Lecture 7.3: transformer

谢丹
清华大学数学系

2025/08

语言模型的基本任务

语言建模问题

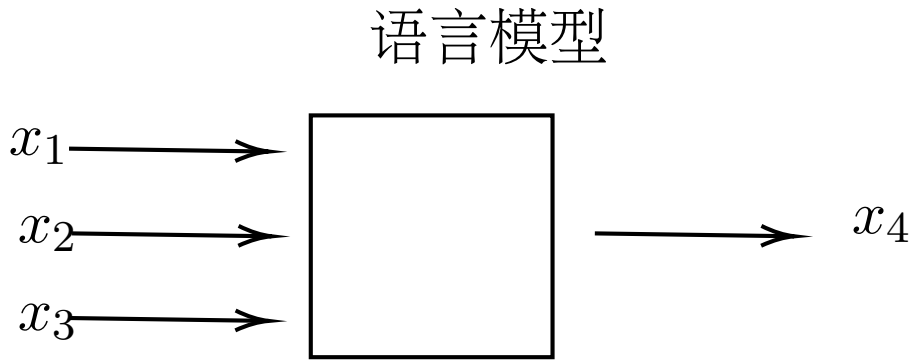
给定上下文，预测下一个最可能的词语：

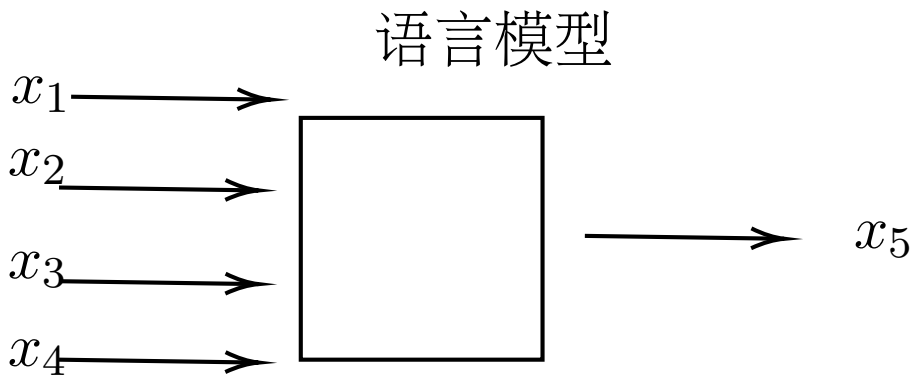
- *"I swam across the river to get to the other"* bank
- *"I walked across the road to get cash from the"* bank

技术挑战

- **长距离依赖**：需要捕捉远距离词语关系
- **多义性处理**：相同词语在不同上下文的含义（如"bank"）
- **复杂逻辑**：理解语法、常识和推理

一旦有了一个语言模型，就可以生成人类语言：





语言模型的数学表示

条件概率模型

语言模型的核心是计算条件概率 (auto regressive):

$$p(x_n|x_1,\ldots,x_{n-1})$$

其中给定前$n-1$个词，预测第n个词的概率分布。

示例

考虑词汇表：$\{\mathsf{I},\mathsf{can},\mathsf{do},\mathsf{it}\}$，给定上下文"I can do"：

$$\begin{aligned} p(\mathsf{it}|\mathsf{I},\mathsf{can},\mathsf{do}) &= 0.7 \\ p(\mathsf{I}|\mathsf{I},\mathsf{can},\mathsf{do}) &= 0.1 \\ p(\mathsf{can}|\mathsf{I},\mathsf{can},\mathsf{do}) &= 0.1 \\ p(\mathsf{do}|\mathsf{I},\mathsf{can},\mathsf{do}) &= 0.1 \end{aligned}$$

模型将预测下一个词为"it"（概率最高）。

Section 3: Transformer

What is a Tokenizer?

Core Purpose

A tokenizer breaks down raw text into smaller units (tokens) that the LLM can process.

Why We Need Tokenization:

- LLMs understand numbers, not text
- Vocabulary size must be manageable
- Handle rare words efficiently
- Process variable-length sequences

Common Approaches:

- **Word-based**: "hello" "world"
- **Character-based**: "h" "e" "l" "l" "o"
- **Subword**: "hello" "world" (most common)

Key Insight

Tokenization is the bridge between human language and numerical representations that models understand.

See the tokenizer of OpenAI:

https://platform.openai.com/tokenizer.

What is BPE Tokenizer?

Definition

Byte Pair Encoding (BPE) is a **subword tokenization** algorithm that iteratively merges the most frequent pairs of bytes/characters to create a vocabulary of subword units.

Key Idea:

- Start with character-level vocabulary
- Learn merges from training data
- Create subword units
- Balance vocabulary size and coverage

Why BPE?

- Handles unknown words
- Compact vocabulary
- Cross-lingual capability
- Used in GPT, RoBERTa, etc.

Core Insight

BPE finds the optimal balance between word-level (too many rare words) and character-level (too long sequences) tokenization.

BPE Algorithm: Training Phase

Step 1: Initialize Vocabulary

Start with all individual characters (bytes) in the training corpus

Step 2: Count Frequency

Count all adjacent pairs in the current vocabulary

Step 3: Merge Most Frequent

Merge the most frequent pair into a new token

Step 4: Repeat

Continue until target vocabulary size is reached or no more merges

Example

Training on: "low", "lower", "newest", "widest"

- Start: l, o, w, e, r, n, s, t, i, d
- Most frequent: 'e'+'s' $\rightarrow$ 'es'
- Next: 'es'+'t' $\rightarrow$ 'est'
- Continue...

BPE Training Example

Example

Training Corpus: "low lower newest widest"
Initial Vocabulary: l, o, w, e, r, n, s, t, i, d

Step-by-Step Merges

1. **Pairs:** lo:2, ow:2, we:1, er:2, ...
   Merge: 'l'+'o' → 'lo'
2. **Pairs:** low:2, we:1, er:2, ...
   Merge: 'lo'+'w' → 'low'
3. **Pairs:** e+r:2, ne:1, es:2, ...
   Merge: 'e'+'r' → 'er'
4. **Pairs:** es:2, st:2, ...
   Merge: 'e'+'s' → 'es'
5. **Pairs:** est:2, ...
   Merge: 'es'+'t' → 'est'

Final Vocabulary

- Characters: l, o, w, e, r, n, s, t, i, d
- Subwords: lo, low, er, es, est
- Words: lower, newest, widest

Result

Vocabulary: 16 tokens (instead of 4 words)
Coverage: Can handle "lowest" as "low" + "est"

BPE Tokenization Process

Encoding New Text

Given trained BPE merges, tokenize new text by applying merges greedily.

