

An Introduction to Transformers

UCSB LLM Seminar

Claire Murphy

UC Santa Barbara

August 14, 2026



Introduction

Transformers: An Overview

History

- 1 2014: Bahdanau et al. propose attention mechanism [BCB].
- 2 2017: Vaswani et al. use the attention mechanism to create a Transformer in the paper "Attention Is All You Need" [VSP].
- 3 2018: Liu et al. show that decoder-only Transformers work well for causal/autoregressive language modeling [LSP].

Transformers: An Overview

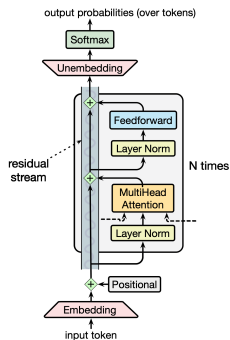
History

- 1 2014: Bahdanau et al. propose attention mechanism [BCB].
- 2 2017: Vaswani et al. use the attention mechanism to create a Transformer in the paper "Attention Is All You Need" [VSP].
- 3 2018: Liu et al. show that decoder-only Transformers work well for causal/autoregressive language modeling [LSP].

Today's goals:

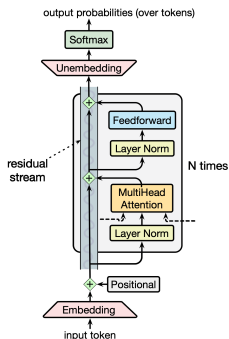
- 1 Understand the decoder-only attention model, following the approach by Jurafsky and Martin [JM].
- 2 Provide necessary vocabulary to read and understand "Attention is All You Need" [VSP].

How Do We Use Transformers?



Autoregressive LLM: Given a prompt, how do we predict the next token in the sequence?

How Do We Use Transformers?



Autoregressive LLM: Given a prompt, how do we predict the next token in the sequence?

- 1 Convert the prompt into an $N \times d$ dimensional matrix, where each row corresponds to a token.
- 2 Pass the prompt through several layers of **transformer blocks**; we obtain an $N \times d$ dimensional matrix.
- 3 Use the **language modeling head** to produce a probability distribution over all possible tokens. Use this distribution to predict the next token.

Review: Tokenization

Last week, we discussed that **tokens** are often short sequences of characters. For simplicity, this week we will assume that tokens are words.

Example:

- Existing structures:
 - Ordered dictionary: A dictionary containing all possible tokens in the prompt, $V = \{\text{and, giraffes, the, zebras}\}$.

Review: Tokenization

Last week, we discussed that **tokens** are often short sequences of characters. For simplicity, this week we will assume that tokens are words.

Example:

1 Existing structures:

- Ordered dictionary: A dictionary containing all possible tokens in the prompt, $V = \{\text{and, giraffes, the, zebras}\}$.
- **Embedding matrix** E : A $|V| \times d$ matrix that stores the embeddings of each word in the dictionary, e.g.

$$E = \begin{bmatrix} 5.448 & -0.153 & 0.751 & 0.012 \\ 0.273 & 2.894 & 0.955 & -0.412 \\ -0.370 & 5.865 & 1.224 & 0.243 \\ 0.092 & 0.472 & -0.595 & 4.753 \end{bmatrix}$$

Formally, two words are similar if the dot product of the corresponding rows of the embedding matrix is large.

Tokenization Continued

- 2 **Prompt:** The input to the LLM, containing N tokens, e.g. "the zebras and the giraffes"
- 3 Each word in the prompt can be encoded in terms of a **one-hot vector**, turning the prompt into an $N \times |V|$ matrix. If $V = \{\text{and, giraffes, the, zebras}\}$, then we can represent the prompt by

$$P := \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$$

Tokenization Continued

- 2 **Prompt**: The input to the LLM, containing N tokens, e.g. "the zebras and the giraffes"
- 3 Each word in the prompt can be encoded in terms of a **one-hot vector**, turning the prompt into an $N \times |V|$ matrix. If $V = \{\text{and, giraffes, the, zebras}\}$, then we can represent the prompt by

$$P := \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$$

- 4 Compute PE to obtain an $N \times d$ dimensional matrix. Each row equals the **embedding** of the corresponding token.

In summary, the actual input to the Transformer is an $N \times d$ dimensional matrix, where N is the number of tokens in the prompt.

