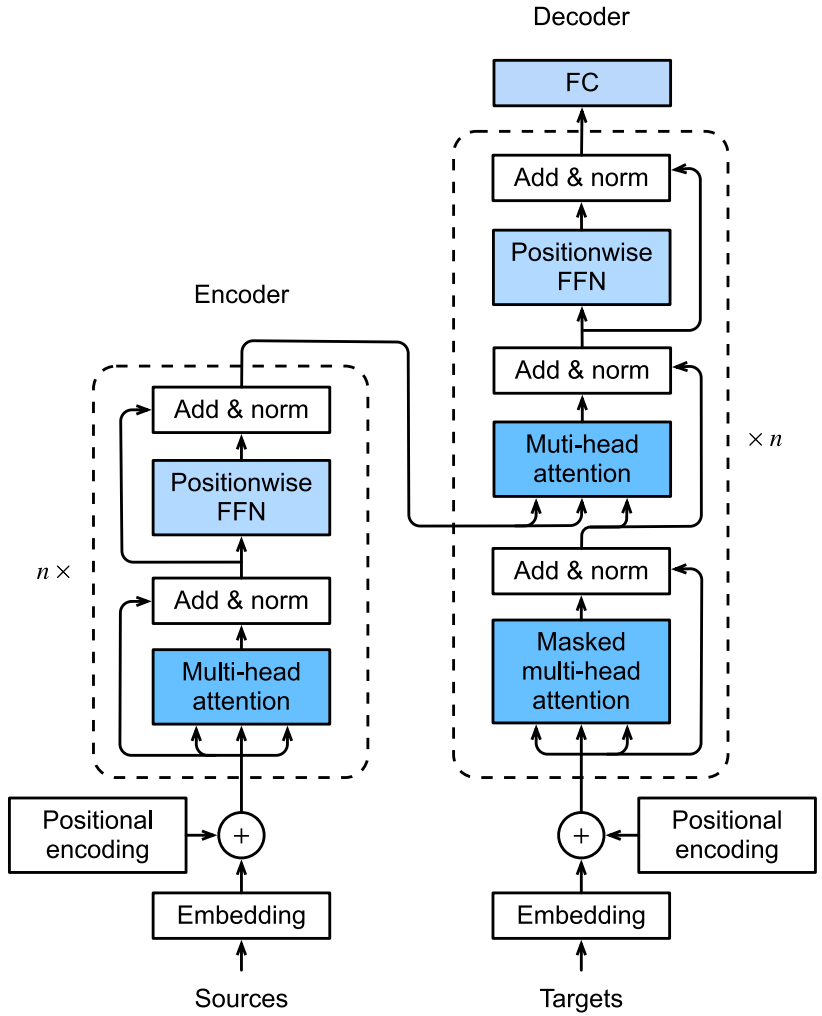
3. Encoder-Decoder or Vanilla Transformer (Seq2Seq) Models

- Examples: T5, BART, mT5, MarianMT
- Architecture: uses both encoder and decoder stacks.
- Purpose: Designed for sequence-to-sequence tasks (mapping input to output).

2-2 Encoder-Only Transformer

Encoder-only models represent a specialized variant of the transformer architecture, focusing solely on the task of understanding and encoding input sequences. Let's delve into the intricacies of encoder-only models and understand how they are transforming the landscape of AI and natural language processing.

At the core of an encoder-only model lies the mission of extracting meaningful context from input sequences. This context-rich representation serves as a valuable resource for downstream tasks that require a thorough understanding of the input's semantics and nuances. Unlike traditional encoder-decoder models that engage in both understanding and generation, encoder-only models specialize in contextual encoding, which can be immensely beneficial in tasks such as sentiment analysis, named entity recognition, and more.



2-2 Encoder-Only Transformer

What is an Encoder-Only Transformer?

- This architecture uses only the encoder stack* of the original Transformer.
- It processes the entire input sequence simultaneously, capturing context from both left and right (bidirectional).
- It does not generate text, but rather encodes input into contextual embeddings useful for classification, tagging, and retrieval tasks.

Key Characteristics:

- Bidirectional attention enables deep understanding of sentence context.
- Stack of self-attention layers followed by feed-forward layers and normalization.
- Trained using masked language modeling (MLM) objectives to learn context.

Popular Models:

- BERT: First major encoder-only model, pretrained on MLM.
- RoBERTa: Robustly optimized version of BERT with more training data and no NSP.
- ALBERT, DistilBERT, DeBERTa: Variants improving efficiency, performance, or scalability.

* The encoder stack means a series of identical layers in the Transformer that process the input step by step. Each layer has two main parts:
  - Self-attention: helps the model focus on relevant words in the input.
  - Feed-forward network: processes and transforms the attended information.

Stacking several of these layers allows the model to build a deeper understanding of the input.

2-3 Decoder-Only Transformers

Decoder-Only Transformers are a type of transformer architecture designed specifically for **autoregressive** language modeling, where the goal is to generate text one token at a time, based only on the previously generated tokens. Unlike full Transformer models that include both an encoder and a decoder, the decoder-only model uses just the decoder stack, equipped with causal (unidirectional) self-attention; meaning each token can only attend to earlier tokens in the sequence, not future ones.