Example

Trained merges: 'l'+'o'→'lo', 'lo'+'w'→'low', 'e'+'s'→'es', 'es'+'t'→'est'

New word: "lowest"

Tokenization Steps:

1. Start: l, o, w, e, s, t
2. Apply: l o → lo
   Result: lo, w, e, s, t
3. Apply: lo w → low
   Result: low, e, s, t
4. Apply: e s → es
   Result: low, es, t
5. Apply: es t → est
   Result: low, est

Final Tokens:

- "low"
- "est"

Greedy Nature

BPE applies the longest possible merges first, which can sometimes be suboptimal but works well in practice.

BPE in Modern LLMs

GPT Series (OpenAI)

- BPE with byte-level encoding
- Handles all Unicode characters
- No unknown tokens
- Vocabulary: 50,000+ tokens

BERT (Google)

- WordPiece variant
- prefix for subwords
- Vocabulary: 30,000 tokens

Typical Vocabulary Sizes

- Small: 10,000-20,000 tokens
- Medium: 30,000-50,000 tokens
- Large: 50,000-100,000 tokens

SentencePiece (Google)

- BPE extension
- Language agnostic
- Handles raw text directly
- Used in T5, XLNet

RoBERTa (Facebook)

- BPE with byte-level
- Larger vocabulary
- Better performance

Advantages and Limitations

Advantages

- **No UNK**: Handles any word
- **Efficient**: Good sequence lengths
- **Multilingual**: Same algorithm
- **Morphology**: Captures word structure
- **Compact**: Smaller vocabularies

Limitations

- **Greedy**: Not always optimal
- **Ambiguity**: Multiple segmentations
- **Over-segmentation**: Common in some languages
- **Training cost**: Requires large corpus
- **Language bias**: Frequency-based

Example

Word: "unhappiness"
BPE: "un", "happi", "ness"
Alternative: "un", "happy", "ness" (but "happy" not in merges)

词嵌入 (Embedding)

从离散标记到连续向量

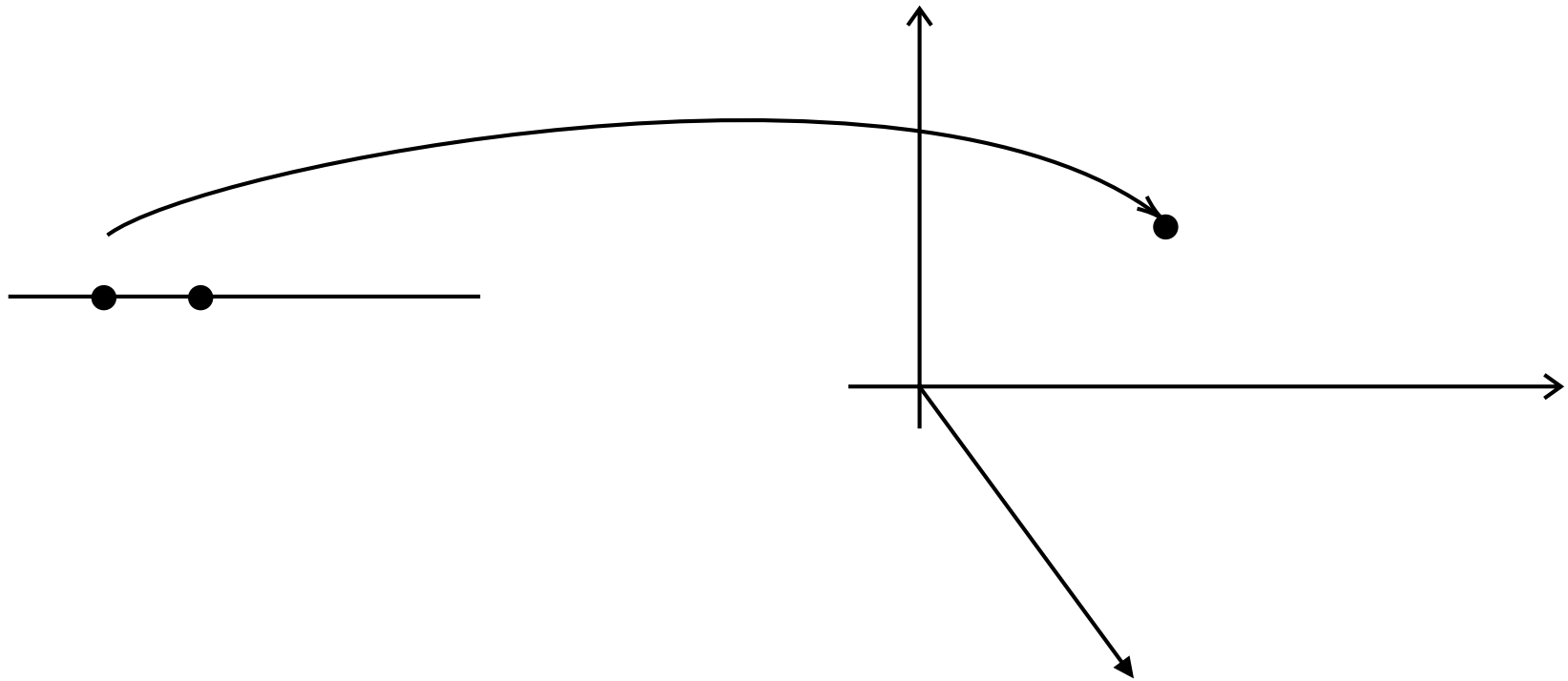
- **问题**：整数标记无法表达语义关系
- **解决方案**：将token映射到高维向量空间