Tokenization Continued

$$\begin{array}{c}
 \begin{array}{c} |V| \\
 \begin{array}{|c|}
 \hline
 0\ 0\ 0\ 0\ 1\ 0\ 0\ \dots\ 0\ 0\ 0\ 0 \\
 \hline
 0\ 0\ 0\ 0\ 0\ 0\ 0\ \dots\ 0\ 0\ 1\ 0 \\
 \hline
 1\ 0\ 0\ 0\ 0\ 0\ 0\ \dots\ 0\ 0\ 0\ 0 \\
 \hline
 \vdots \\
 \hline
 0\ 0\ 0\ 0\ 1\ 0\ 0\ \dots\ 0\ 0\ 0\ 0 \\
 \hline
 \end{array} \\
 N
 \end{array}
 \times
 \begin{array}{c}
 d \\
 \begin{array}{|c|}
 \hline
 E \\
 \hline
 \end{array} \\
 |V|
 \end{array}
 =
 \begin{array}{c}
 d \\
 \begin{array}{|c|}
 \hline
 \\
 \hline
 \end{array} \\
 N
 \end{array}
 \end{array}$$

Figure: We multiply the matrix P with rows containing one-hot vectors by the embedding matrix E . The resulting matrix is then passed into a transformer block.

Attention Is All You Need

(Simplified) Attention

Let $\mathbf{x}_j \in \mathbb{R}^d$ equal the token embedding of the j th token in the prompt.

- 1 For $j \leq i$, determine the similarity between the i th and j th token:

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

(Simplified) Attention

Let $\mathbf{x}_j \in \mathbb{R}^d$ equal the token embedding of the j th token in the prompt.

- 1 For $j \leq i$, determine the similarity between the i th and j th token:

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

- 2 For $j \leq i$, normalize the scores:

$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) := \frac{e^{\text{score}(\mathbf{x}_i, \mathbf{x}_j)}}{\sum_{\ell \leq i} e^{\text{score}(\mathbf{x}_i, \mathbf{x}_\ell)}}.$$

(Simplified) Attention

Let $\mathbf{x}_j \in \mathbb{R}^d$ equal the token embedding of the j th token in the prompt.

- 1 For $j \leq i$, determine the similarity between the i th and j th token:

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

- 2 For $j \leq i$, normalize the scores:

$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) := \frac{e^{\text{score}(\mathbf{x}_i, \mathbf{x}_j)}}{\sum_{\ell \leq i} e^{\text{score}(\mathbf{x}_i, \mathbf{x}_\ell)}}.$$

- 3 Compute weighted sum of embedding vectors: $\mathbf{a}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{x}_j$.

(Simplified) Attention

Let $\mathbf{x}_j \in \mathbb{R}^d$ equal the token embedding of the j th token in the prompt.

- 1 For $j \leq i$, determine the similarity between the i th and j th token:

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

- 2 For $j \leq i$, normalize the scores:

$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) := \frac{e^{\text{score}(\mathbf{x}_i, \mathbf{x}_j)}}{\sum_{\ell \leq i} e^{\text{score}(\mathbf{x}_i, \mathbf{x}_\ell)}}.$$

- 3 Compute weighted sum of embedding vectors: $\mathbf{a}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{x}_j$.

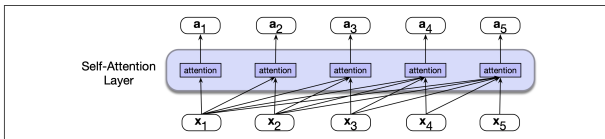


Figure 8.4 Information flow in causal self-attention. When processing each input $\mathbf{x}_i$, the model attends to all the inputs up to, and including $\mathbf{x}_i$.

Queries, Keys, and Values

Let $\mathbf{x}_j \in \mathbb{R}^d$ equal the token embedding of the j th token in the prompt.

- 1 For $j \leq i$, **score** $(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$.
- 2 For $j \leq i$, $\alpha_{ij} = \mathbf{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) := \frac{e^{\text{score}(\mathbf{x}_i, \mathbf{x}_j)}}{\sum_{\ell \leq i} e^{\text{score}(\mathbf{x}_i, \mathbf{x}_\ell)}}$.
- 3 $\mathbf{a}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{x}_j$.

Notice that we use the token embeddings in three different ways:

- 1 **Query**: The query refers to our current token, i.e. $\mathbf{x}_i$.
- 2 **Key**: The keys are the token embeddings use to determine the scores.
- 3 **Value**: The values are the token embeddings use to compute the weighted sum $\mathbf{a}_i$.