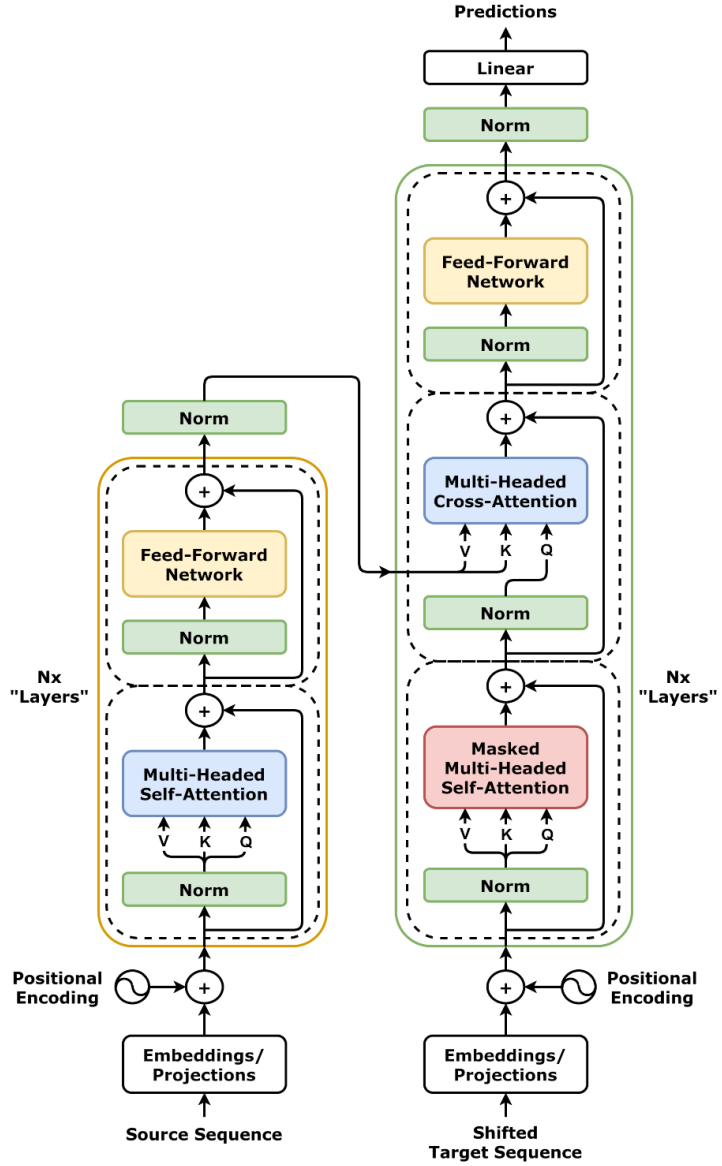
🔷 Definition: Autoregressive Model (Mathematical Form)

An **autoregressive (AR) model** is a type of model where the **output at time step *t*** depends on the **previous outputs**.

➤ General Form (Time Series / Sequence Modeling)

$$x_t = f(x_{t-1}, x_{t-2}, \ldots, x_{t-p}) + \epsilon_t$$

- x_t: value at time t
- p: number of past time steps used
- f: usually linear (in classical AR models), or nonlinear (in neural nets)
- ϵ_t: noise term



2-3 Decoder-Only Transformers

What's Being "Decoded" in Decoder-Only Transformers?

In decoder-only models, the model "decodes" the output sequence step by step from just the previous context; this could be a prompt, an initial sentence, or even an empty string. Instead of decoding from an encoded input (like in translation), it's generating (i.e., decoding) new tokens based on what it has generated so far.
In this sense:

- The decoder is still doing its job: turning representations into output tokens.
- But the input is implicit; it's the previous tokens in the same sequence (i.e., the left-side context).

► So, it's more accurate to think of it as:

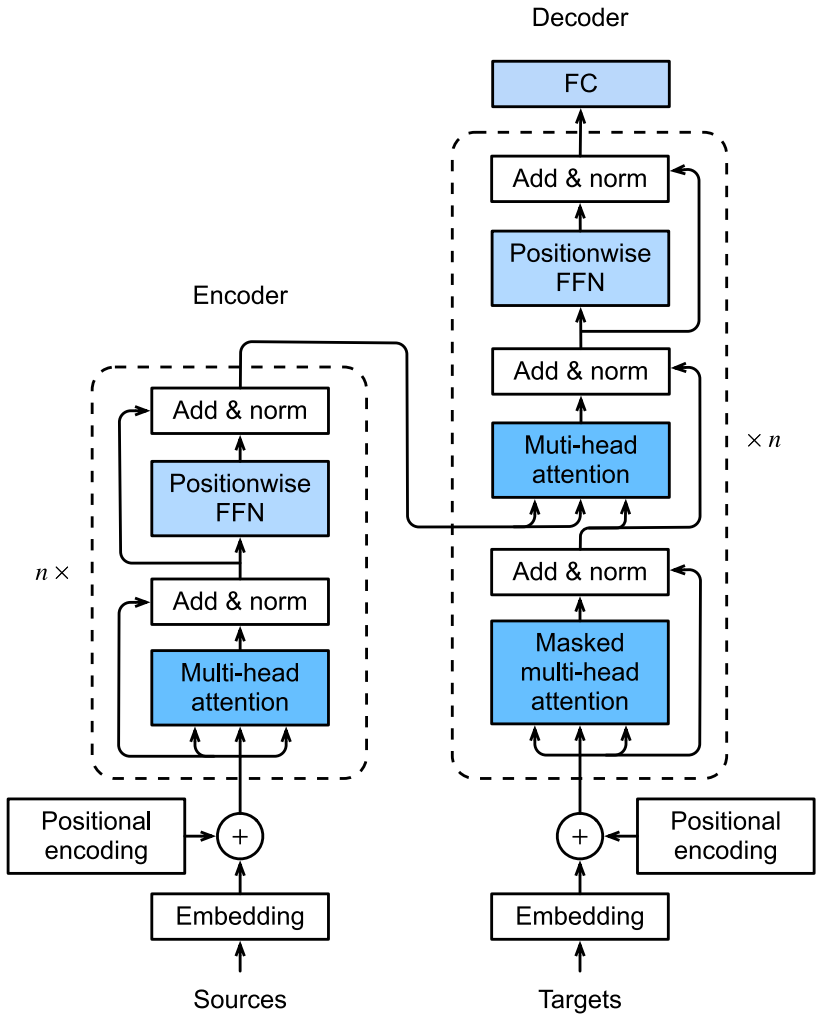
A decoder that decodes from previously generated tokens; not from an external encoder.

This is why in models like GPT, the entire model is called a decoder-only transformer, even though it might feel like it's "just generating" rather than decoding in the traditional sense.

2-3 Decoder-Only Transformers

Decoder-only models, like those in the GPT series, are Transformer-based models that use only the decoder part of the architecture for text generation. They are trained to predict the next word in a sequence, making them well-suited for tasks like language generation and translation. Unlike encoder-decoder models, they don't encode input information first; instead, they directly process the input sequence through the decoder.

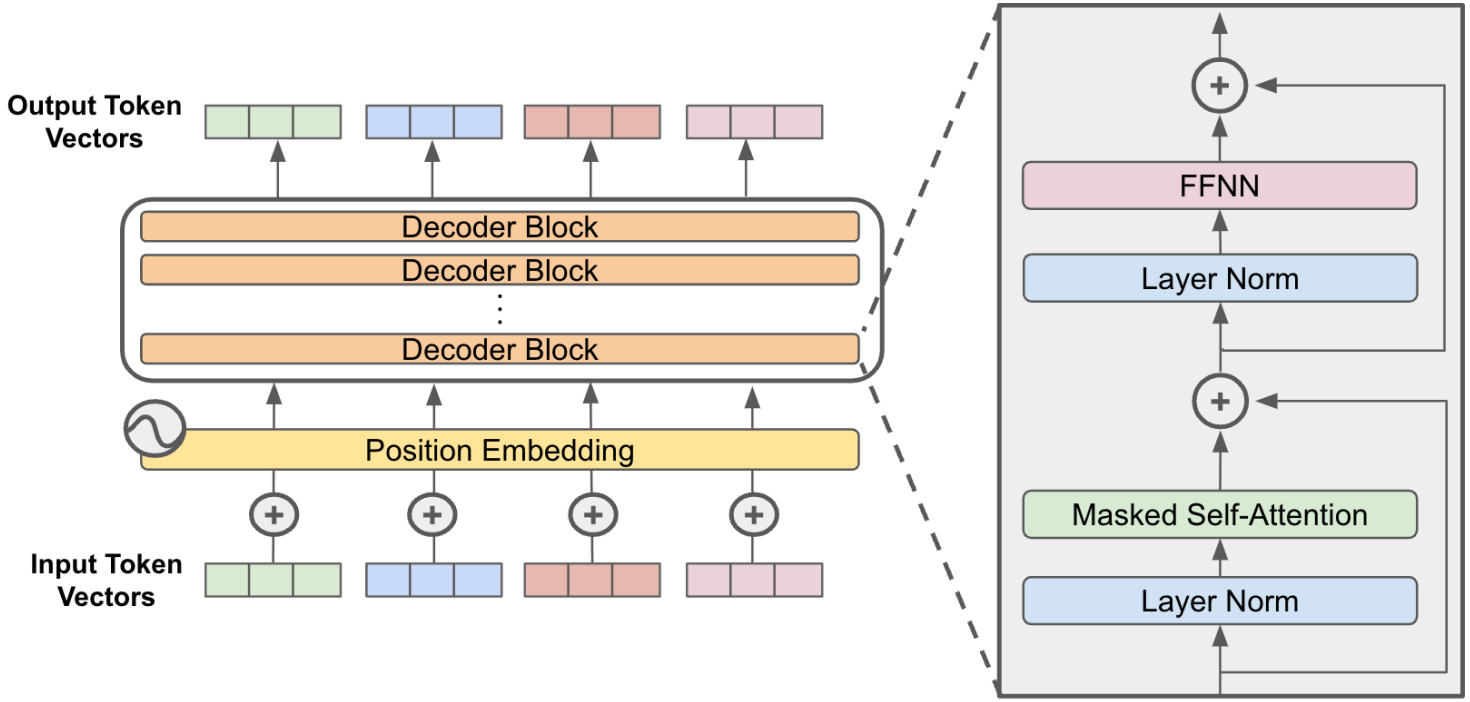
Decoder-only transformers, like GPT models, are designed purely for causal (left-to-right) language modeling. The model architecture is based on the transformer decoder stack from the original Transformer paper; but used without an encoder component.