关键特性

- **可学习性**：通过训练自动优化

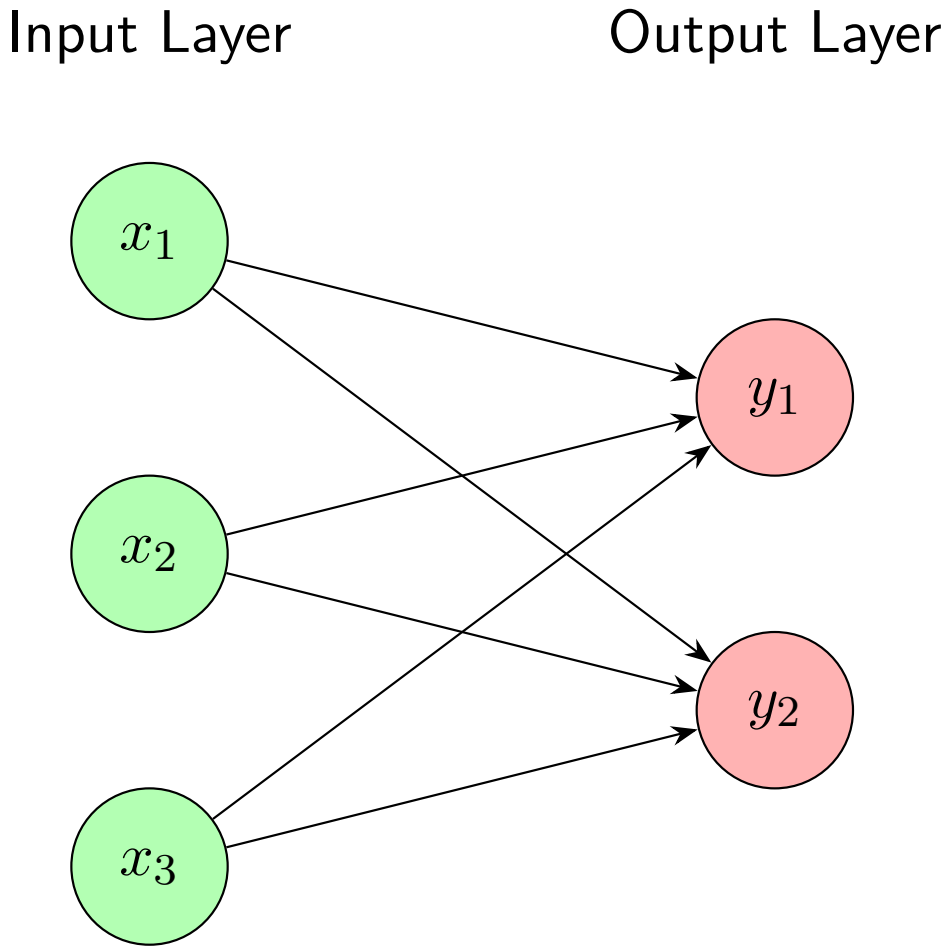
$$\text{embedding} : \mathbb{N} \to \mathbb{R}^d$$

- **语义保持**：相似词距离相近
- **维度可控**：典型维度$d \in [256, 1024]$



Embedding

The network for the embedding:



What are Embeddings in LLMs?

Core Concept

Embeddings transform discrete tokens into continuous vector representations that capture semantic meaning.

The Problem:

- Tokens are just IDs: [124, 567, 893]
- No inherent meaning
- No relationships between words
- Models need numerical features

The Solution:

- Learn dense vector representations
- Similar words have similar vectors
- Capture semantic relationships
- Enable mathematical operations

Example

"king" $\rightarrow$ [0.2, -0.5, 0.8, . . .]
"queen" $\rightarrow$ [0.3, -0.4, 0.7, . . .]
"apple" $\rightarrow$ [-0.8, 0.1, -0.2, . . .]

How Embeddings Work in LLMs

Technical Details:

- Lookup table: $E \in \mathbb{R}^{V \times d}$
- V: vocabulary size
- d: embedding dimension
- Each token maps to a row in E
- Learned during training

Key Properties:

- **Semantic Similarity**: "cat" $\approx$ "dog"
- **Analogies**: king - man + woman $\approx$ queen
- **Contextual**: Later layers add context
- **Dimensionality**: Typical d: 512-4096

Overview of Common Embedding Methods

What are Embedding Methods?

Techniques to represent discrete symbols (words, tokens) as continuous vectors in dense space.

Major Categories:

- **Static Embeddings**
  - Word2Vec
  - GloVe
  - FastText
- **Contextual Embeddings**
  - BERT
  - ELMo
  - Transformer-based

Key Evolution:

- Static → Contextual
- Shallow → Deep
- Fixed → Dynamic
- Word-level → Subword-level

Static Embedding Methods

Word2Vec (2013)

Approach:

- Skip-gram or CBOW
- Predict context words
- Shallow neural network

Properties:

- Fixed per word
- Captures analogies
- Efficient training

GloVe (2014)

Approach:

- Matrix factorization
- Global co-occurrence statistics
- Combines count-based + prediction

Properties:

- Fixed per word
- Good global semantics
- Fast inference

FastText (2016)

Approach:

- Subword information
- Character n-grams
- Handles OOV words

Properties:

- Morphological awareness
- Robust to spelling
- Multiple languages

Word2Vec: Overview

Core Idea

Learn word embeddings by training a simple neural network to predict words from their context.

Two Main Architectures:

- **Skip-gram**: Predict context words from target word
- **CBOW**: Predict target word from context words

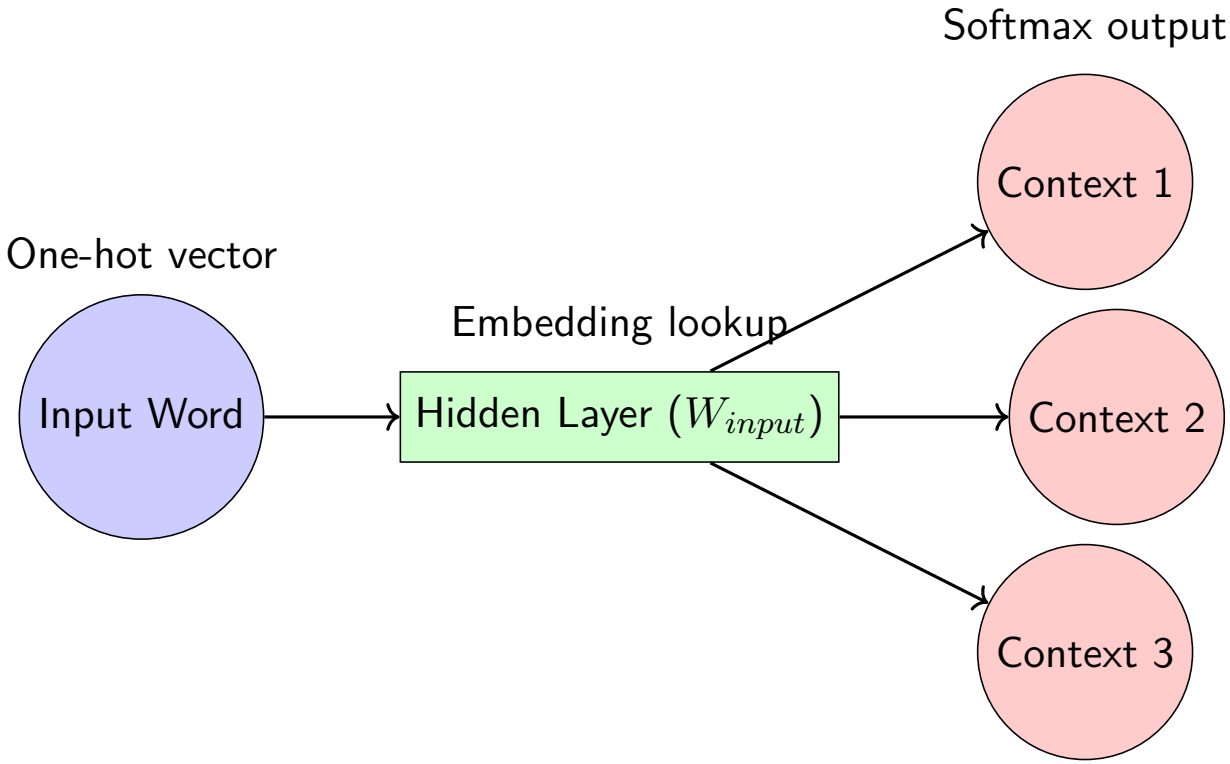
Key Innovation:

- Simple but effective
- No deep networks needed
- Captures semantic relationships
- Efficient training algorithms

Example

Task: Learn that "king" and "queen" are similar by seeing them in similar contexts.

Skip-gram Architecture



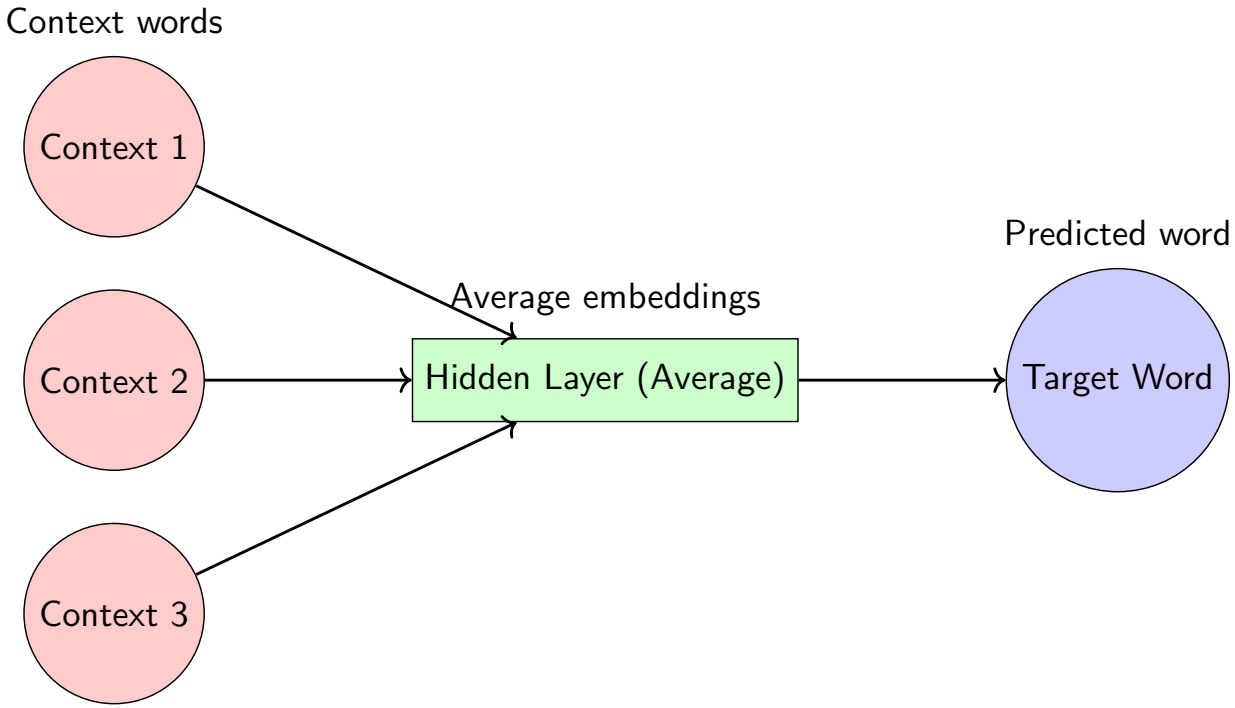
Training Objective

Given a target word, predict words that appear within a context window.

Example

For sentence: "The quick brown fox jumps"
Input: "brown", Context window size: 2
Target: predict ["quick", "fox"]

CBOW (Continuous Bag of Words) Architecture



Training Objective

Given context words, predict the target word in the middle.

Example

For sentence: "The quick brown fox jumps"
Context: ["quick", "fox"], Target: "brown"

Mathematical Formulation: Skip-gram

Objective Function

Maximize the average log probability of context words given target words:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t)$$

where:

- T: total number of training words
- c: context window size
- w_t: target word at position t
- w_{t+j}: context word

Probability Calculation

Using softmax:

$$p(w_O|w_I) = \frac{\exp(\mathbf{v}_{w_O}^\top \mathbf{v}_{w_I})}{\sum_{w=1}^{V}\exp(\mathbf{v}_w^\top \mathbf{v}_{w_I})}$$

where:

- $\mathbf{v}_{w_I}$: input vector of word w_I
- $\mathbf{v}_{w_O}$: output vector of word w_O
- V: vocabulary size

Modern LLM Embedding Approach

Transformer-based Embeddings

- **Input**: Subword tokens (BPE, SentencePiece)
- **Architecture**: Learned embedding matrix
- **Integration**: Combined with positional encoding
- **Training**: End-to-end with transformer

Key Features:

- **Contextual**: Depends on full sequence
- **Dynamic**: Changes with context
- **Deep**: Multiple layers of representation
- **Task-aware**: Fine-tuned for specific tasks

Advantages:

- Handles polysemy
- Captures complex syntax
- Transfer learning
- Multi-lingual capabilities

Attention Mechanism

Core Concept

注意力机制有效解决了语言模型中的长距离依赖问题，其核心思想是为每个token计算与其他token的关联权重：

- 权重越大表示关系越重要
- 权重通过训练自动学习

示例分析

句子："I swam across the river to get to the other bank"