Attention

We introduce 3 matrices to transform the token embeddings into queries, keys, and values.

- 1 $\mathbf{q}_i = \mathbf{x}_i W^Q$, where W^Q is $d \times d_k$.
- 2 For $j \leq i$, $\mathbf{k}_j = \mathbf{x}_j W^K$, where W^K is $d \times d_k$.
- 3 For $j \leq i$, $\mathbf{v}_j = \mathbf{x}_j W^V$, where W^V is $d \times d_v$.

Attention

We introduce 3 matrices to transform the token embeddings into queries, keys, and values.

- 1 $\mathbf{q}_i = \mathbf{x}_i W^Q$, where W^Q is $d \times d_k$.
- 2 For $j \leq i$, $\mathbf{k}_j = \mathbf{x}_j W^K$, where W^K is $d \times d_k$.
- 3 For $j \leq i$, $\mathbf{v}_j = \mathbf{x}_j W^V$, where W^V is $d \times d_v$.

The attention mechanism:

- 1 Determine the similarity between the query $\mathbf{q}_i$ and the j th key $\mathbf{k}_j$:

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

- 2 Normalize the scores:

$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{q}_i, \mathbf{k}_j)) := \frac{e^{\text{score}(\mathbf{q}_i, \mathbf{k}_j)}}{\sum_{\ell \leq i} e^{\text{score}(\mathbf{q}_i, \mathbf{k}_\ell)}}.$$

- 3 Compute weighted sum of embedding vectors: $\mathbf{a}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{v}_j$.
- 4 $\mathbf{h}_i = \mathbf{a}_i W^O$, where W^O is $d_v \times d$ matrix.

Attention: A Worked Example

Recall $V = \{\text{and, giraffes, the, zebras}\}$ and the prompt “the zebras and the giraffes,” so $\mathbf{x}_1, \dots, \mathbf{x}_5$ are the embeddings of “the,” “zebras,” “and,” “the,” “giraffes,” respectively.

Let’s compute attention for $i = 4$ (the second “the”), which attends to $j \leq 4$: “the,” “zebras,” “and,” “the.”

Attention: A Worked Example

Recall $V = \{\text{and, giraffes, the, zebras}\}$ and the prompt “the zebras and the giraffes,” so $\mathbf{x}_1, \dots, \mathbf{x}_5$ are the embeddings of “the,” “zebras,” “and,” “the,” “giraffes,” respectively.

Let’s compute attention for $i = 4$ (the second “the”), which attends to $j \leq 4$: “the,” “zebras,” “and,” “the.”

Suppose (for illustration, with toy $d_k = 4$, $d_v = 2$, rather than vectors derived from real learned W^Q, W^K, W^V) training has produced:

Token j	$\mathbf{k}_j$	$\mathbf{v}_j$
1: “the”	(0, 3, 0, 0)	(0.5, 0)
2: “zebras”	(4, 0, 0, 0)	(5, 1)
3: “and”	(0, −2, 0, 0)	(0, 0)
4: “the”	(0, 3, 0, 0)	(0.5, 0)

and query $\mathbf{q}_4 = (2, 0, 0, 0)$.

Attention: A Worked Example (continued)

1 Scores: $\text{score}(\mathbf{q}_4, \mathbf{k}_j) = \frac{\mathbf{q}_4 \cdot \mathbf{k}_j}{\sqrt{d_k}}$

$$\text{score}(\mathbf{q}_4, \mathbf{k}_1) = \frac{0}{2} = 0, \quad \text{score}(\mathbf{q}_4, \mathbf{k}_2) = \frac{8}{2} = 4,$$

$$\text{score}(\mathbf{q}_4, \mathbf{k}_3) = \frac{0}{2} = 0, \quad \text{score}(\mathbf{q}_4, \mathbf{k}_4) = 0.$$

Attention: A Worked Example (continued)

1 Scores: $\text{score}(\mathbf{q}_4, \mathbf{k}_j) = \frac{\mathbf{q}_4 \cdot \mathbf{k}_j}{\sqrt{d_k}}$

$$\text{score}(\mathbf{q}_4, \mathbf{k}_1) = \frac{0}{2} = 0, \quad \text{score}(\mathbf{q}_4, \mathbf{k}_2) = \frac{8}{2} = 4,$$

$$\text{score}(\mathbf{q}_4, \mathbf{k}_3) = \frac{0}{2} = 0, \quad \text{score}(\mathbf{q}_4, \mathbf{k}_4) = 0.$$

2 Normalize (softmax):

$$\alpha_{4,1} \approx 0.017, \quad \alpha_{4,2} \approx \mathbf{0.948}, \quad \alpha_{4,3} \approx 0.017, \quad \alpha_{4,4} \approx 0.017.$$

"The" attends almost entirely to "zebras."