2-3 Decoder-Only Transformers

The decoder stack structure

The **decoder stack** is the backbone of decoder-only transformer models (like GPT). It's a sequence of identical layers that process input tokens autoregressively; meaning, each token is generated by attending only to the previous ones.



2-3 Decoder-Only Transformers

The decoder stack structure (continued….)

Each decoder layer contains **three main components**:

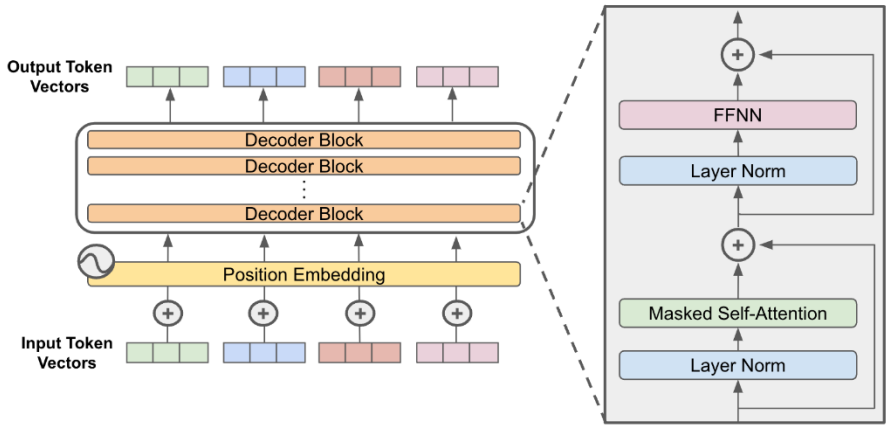
1. Masked Multi-Head Self-Attention (MHSA)
- Allows the model to attend to **previous tokens only** (via causal masking).
- Helps the model learn dependencies between tokens across different positions.
- Uses **multiple attention heads** to capture various contextual patterns.

2. Feed-Forward Neural Network (FFN)
- A fully connected network applied to each position separately.
- Consists of two linear layers with a non-linear activation (e.g., GELU or ReLU) in between.

3. Layer Normalization + Residual Connections
- Each sublayer is wrapped with residual connections and followed by layer normalization:

$$x = \text{LayerNorm}(x + \text{Sublayer}(x))$$

- This stabilizes training and helps the model converge faster.



Typical Stack Depth:
- A decoder-only model usually includes **12 to 96 layers**, depending on model size (e.g., GPT-2 vs. GPT-3).
- The output of each layer becomes the input to the next layer.

Final Output:
- After passing through the full stack, the final hidden state is projected to the vocabulary size using a **linear layer + softmax**, to predict the next token.

2-3 Decoder-Only Transformers

Decoder-Only Transformer & Autoregressive Language Modelling

What is Autoregressive Language Modeling?

- Autoregressive modeling is a method where a model predicts the next token in a sequence based on the previous tokens.
- The model generates text one token at a time, always looking at what has come before, never at the future (causal constraint).

How Decoder-Only Transformers Enable It?

- Decoder-only Transformers are designed for autoregressive generation.
- Each layer uses causal self-attention, which masks future positions so that the model can't peek ahead.
- During both training and inference, the model builds the output step by step, conditioning each prediction on all previous tokens.

2-3 Decoder-Only Transformers

Applications of Decoder-Only Models

Decoder-only Transformer models (like GPT-2, GPT-3, GPT-4, and similar) are designed to generate text one token at a time, making them highly effective for a wide range of **natural language generation (NLG)** tasks. These models learn to understand and produce language in a coherent, context-aware, and flexible way.

► Core Capability: Text Generation

- Because decoder-only models are trained autoregressively, they predict the next token given all prior ones.
- This makes them particularly powerful in free-form generation and tasks where output is not strictly aligned with input.

► Behind the Power: Autoregressive Generation

- Autoregressive decoding gives models strong control over output fluency and long-range coherence.