	I	swam	across	the	river	to	get	to	the	other
bank		0.2			0.5		0.1			

- "river"和"swam"获得最高权重（0.5和0.2）
- 模型成功捕捉语义关联

Basic Attention Mechanism

Mathematical Formulation

注意力机制计算值的加权和，权重由查询-键的兼容性决定：

输入矩阵：

- 查询 $Q \in \mathbb{R}^{n \times d_k}$
- 键 $K \in \mathbb{R}^{m \times d_k}$
- 值 $V \in \mathbb{R}^{m \times d_v}$

维度说明：

- n: 查询数量
- m: 键值对数量
- d_k: 查询/键维度
- d_v: 值维度

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{1}$$

The softmax is along the row direction. Check: the dimension of the matrix of the output.

$$\underset{Q}{\begin{Bmatrix} q_{11} & q_{12} & q_{13} & \cdot & \cdot & \cdot & q_{1p} \\ q_{21} & q_{22} & q_{23} & \cdot & \cdot & \cdot & q_{2p} \\ q_{31} & q_{32} & q_{33} & \cdot & \cdot & \cdot & q_{3p} \\ \cdot & & & \cdot & & & \\ \cdot & & & & \cdot & & \\ \cdot & & & & & \cdot & \\ q_{n1} & q_{n2} & q_{n3} & \cdot & \cdot & \cdot & q_{np} \end{Bmatrix}} \times \underset{K^T}{\begin{Bmatrix} k_{11} & k_{21} & k_{31} & \cdot & \cdot & \cdot & k_{n1} \\ k_{12} & k_{22} & k_{32} & \cdot & \cdot & \cdot & k_{n2} \\ k_{13} & k_{23} & k_{32} & \cdot & \cdot & \cdot & k_{n2} \\ \cdot & & & \cdot & & & \\ \cdot & & & & \cdot & & \\ \cdot & & & & & \cdot & \\ k_{1p} & k_{2p} & k_{3p} & \cdot & \cdot & \cdot & k_{np} \end{Bmatrix}} = \underset{C/\sqrt{d}}{\begin{Bmatrix} c_{11} & c_{12} & c_{13} & \cdot & \cdot & \cdot & c_{1n} \\ \cdot & \cdot & & & & & \\ \cdot & & \cdot & & & & \\ \cdot & & & \cdot & & & \\ \cdot & & & & \cdot & & \\ \cdot & & & & & \cdot & \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{Bmatrix}} \Longrightarrow \underset{Softmax(.)}{\begin{Bmatrix} 0.9 & 0 & 0 & \cdot & \cdot & \cdot & 0.1 \\ \cdot & \cdot & & & & & \\ \cdot & & \cdot & & & & \\ \cdot & & & \cdot & & & \\ \cdot & & & & \cdot & & \\ \cdot & & & & & \cdot & \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{Bmatrix}}$$

Figure: Attention score

Self-Attention

Special Case of Attention

自注意力中查询、键、值同源:

$$\text{SelfAttention}(X) = \text{Attention}(XW^Q, XW^K, XW^V) \quad (2)$$

缩放因子

$\frac{1}{\sqrt{d_k}}$的作用:

- 防止点积结果过大
- 避免softmax进入梯度饱和区

Attention Mechanism

Let X be the input sequence: an $n \times d_{\text{model}}$ matrix.

The weight matrices W^Q, W^K, and W^V are $d_{\text{model}} \times d_k$ matrices, which serve as **learnable parameters**.

The attention mechanism processes an input sequence of tokens and produces a **weighted dependence matrix** as output. Each element in this matrix quantifies the **contextual relationship** between corresponding token pairs.

Multi-Head Attention (Part 1/2)

Core Mechanism

Given input $X \in \mathbb{R}^{n \times d_{\text{model}}}$:

1. **Project into multiple subspaces**:

$$
\begin{aligned}
Q_i &= XW_i^Q \quad (W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}) \\
K_i &= XW_i^K \quad (W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}) \\
V_i &= XW_i^V \quad (W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v})
\end{aligned}
$$

   for each head $i \in \{1, ..., h\}$

2. **Compute attention per head**:

$$
\text{head}_i = \text{softmax}\left(\frac{Q_i K_i^T}{\sqrt{d_k}}\right) V_i
$$

Why Multiple Heads?

- Each head learns different attention patterns
- Enables simultaneous focus on different positions

Recall the channels in the CNN network.

Multi-Head Attention (Part 2/2)

Output Composition

$$\text{MultiHead}(X) = \underbrace{[\text{head}_1; \ldots; \text{head}_h]}_{\text{Concatenation}} \underbrace{W^O}_{\text{Projection}} \qquad (W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}})$$

Implementation Details

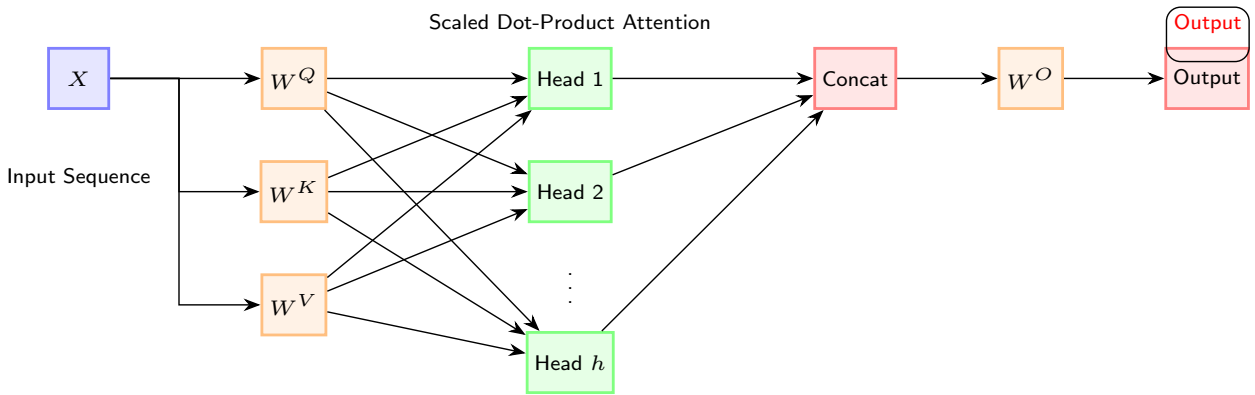
- Typical configuration: $h = 8$ heads
- Dimension allocation: $d_k = d_v = d_{\text{model}}/h$
- Computational complexity: $\mathcal{O}(n^2 \cdot d_{\text{model}})$

Counting parameters: the contribution of the multiple attention heads is

$$3 \times h \times d_{model} \times d_k$$

The concatenation contributes $h \times d_v \times d_model = d_k \times d_{model}$.