Attention: A Worked Example (continued)

1 Scores: $\text{score}(\mathbf{q}_4, \mathbf{k}_j) = \frac{\mathbf{q}_4 \cdot \mathbf{k}_j}{\sqrt{d_k}}$

$$\text{score}(\mathbf{q}_4, \mathbf{k}_1) = \frac{0}{2} = 0, \quad \text{score}(\mathbf{q}_4, \mathbf{k}_2) = \frac{8}{2} = 4,$$

$$\text{score}(\mathbf{q}_4, \mathbf{k}_3) = \frac{0}{2} = 0, \quad \text{score}(\mathbf{q}_4, \mathbf{k}_4) = 0.$$

2 Normalize (softmax):

$$\alpha_{4,1} \approx 0.017, \quad \alpha_{4,2} \approx \mathbf{0.948}, \quad \alpha_{4,3} \approx 0.017, \quad \alpha_{4,4} \approx 0.017.$$

"The" attends almost entirely to "zebras."

3 Weighted sum:

$$\mathbf{a}_4 = \sum_{j \leq 4} \alpha_{4j} \mathbf{v}_j \approx 0.948 (5, 1) \approx (4.74, 0.95) \approx \mathbf{v}_2.$$

Attention: A Worked Example (continued)

1 Scores: $\text{score}(\mathbf{q}_4, \mathbf{k}_j) = \frac{\mathbf{q}_4 \cdot \mathbf{k}_j}{\sqrt{d_k}}$

$$\text{score}(\mathbf{q}_4, \mathbf{k}_1) = \frac{0}{2} = 0, \quad \text{score}(\mathbf{q}_4, \mathbf{k}_2) = \frac{8}{2} = 4,$$

$$\text{score}(\mathbf{q}_4, \mathbf{k}_3) = \frac{0}{2} = 0, \quad \text{score}(\mathbf{q}_4, \mathbf{k}_4) = 0.$$

2 Normalize (softmax):

$$\alpha_{4,1} \approx 0.017, \quad \alpha_{4,2} \approx \mathbf{0.948}, \quad \alpha_{4,3} \approx 0.017, \quad \alpha_{4,4} \approx 0.017.$$

"The" attends almost entirely to "zebras."

3 Weighted sum:

$$\mathbf{a}_4 = \sum_{j \leq 4} \alpha_{4j} \mathbf{v}_j \approx 0.948 (5, 1) \approx (4.74, 0.95) \approx \mathbf{v}_2.$$

Takeaway: The updated representation $\mathbf{a}_4$ for "the" now looks almost like the value vector for "zebras" – attention let token 4 "borrow" context from "zebras," which is exactly the kind of information the model needs to correctly predict the next token, "giraffes."

Multi-head Attention

Multi-head attention: $A \in \mathbb{N}$ separate attention heads, each specialized to represent different linguistic relationships between tokens. For $1 \leq c \leq A$,

1 Create queries, keys, and values:

- 1 $\mathbf{q}_i^c = \mathbf{x}_i W^{Qc}$, where W^{Qc} is $d \times d_k$.
- 2 For $j \leq i$, $\mathbf{k}_j^c = \mathbf{x}_j W^{Kc}$, where W^{Kc} is $d \times d_k$.
- 3 For $j \leq i$, $\mathbf{v}_j^c = \mathbf{x}_j W^{Vc}$, where W^{Vc} is $d \times d_v$.