💡 Common Applications

1. Text Generation
2. Question Answering (QA)
3. Machine Translation
4. Summarization
5. Conversational Agents
6. Code Generation
7. Prompt-Based Tasks & In-Context Learning

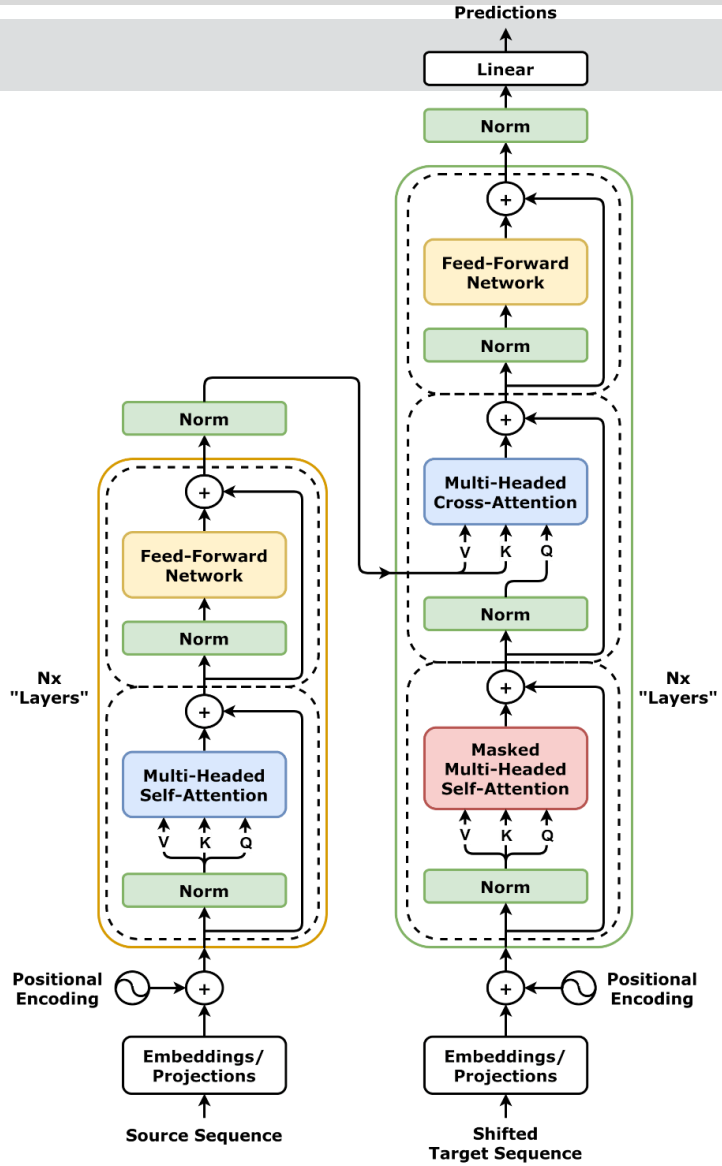
2-4 Encoder-Decoder Transformers (Vanilla/Seq2Seq Transformer)

Brief recap of encoder and decoder roles

The encoder-decoder architecture is a foundational design in **sequence-to-sequence (seq2seq) modeling**. It was originally introduced to solve tasks where the input and output are both sequences of varying lengths, such as machine translation, summarization, and dialog generation.

Motivation

Traditional models like RNNs and LSTMs struggled to handle long-range dependencies and parallel computation. The Transformer-based encoder-decoder model addressed these by introducing self-attention and positional encoding, allowing parallelism, richer contextual understanding, and better scalability.



2-4 Encoder-Decoder Transformers (Vanilla/Seq2Seq Transformer)

Brief recap of encoder and decoder roles (continued….)

Architecture Overview:

♦ **Encoder:**

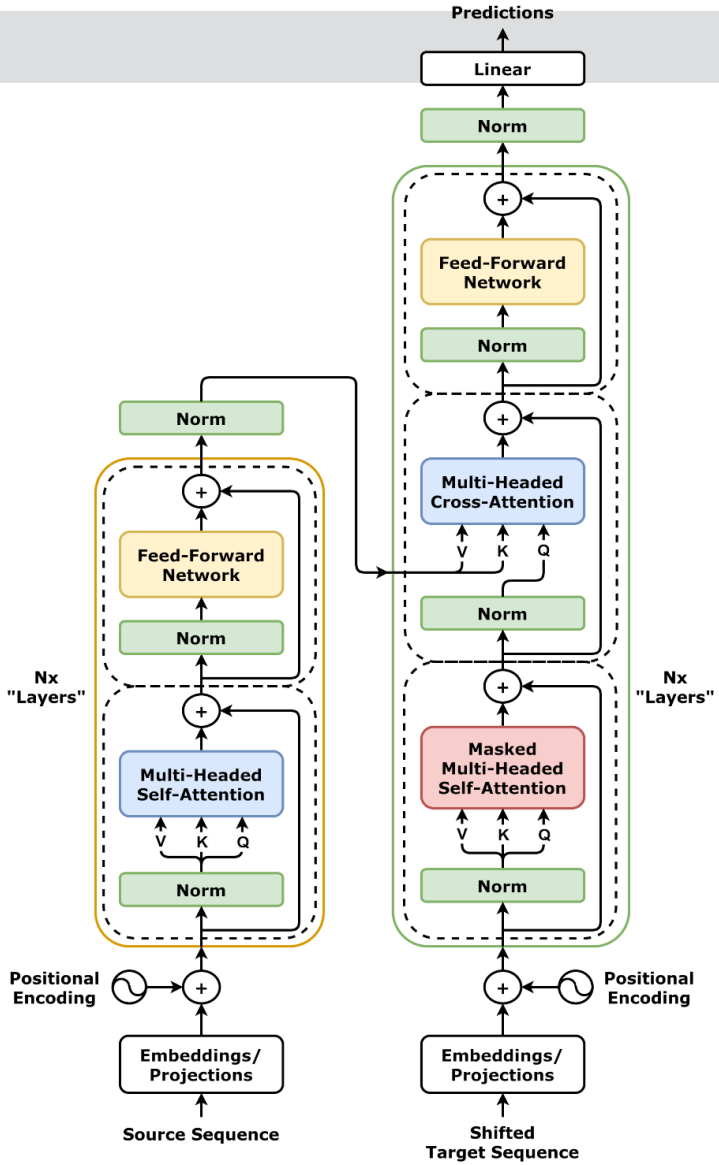
- Takes the input sequence (e.g., a sentence in English) and processes it into contextual embeddings.
- Uses self-attention layers to capture relationships between input tokens.
- Outputs a set of vectors representing the entire input sequence.