Multi-Head Attention Architecture



Positional Encoding in Transformers

The attention mechanism is symmetric in positions, we need to add position information.

Sinusoidal Positional Encoding

For position pos and $2i$ and $2i+1$ entries in model dimension d_{model}:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right) \tag{3}$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right) \tag{4}$$

This will produce a d_{model} dimensional vector.

- Fixed (non-learnable) encoding pattern
- Captures relative positions through sinusoidal frequencies
- Allows extrapolation to longer sequences

The input for the ith token is now a d_{model} dimensional vector derived from the embedding model, and the contribution of the position encoding.

Advanced Position Representations

Relative Position Representations

Modifies attention scores with relative positions:

$$e_{ij} = \frac{(x_i W^Q)(x_j W^K + \boldsymbol{a}_{ij}^K)^T}{\sqrt{d_k}} \tag{5}$$

where $\boldsymbol{a}_{ij}^K$ encodes relative position between tokens i and j.

- Captures pairwise distance relationships
- Learned embeddings for different offsets

Rotary Position Embedding (RoPE) Formulation

For a position m and embedding dimension d, define angles:

$$\theta_i = 10000^{-2i/d}$$

Rotation matrix R_m for position m:

$$R_m = \begin{pmatrix} \cos m\theta_i & -\sin m\theta_i \\ \sin m\theta_i & \cos m\theta_i \end{pmatrix}$$

We get a $d \times d$ dimensional matrix.
Applied to queries/keys:

$$\text{Attention}(m, n) = (R_m q)^\top (R_n k) = q^\top R_{n-m} k$$

Key Property

Attention scores depend only on **relative position** $(n - m)$.

Masked Self-Attention Formulation

For the purpose of text generation, we only need to know the attention score before a given token.

Core Equations

Given input $\boldsymbol{X} \in \mathbb{R}^{n \times d_{\text{model}}}$:

1. Project to queries, keys, values:

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}^{Q}, \quad \boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^{K}, \quad \boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^{V}$$

2. Compute masked attention:

$$\text{MaskedAttention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\underbrace{\frac{\boldsymbol{Q}\boldsymbol{K}^{T}}{\sqrt{d_k}}}_{\text{Scaled scores}} + \boldsymbol{M} \right) \boldsymbol{V}$$

Mask Matrix $\boldsymbol{M}$

$$\boldsymbol{M} = \begin{pmatrix} 0 & -\infty & -\infty & \cdots & -\infty \\ 0 & 0 & -\infty & \cdots & -\infty \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & -\infty \\ 0 & 0 & \cdots & 0 & 0 \end{pmatrix}$$

- Lower triangular structure
- Allows attention only to previous positions
- Critical for autoregressive generation

What is "Add & Norm"?

- **Core component** in Transformer layers (encoder/decoder).
- Combines two operations:
  1. **Add**: Residual connection (skip connection).
  2. **Norm**: Layer Normalization.
- Applied after **each sub-layer** (self-attention, FFN).

Purpose

Stabilize training and mitigate vanishing gradients.

Residual Connection

- **Operation**: Adds the sub-layer's input to its output.

$$\mathbf{x}_{\text{out}} = \mathbf{x} + \text{SelfAttention}(\mathbf{x})$$

- **Why?**
  - Preserves gradients (avoids vanishing gradients).
  - Allows deep networks to train efficiently.

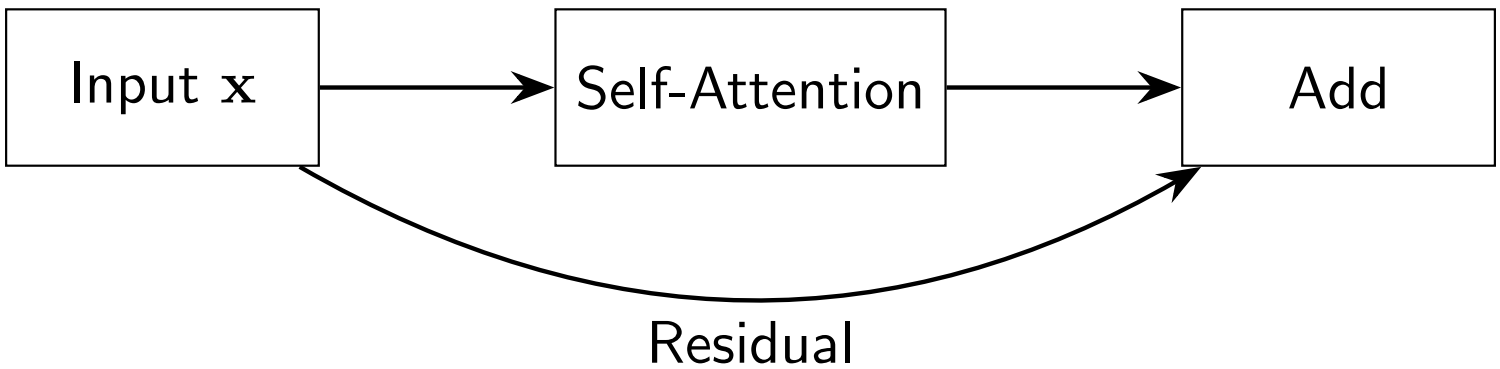


Figure: Residual connection in self-attention.

Layer Normalization

- **Operation**: Normalizes across the **feature dimension** (not batch).

$$\text{LayerNorm}(\mathbf{x}) = \gamma \left(\frac{\mathbf{x} - \mu}{\sqrt{\sigma^2 + \epsilon}} \right) + \beta$$

  - μ, σ^2: Mean/variance of $\mathbf{x}$.
  - γ, β: Learnable parameters.
  - ϵ: Small constant (e.g., 10^{-5}).
- **Why?**
  - Stabilizes activations (reduces internal covariate shift).
  - Faster convergence.

Where is "Add & Norm" Used?