2 Determine the similarity between the query $\mathbf{q}_i^c$ and the j th key $\mathbf{k}_j^c$:

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

3 Normalize the scores:

$$\alpha_{ij}^c = \text{softmax}(\text{score}(\mathbf{q}_i^c, \mathbf{k}_j^c)) := \frac{e^{\text{score}(\mathbf{q}_i^c, \mathbf{k}_j^c)}}{\sum_{\ell \leq i} e^{\text{score}(\mathbf{q}_i^c, \mathbf{k}_\ell^c)}}.$$

4 Compute weighted sum of embedding vectors: $\mathbf{a}_i^c = \sum_{j \leq i} \alpha_{ij}^c \mathbf{v}_j^c$.

$$\mathbf{h}_i = \begin{bmatrix} \mathbf{a}_i^1 & \dots & \mathbf{a}_i^A \end{bmatrix} \times W^O, \text{ where } W^O \text{ is } Ad_v \times d \text{ matrix.}$$

The Transformer Block

The Transformer Architecture

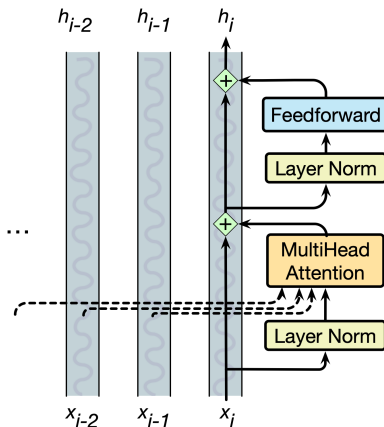


Figure: In the **residual stream** viewpoint, we consider the processing of an individual token i through the transformer block as a single stream of d -dimensional representations for token position i .

Transformer Blocks Continued

Layer Norm: We normalize the tokens. Define

$$\mu(\mathbf{x}) := \frac{1}{d} \sum_{k=1}^d x_k, \quad \sigma(\mathbf{x}) = \sqrt{\frac{1}{d} \sum_{k=1}^d (x_k - \mu)^2}$$

$\text{LayerNorm}(\mathbf{x}) = \gamma \left(\frac{\mathbf{x} - \mu}{\sigma} \right) + \beta$, where γ and β are learnable parameters.

Transformer Blocks Continued

Layer Norm: We normalize the tokens. Define

$$\mu(\mathbf{x}) := \frac{1}{d} \sum_{k=1}^d x_k, \quad \sigma(\mathbf{x}) = \sqrt{\frac{1}{d} \sum_{k=1}^d (x_k - \mu)^2}$$

$\text{LayerNorm}(\mathbf{x}) = \gamma \left(\frac{\mathbf{x} - \mu}{\sigma} \right) + \beta$, where γ and β are learnable parameters.

Feedforward Layer: A two-layer neural network. $\text{FFN}(\mathbf{x}) = \text{ReLU}(\mathbf{x}W_1 + b_1) W_2 + b_2$. Here, W_1 , W_2 , b_1 , and b_2 are learnable parameters.

Parallelization

A single transformer block accepts inputs $\{x_1, \dots, x_i\}$ and outputs h_i . Can we parallelize so a transformer block accepts inputs $\{x_1, \dots, x_N\}$ and outputs $\{h_1, \dots, h_N\}$?

Parallelization

A single transformer block accepts inputs $\{x_1, \dots, x_i\}$ and outputs h_i . Can we parallelize so a transformer block accepts inputs $\{x_1, \dots, x_N\}$ and outputs $\{h_1, \dots, h_N\}$?

We consider single-headed attention. Consider input X , an $N \times d$ matrix.

1 Create queries, keys, and values:

- 1 $Q = XW^Q$, where W^Q is $d \times d_k$.
- 2 $K = XW^K$, where W^K is $d \times d_k$.
- 3 $V = XW^V$, where W^V is $d \times d_v$.

Parallelization

A single transformer block accepts inputs $\{x_1, \dots, x_i\}$ and outputs h_i . Can we parallelize so a transformer block accepts inputs $\{x_1, \dots, x_N\}$ and outputs $\{h_1, \dots, h_N\}$?

We consider single-headed attention. Consider input X , an $N \times d$ matrix.

1 Create queries, keys, and values:

1 $Q = XW^Q$, where W^Q is $d \times d_k$.

2 $K = XW^K$, where W^K is $d \times d_k$.

3 $V = XW^V$, where W^V is $d \times d_v$.

2 Determine the similarity between corresponding rows of Q and K :

$$\text{score}(Q, K) = \frac{QK^T}{\sqrt{d_k}}.$$

This score is an $N \times N$ dimensional matrix.

Parallelization

A single transformer block accepts inputs $\{x_1, \dots, x_i\}$ and outputs h_i . Can we parallelize so a transformer block accepts inputs $\{x_1, \dots, x_N\}$ and outputs $\{h_1, \dots, h_N\}$?