♦ **Decoder:**

- Generates the output sequence (e.g., the same sentence translated to French) one token at a time.
- Uses masked self-attention to prevent seeing future tokens during training.
- Incorporates encoder-decoder attention to access the encoded input representations.

♦ **Encoder-Decoder Attention:**

- Allows the decoder to "look at" relevant parts of the input when generating each output token.
- Crucial for tasks requiring alignment between input and output sequences.



2-4 Encoder-Decoder Transformers (Vanilla/Seq2Seq Transformer)

Brief recap of encoder and decoder roles (continued….)

Top encoder-decoder models

- **T5 (Text-to-Text Transfer Transformer):** Converts every NLP task into a text-to-text format, enabling a unified framework for classification, translation, and more.
- **BART (Bidirectional and Auto-Regressive Transformers):** Combines a bidirectional encoder (like BERT) and an autoregressive decoder (like GPT) for sequence-to-sequence tasks such as summarization and translation.
- **mT5 (Multilingual T5):** A multilingual variant of T5 trained on a large multilingual corpus, supporting dozens of languages for cross-lingual tasks.
- **mBART (Multilingual BART):** A multilingual sequence-to-sequence model designed for translation and cross-lingual generation tasks.
- **PEGASUS:** Optimized for abstractive summarization using gap-sentence generation as a pretraining objective.
- **MarianMT:** A fast, translation-focused model trained on many language pairs, widely used in machine translation systems.
- **UL2 (Unifying Language Learning):** Trained with a mixture of objectives (causal, masked, prefix), enabling flexible use in both encoder-decoder and decoder-only setups.

Lecture Outline

Chapter 1. Introduction to Language and Language Models

Chapter 2. Types of LLM Architectures

Chapter 3. Core Mathematical & Statistical Concepts

3 Core Mathematical & Statistical Concepts

Distributional Hypothesis

Distributional Hypothesis: "Words that occur in similar contexts tend to have similar meanings."

In sequence-to-sequence processing, this means that the neighborhood (context) of a word helps define its role and meaning.

Encoders & Contextual Embeddings

Transformer encoders capture this hypothesis by producing similar embeddings for tokens with similar surrounding contexts. Through self-attention, each token attends to its neighbors, allowing the model to generate rich, context-aware representations; essential for understanding meaning in tasks like translation, summarization, and question answering.

Example:

- I read the book.
- She finished the novel.

► "Book" and "novel" appear in similar contexts → similar meaning → similar embeddings.

Think to large amount of data!!

3 Core Mathematical & Statistical Concepts

Distributional Hypothesis (continued….)

The Distributional Hypothesis (that words used in similar contexts tend to have similar meanings) has inspired many language models, both classical and modern. Here's a list of key models based on this idea, along with short descriptions for each:

1. N-Gram Models

- Description: Predict the next word based on the previous n−1 words.
- Example: In a trigram model, the probability of "cake" in "chocolate birthday ___" is learned from how often that 3-word sequence appears.
- Distributional Basis: Learns word likelihoods from local context windows.

2. Hidden Markov Models (HMMs)

- Description: Probabilistic models that use hidden states (like parts of speech) to model sequences.
- Example: Predicts word sequences by modeling the transition probabilities between states (e.g., noun → verb).
- Distributional Basis: Learns transitions and emissions based on observed word co-occurrence and sequence patterns.

3. CBOW (Continuous Bag of Words)

- Description: Predicts a target word from surrounding context words.
- Example: Given "The ___ chased the mouse", predict "cat".
- Distributional Basis: Similar contexts produce similar word embeddings.

3 Core Mathematical & Statistical Concepts

Distributional Hypothesis (continued….)

4. Skip-Gram (Word2Vec)

- Description: Predicts surrounding context words from a target word.
- Example: given “cat”, predict nearby words like “pet”, “fur”, “meow”.
- Distributional Basis: Embeddings are trained so that similar words (contextually) are close in vector space.