- **After self-attention**:

  $\mathbf{x}' = \text{LayerNorm}(\mathbf{x}+\text{SelfAttention}(\mathbf{x}))$

- **After FFN**:

  $\mathbf{x}'' = \text{LayerNorm}(\mathbf{x}' + \text{FFN}(\mathbf{x}'))$

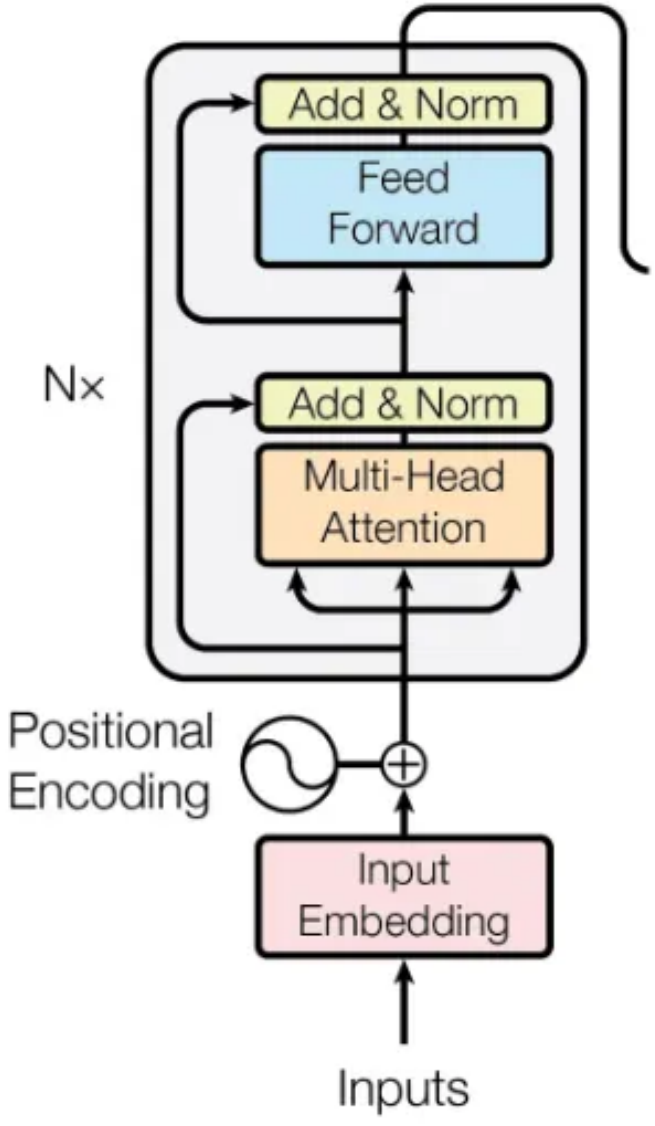
Feedforward Layer

The feedforward layer is a standard neural network with:

- Input dimension: d_{model}
- Output dimension: d_{model}
- (May include hidden layers)

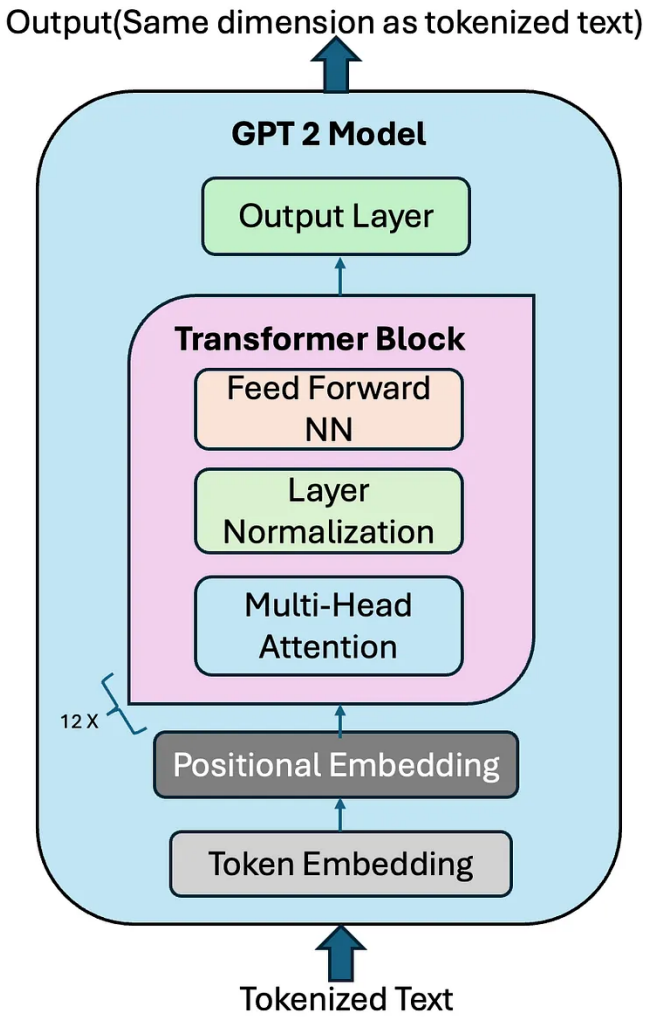
By combining the above components, we obtain a **transformer block**.

This block transforms an input tensor X of shape $n \times d_{\text{model}}$ into an output tensor $\tilde{X}$ of the **same shape**, which is why the architecture is called a "transformer."


Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Input Embedding
Inputs

We can stack many transformer blocks and at the end add a linear layer with K output, finally, we produce K probabilities using the softmax function. At the end, we produce a tensor of shape $n \times K$, with K the size of vocabularies.

GPT2 architecture



Training Process I

The training procedure operates as follows: given an input sequence of n tokens, the model produces n probability distributions over the vocabulary of size K.

Input-Output Mapping

$$[t_1, t_2, \ldots, t_n] \rightarrow [\text{out}_1, \text{out}_2, \ldots, \text{out}_n]$$

where each out_i predicts the **next token** in the sequence.