We consider single-headed attention. Consider input X , an $N \times d$ matrix.

1 Create queries, keys, and values:

1 $Q = XW^Q$, where W^Q is $d \times d_k$.

2 $K = XW^K$, where W^K is $d \times d_k$.

3 $V = XW^V$, where W^V is $d \times d_v$.

2 Determine the similarity between corresponding rows of Q and K :

$$\text{score}(Q, K) = \frac{QK^T}{\sqrt{d_k}}.$$

This score is an $N \times N$ dimensional matrix.

3 Normalize the scores:

$$B_{ij} = \text{softmax}(\text{score}(Q, K))_{ij} := \frac{e^{\text{score}(Q, K)_{ij}}}{\sum_{\ell=1}^N e^{\text{score}(Q, K)_{i\ell}}}.$$

B is an $N \times N$ dimensional matrix.

Parallelization

A single transformer block accepts inputs $\{x_1, \dots, x_i\}$ and outputs h_i . Can we parallelize so a transformer block accepts inputs $\{x_1, \dots, x_N\}$ and outputs $\{h_1, \dots, h_N\}$?

We consider single-headed attention. Consider input X , an $N \times d$ matrix.

1 Create queries, keys, and values:

- 1 $Q = XW^Q$, where W^Q is $d \times d_k$.
- 2 $K = XW^K$, where W^K is $d \times d_k$.
- 3 $V = XW^V$, where W^V is $d \times d_v$.

2 Determine the similarity between corresponding rows of Q and K :

$$\text{score}(Q, K) = \frac{QK^T}{\sqrt{d_k}}.$$

This score is an $N \times N$ dimensional matrix.

3 Normalize the scores:

$$B_{ij} = \text{softmax}(\text{score}(Q, K))_{ij} := \frac{e^{\text{score}(Q, K)_{ij}}}{\sum_{\ell=1}^N e^{\text{score}(Q, K)_{i\ell}}}.$$

B is an $N \times N$ dimensional matrix.

4 Compute weighted sum of embedding vectors: $A = BV$. A is an $N \times d_v$ dimensional matrix.

Parallelization

A single transformer block accepts inputs $\{x_1, \dots, x_i\}$ and outputs h_i . Can we parallelize so a transformer block accepts inputs $\{x_1, \dots, x_N\}$ and outputs $\{h_1, \dots, h_N\}$?

We consider single-headed attention. Consider input X , an $N \times d$ matrix.

- 1 Create queries, keys, and values:

- 1 $Q = XW^Q$, where W^Q is $d \times d_k$.

- 2 $K = XW^K$, where W^K is $d \times d_k$.

- 3 $V = XW^V$, where W^V is $d \times d_v$.

- 2 Determine the similarity between corresponding rows of Q and K :

$$\text{score}(Q, K) = \frac{QK^T}{\sqrt{d_k}}.$$

This score is an $N \times N$ dimensional matrix.

- 3 Normalize the scores:

$$B_{ij} = \text{softmax}(\text{score}(Q, K))_{ij} := \frac{e^{\text{score}(Q, K)_{ij}}}{\sum_{\ell=1}^N e^{\text{score}(Q, K)_{i\ell}}}.$$

B is an $N \times N$ dimensional matrix.

- 4 Compute weighted sum of embedding vectors: $A = BV$. A is an $N \times d_v$ dimensional matrix.

- 5 $H = AW^O$, where W^O is $d_v \times d$ dimensional matrix.

Masking

Consider the computation

$$\text{score}(Q, K) = \frac{QK^T}{\sqrt{d_k}}.$$

Before parallelization, we only computed $\text{score}(q_i, k_j)$ if $j \leq i$. To recover this behavior, we use **masking**.

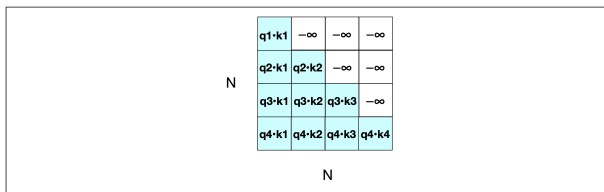
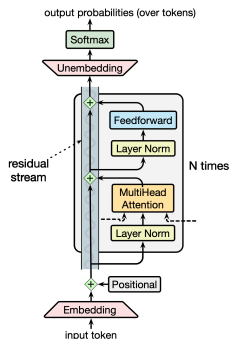


Figure 8.10 The $N \times N$ QK^T matrix showing the $q_i \cdot k_j$ values, with the upper-triangle portion of the comparisons matrix zeroed out (set to $-\infty$, which the softmax will turn to zero).