5. Latent Semantic Analysis (LSA)

- Description: Matrix factorization technique that captures relationships between terms and documents based on co-occurrence.
- Distributional Basis: words with similar usage patterns across documents have similar vector representations.

6. GloVe (Global Vectors for Word Representation)

- Description: Learns word embeddings by factorizing a co-occurrence matrix (how often words appear together).
- Distributional Basis: embeddings reflect global word-word co-occurrence statistics..

7. FastText

- Description: Extends Word2Vec by representing words as bags of character n-grams.
- Distributional Basis: similar words share character n-gram structure and context, aiding generalization to rare or misspelled words.

Each of these models captures distributional properties in different ways (from simple co-occurrence to complex probabilistic transitions) and laid the foundation for modern neural language models like BERT and GPT.

3 Core Mathematical & Statistical Concepts

Encoder-Only transformers

1. Tokenization and Input Representation

Text is first tokenized into subword units (e.g., using WordPiece or Byte-Pair Encoding). Each token is mapped to a unique ID in a vocabulary.

- **Input Embedding**: Each token ID is converted into a dense vector (embedding) via an embedding matrix $E \in R^{V \times d}$, where V is the vocabulary size and d is the embedding dimension (e.g., 768 for BERT). For a token t_i the embedding is:

$$x_i = E[t_i]$$

- **Positional Encoding**: Since transformers don't inherently understand word order, positional encodings are added to the embeddings to encode token positions. These are typically fixed (sinusoidal) or learned vectors $P \in R^{L \times d}$, where L is the maximum sequence length:

$$x_i = x_i + P[i]$$

The input to the transformer is a matrix $X \in R^{n \times d}$, where n is the number of tokens in the sequence.

3 Core Mathematical & Statistical Concepts

Encoder-Only transformers

2. Transformer Encoder

The encoder consists of L identical layers, each with two main sub-components: multi-head self-attention and a feed-forward neural network. Layer normalization and residual connections are applied throughout.

a) Multi-Head Self-Attention: self-attention allows the model to weigh the importance of each token in the context of all others. For a single attention head:

- **Query, Key, and Value Vectors**: For input X, compute:

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

where $W_Q, W_K, W_V \in \mathbb{R}^{d \times d_h}$ are learned weight matrices, and $d_h = d/h$ is the dimension per head (h is the number of heads, e.g., 12 in BERT).

- **Scaled Dot-Product Attention**: Compute attention scores and weights:

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

The scaling factor $\sqrt{d_h}$ prevents large values in the dot product, stabilizing gradients.

3 Core Mathematical & Statistical Concepts

Encoder-Only transformers

- **Multi-Head Attention**: Concatenate outputs from h heads and project:

$$\text{MultiHead}(X) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W_O$$

where $W_O \in \mathbb{R}^{d \times d}$ is a projection matrix.

- **Residual Connection and Layer Normalization**: Add the input to the output (residual) and normalize:

$$X' = \text{LayerNorm}(X + \text{MultiHead}(X))$$

3 Core Mathematical & Statistical Concepts

Encoder-Only transformers

b) Feed-Forward Neural Network (FFN): Each token's representation is processed independently by a two-layer FFN:

$$\mathrm{FFN}(x) = W_2 \cdot \mathrm{ReLU}(W_1 x + b_1) + b_2$$

where $W_1 \in \mathbb{R}^{d \times d_{\text{ff}}}$, $W_2 \in \mathbb{R}^{d_{\text{ff}} \times d}$, and d_{ff} is the hidden dimension (e.g., 3072 in BERT). Residual connection and normalization are applied again:

$$X'' = \mathrm{LayerNorm}(X' + \mathrm{FFN}(X'))$$

This process repeats for all L layers, producing contextualized representations $X_L \in \mathbb{R}^{n \times d}$.

3. Output Processing

The final representations are used for downstream tasks. For example:

- **Classification (e.g., sentiment analysis)**: The representation of a special [CLS] token (added at the start of the sequence) is passed through a linear layer and softmax:

$$y = \mathrm{softmax}(W_{\text{cls}} x_{[\text{CLS}]} + b_{\text{cls}})$$

3 Core Mathematical & Statistical Concepts

Decoder-Only transformers

1. Autoregressive Function

The defining feature of decoder-only models is their autoregressive nature, where the probability of a sequence of tokens $w_1, w_2, \ldots w_n$ is modeled as the product of conditional probabilities:

$$P(w_1, w_2, \ldots, w_n) = \prod_{t=1}^{n} P(w_t | w_1, w_2, \ldots, w_{t-1})$$

- **Intuition**: At each step, the model predicts the next token w_t given the context of all prior tokens $w_1, w_2, \ldots w_{t-1}$. This is achieved through causal (masked) self-attention, ensuring the model only attends to previous positions.
- **Training Objective**: The model is trained to maximize the likelihood of the next token, minimizing the negative log-likelihood (cross-entropy loss):

$$\mathcal{L} = -\sum_{t=1}^{n} \log P(w_t | w_1, \ldots, w_{t-1})$$

- **Generation**: During inference, the model generates text autoregressively by sampling from $P(w_t | \, w_1, w_2, \ldots w_{t-1})$, often using techniques like greedy sampling, beam search, or top-k/top-p sampling.