Training Targets

The target sequence is shifted by one position:

$$[t_2, t_3, \ldots, t_{n+1}]$$

Loss Function

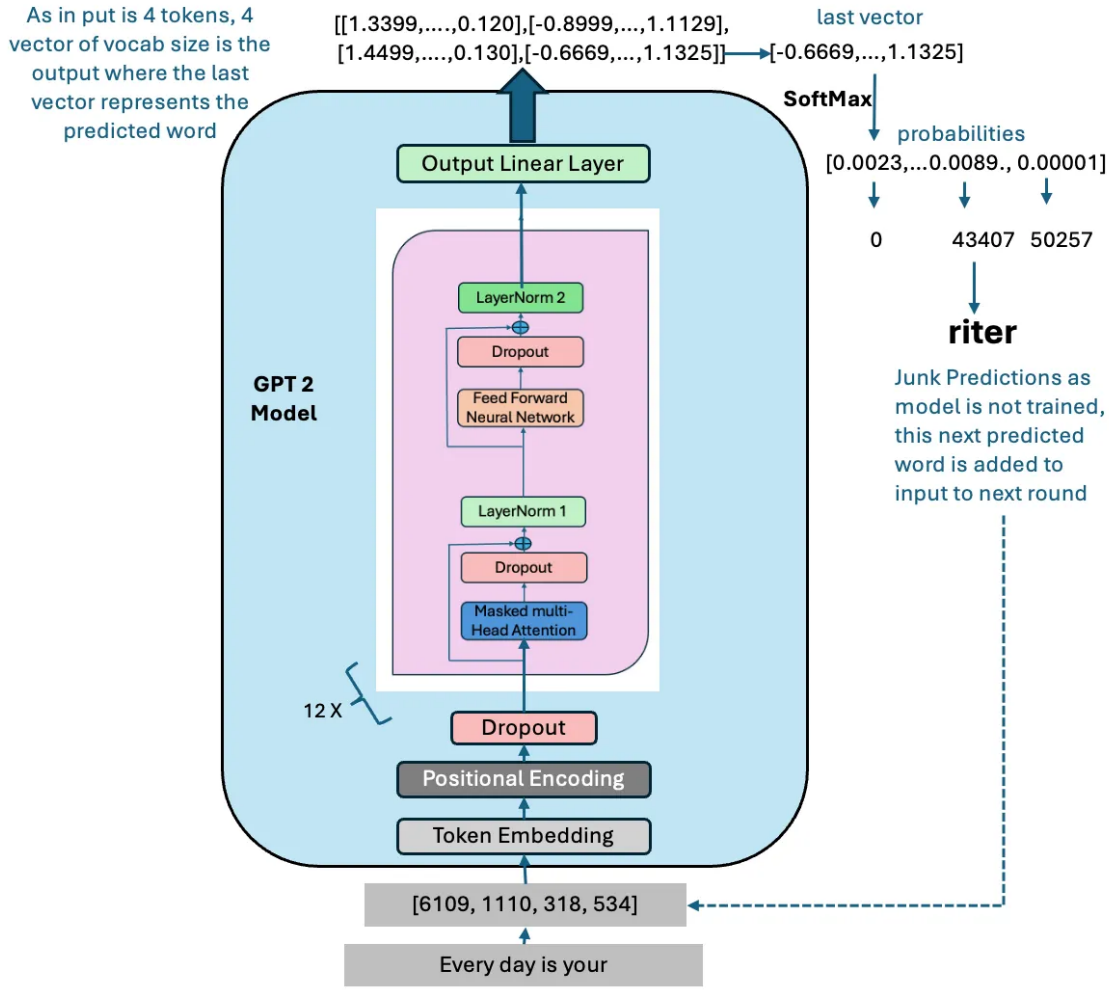
Training Process II

The loss is computed as the average cross-entropy across all positions:

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^{n} \text{CrossEntropy}(\text{out}_i, t_{i+1})$$

Prediction

Once the model is trained, one can use it to do the prediction of next token. Here is the summary: given the input of a sequence of words n, we get K probabilities, with K the size of vocabulary. We can then use it to make prediction for the next token.


As in put is 4 tokens, 4 vector of vocab size is the output where the last vector represents the predicted word
[[1.3399,....,0.120],[-0.8999,...,1.1129],
[1.4499,....,0.130],[-0.6669,...,1.1325]]
last vector
[-0.6669,...,1.1325]
SoftMax
probabilities
[0.0023,...0.0089., 0.00001]
0
43407
50257
riter
Junk Predictions as model is not trained, this next predicted word is added to input to next round
GPT 2 Model
Output Linear Layer
LayerNorm 2
Dropout
Feed Forward Neural Network
LayerNorm 1
Dropout
Masked multi-Head Attention
12 X
Dropout
Positional Encoding
Token Embedding
[6109, 1110, 318, 534]
Every day is your

What is Sampling in LLMs?

After a language model predicts probabilities for the next word, we need to choose which word to use.

The Challenge

How do we balance between:

- **Safe choices** (accurate but boring)
- **Creative choices** (interesting but risky)

Two main tools help us: **Temperature** and **Top-k sampling**

Temperature: The Creativity Control

What it does: Makes the model more predictable OR more creative

Low Temperature

T = 0.2

- More predictable
- Conservative
- Good for facts
- Can be repetitive

High Temperature

T = 1.5

- More creative
- Surprising
- Good for stories
- Can be random

Default: T = 1.0 (no change)

Temperature Example

Model's word probabilities:

Low Temp (T=0.2)

- **The: 85%**
- A: 10%
- Our: 5%

Almost always picks "The"

Normal (T=1.0)

- **The: 60%**
- A: 25%
- Our: 15%

Balanced choices

High Temp (T=1.5)

- **The: 40%**
- A: 35%
- Our: 25%

Much more variety

Remember

Temperature doesn't change which words are possible, just how likely they are to be chosen.

Top-k Sampling: The Quality Filter

What it does: Only considers the top k most likely words

1. Sort all words by probability
2. Keep only top k words
3. Choose from these k words

Top-k Examples

Model has 50,000 possible words

Small k (k=10)

Very selective

- Only considers 10 best words
- Very safe output
- Might miss good options

Large k (k=50)

More flexible

- Considers 50 best words
- More variety
- Might include bad options

Why use Top-k? Prevents the model from choosing terrible words that have very low probability.

How They Work Together

Typical Process:

1. **Model** calculates scores for all words
2. **Apply Temperature** to adjust creativity
3. **Apply Top-k** to remove bad options
4. **Choose** the final word

Real World Settings

- **Technical writing:** T=0.3, k=20
- **Creative writing:** T=0.8, k=50
- **Chatbot:** T=0.7, k=40

Quick Comparison

Setting	Temperature	Top-k
What it controls	Creativity vs Safety	Quality vs Variety
Low value	Predictable, boring	Very selective
High value	Creative, random	More options
Good for	Stories vs Facts	Removing bad words

Key Insight

Use **temperature** to control creativity, and **top-k** to ensure quality. They work best together!

Summary

Temperature

- Controls randomness
- Low T = safe choices
- High T = creative choices

Top-k

- Filters bad options
- Small k = very selective
- Large k = more variety

Practice Tip

Start with T=0.7 and k=40, then adjust based on your needs!

Result: Better control over your AI's writing style and quality.