The Language Modeling Head

Motivation: The Language Modeling Head



Recall **autoregressive** LLM: Given a prompt, how do we predict the next token in the sequence?

- 1 Convert the prompt into an $N \times d$ dimensional matrix, where each row corresponds to a token.
- 2 Pass the prompt through several layers of **transformer blocks**; we obtain an $N \times d$ dimensional matrix.
- 3 Use the **language modeling head** to produce a probability distribution over all possible tokens. Use this distribution to predict the next token.

The Language Modeling Head

After passing the prompt through several layers of transformer blocks, we obtain N d -dimensional vectors $\{\mathbf{h}_1^L, \dots, \mathbf{h}_N^L\}$.

The Language Modeling Head

After passing the prompt through several layers of transformer blocks, we obtain N d -dimensional vectors $\{\mathbf{h}_1^L, \dots, \mathbf{h}_N^L\}$.

- 1 We only care about the final vector $\mathbf{h}_N^L$.

The Language Modeling Head

After passing the prompt through several layers of transformer blocks, we obtain N d -dimensional vectors $\{\mathbf{h}_1^L, \dots, \mathbf{h}_N^L\}$.

- 1 We only care about the final vector $\mathbf{h}_N^L$.
- 2 Compute the **logit vector** $\mathbf{u} = \mathbf{h}_N^L E^T$, a vector of length $|V|$.

The Language Modeling Head

After passing the prompt through several layers of transformer blocks, we obtain N d -dimensional vectors $\{\mathbf{h}_1^L, \dots, \mathbf{h}_N^L\}$.

- 1 We only care about the final vector $\mathbf{h}_N^L$.
- 2 Compute the **logit vector** $\mathbf{u} = \mathbf{h}_N^L E^T$, a vector of length $|V|$.
- 3 Perform a softmax function on the entries of $\mathbf{u}$ to obtain a $|V|$ -dimensional vector $\mathbf{y}$, a probability distribution on the elements of the dictionary.

The Language Modeling Head

After passing the prompt through several layers of transformer blocks, we obtain N d -dimensional vectors $\{\mathbf{h}_1^L, \dots, \mathbf{h}_N^L\}$.

- 1 We only care about the final vector $\mathbf{h}_N^L$.
- 2 Compute the **logit vector** $\mathbf{u} = \mathbf{h}_N^L E^T$, a vector of length $|V|$.
- 3 Perform a softmax function on the entries of $\mathbf{u}$ to obtain a $|V|$ -dimensional vector $\mathbf{y}$, a probability distribution on the elements of the dictionary.
- 4 **Sample** a token based on this probability distribution.

The Language Modeling Head: Diagram 1

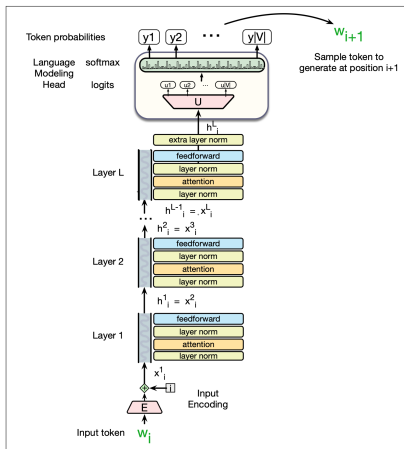


Figure 8.16 A transformer language model (decoder-only), stacking transformer blocks and mapping from an input token w_i to a predicted next token w_{i+1} .

The Language Modeling Head: Diagram 2

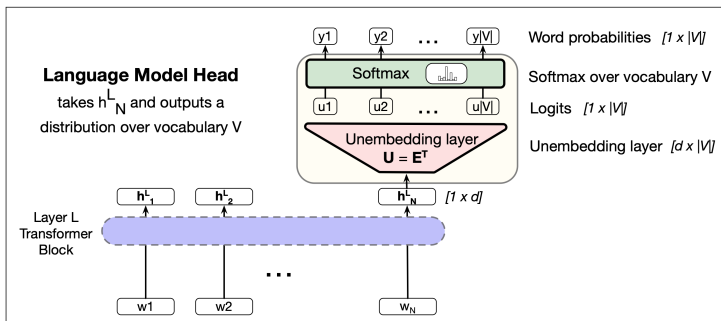


Figure 8.15 The language modeling head: the circuit at the top of a transformer that maps from the output embedding for token N from the last transformer layer (h_N^L) to a probability distribution over words in the vocabulary V .