Decoder-Only transformers

2. Input Representation

Similar to encoder-only models, text is tokenized into subword units (e.g., Byte-Pair Encoding) and mapped to token IDs.

- **Input Embedding**: Each token ID is converted into a dense vector (embedding) via an embedding matrix $E \in R^{V \times d}$, where V is the vocabulary size and d is the embedding dimension (e.g., 768 for BERT). For a token t_i the embedding is:

$$x_i = E[t_i]$$

- **Positional Encoding**: Since transformers don't inherently understand word order, positional encodings are added to the embeddings to encode token positions. These are typically fixed (sinusoidal) or learned vectors $P \in R^{L \times d}$, where L is the maximum sequence length:

$$x_i = x_i + P[i]$$

The input to the transformer is a matrix $X \in R^{n \times d}$, where n is the number of tokens in the sequence.

Decoder-Only transformers

2. Input Representation

Similar to encoder-only models, text is tokenized into subword units (e.g., Byte-Pair Encoding) and mapped to token IDs.

- **Input Embedding**: Each token ID is converted into a dense vector (embedding) via an embedding matrix $E \in R^{V \times d}$, where V is the vocabulary size and d is the embedding dimension (e.g., 768 for BERT). For a token t_i the embedding is:

$$x_i = E[t_i]$$

- **Positional Encoding**: Since transformers don't inherently understand word order, positional encodings are added to the embeddings to encode token positions. These are typically fixed (sinusoidal) or learned vectors $P \in R^{L \times d}$, where L is the maximum sequence length:

$$x_i = x_i + P[i]$$

The input to the transformer is a matrix $X \in R^{n \times d}$, where n is the number of tokens in the sequence.

3 Core Mathematical & Statistical Concepts

Decoder-Only transformers

3. Transformer Decoder

The decoder consists of L layers, each with two main sub-components: masked multi-head self-attention and a feed-forward neural network (FFN). Residual connections and layer normalization are applied throughout.

a) Masked Multi-Head Self-Attention: Decoder-only models use causal self-attention to ensure autoregressivity, preventing the model from attending to future tokens.

- **Query, Key, Value Vectors**: For input X, compute:

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

where $W_Q, W_K, W_V \in \mathbb{R}^{d \times d_h}$, and $d_h = d/h$ is the dimension per head (h is the number of heads, e.g., 12 or 16).

- **Causal Masking**: A mask is applied to the attention scores to prevent attending to future tokens. The attention score matrix $QK^T/\sqrt{d_h} \in \mathbb{R}^{n \times n}$ is masked such that position i only attends to positions $j \leq i$:

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

where M is a mask with $M_{ij} = 0$ if $j \leq i$ and $M_{ij} = -\infty$ otherwise, ensuring future tokens are ignored.

3 Core Mathematical & Statistical Concepts

Decoder-Only transformers

3. Transformer Decoder

a) Masked Multi-Head Self-Attention:

- **Multi-Head Attention**: Concatenate outputs from h heads:

$$\text{MultiHead}(X) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W_O$$

where $W_O \in \mathbb{R}^{d \times d}$.

- **Residual Connection and Layer Normalization**:

$$X' = \text{LayerNorm}(X + \text{MultiHead}(X))$$

3 Core Mathematical & Statistical Concepts

Decoder-Only transformers

3. Transformer Decoder

b) Feed-Forward Neural Network (FFN): Each token's representation is processed independently:

$$\mathrm{FFN}(x) = W_2 \cdot \mathrm{ReLU}(W_1 x + b_1) + b_2$$

where $W_1 \in \mathbb{R}^{d \times d_{\text{ff}}}$, $W_2 \in \mathbb{R}^{d_{\text{ff}} \times d}$, and d_{ff} is the hidden dimension (e.g., 2048 or 4096). Then:

$$X'' = \mathrm{LayerNorm}(X' + \mathrm{FFN}(X'))$$

This process repeats for L layers, producing contextualized representations $X_L \in \mathbb{R}^{n \times d}$.

3 Core Mathematical & Statistical Concepts

Decoder-Only transformers

4. Output Processingr Decoder

The final layer's representations are used to predict the next token:

- **Token Prediction**: The output for each position is projected to the vocabulary size:

$$p(w_t|w_1,\ldots,w_{t-1}) = \text{softmax}(W_{\text{vocab}}x_t + b_{\text{vocab}})$$

where $W_{\text{vocab}} \in \mathbb{R}^{d\times V}$.

- **Loss**: The cross-entropy loss compares the predicted distribution to the true next token:

$$\mathcal{L}_t = -\log p(w_t|w_1,\ldots,w_{t-1})$$

Main Goals!

- **At the end of this session, we are going to know that:**

G1: What is the language!

G2: What is the language model!

G3: What are the types of language models and what are their use cases?

G4: What is the main mathematical/statistical concept behind language models?

Questions!