Training

The Training Set

We are provided with pairs $\{(\underbrace{X^m}_{N \times d}, \underbrace{y^m}_{|V| \times 1})\}_{m=1}^M$.

- i th row of X^m is i th token embedding.
- $y_k^m = \begin{cases} 1 & (N+1)\text{th token is } k\text{th elt. of dictionary } V \\ 0 & \text{otherwise} \end{cases}$

Example

- 1 $V = \{\text{and, giraffes, the, zebras}\}.$
- 2 "The zebras and the" => "giraffes."

- 3 $X^m = \begin{bmatrix} -0.370 & 5.865 & 1.224 & 0.243 \\ 0.092 & 0.472 & -0.595 & 4.753 \\ 5.448 & -0.153 & 0.751 & 0.012 \\ -0.370 & 5.865 & 1.224 & 0.243 \end{bmatrix}.$

- 4 $y^m = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}.$

Training

We let Θ equal the set of all parameters in our LLM; Θ includes W^Q , W^K , W^V , W^O for each attention head and each transformer block, as well as the parameters involved in the Layer Norm and feedforward layer.

Training

We let Θ equal the set of all parameters in our LLM; Θ includes W^Q, W^K, W^V, W^O for each attention head and each transformer block, as well as the parameters involved in the Layer Norm and feedforward layer.

Define $F(X, \Theta)$ as the function that:

- Accepts an input embedding X and parameters Θ ,
- Outputs $\hat{\mathbf{y}} \in \mathbb{R}^{|V|}$, where the k th entry of $\hat{\mathbf{y}}$ equals the probability that the next token is the k th element of the dictionary V .

Training

We let Θ equal the set of all parameters in our LLM; Θ includes W^Q, W^K, W^V, W^O for each attention head and each transformer block, as well as the parameters involved in the Layer Norm and feedforward layer.

Define $F(X, \Theta)$ as the function that:

- Accepts an input embedding X and parameters Θ ,
- Outputs $\hat{\mathbf{y}} \in \mathbb{R}^{|V|}$, where the k th entry of $\hat{\mathbf{y}}$ equals the probability that the next token is the k th element of the dictionary V .

We seek

$$\hat{\Theta} := \operatorname{argmin}_{\Theta} \frac{1}{M} \sum_{m=1}^M L_{CE}(F(X^m, \Theta), \mathbf{y}^m).$$

We will use the cross-entropy loss function $L_{CE}(\hat{\mathbf{y}}^m, \mathbf{y}^m) := - \sum_{k=1}^{|V|} \mathbf{1}_{\{y_k^m=1\}} \ln(\hat{y}_k^m)$.

Training Continued

Define

$$G(\Theta) = -\frac{1}{M} \sum_{m=1}^M \sum_{k=1}^{|V|} \mathbf{1}_{\{y_k^m=1\}} \ln(F(X^m, \Theta)).$$

We seek

$$\hat{\Theta} := \underset{\Theta}{\operatorname{argmin}} G(\Theta)$$

Training Continued

Define

$$G(\Theta) = -\frac{1}{M} \sum_{m=1}^M \sum_{k=1}^{|V|} \mathbf{1}_{\{y_k^m=1\}} \ln(F(X^m, \Theta)).$$

We seek

$$\hat{\Theta} := \underset{\Theta}{\operatorname{argmin}} G(\Theta)$$

Attempt 1: **Gradient descent**.

$$\begin{aligned} \Theta_{t+1} &= \Theta_t - \eta \nabla_{\Theta} G(\Theta_t) \\ &= \Theta_t - \eta \frac{1}{M} \sum_{m=1}^M \sum_{k=1}^{|V|} \mathbf{1}_{\{y_k^m=1\}} \nabla_{\Theta} (\ln(F(X^m, \Theta))). \end{aligned}$$

Training Continued

Define

$$G(\Theta) = -\frac{1}{M} \sum_{m=1}^M \sum_{k=1}^{|V|} \mathbf{1}_{\{y_k^m=1\}} \ln(F(X^m, \Theta)).$$

We seek

$$\hat{\Theta} := \underset{\Theta}{\operatorname{argmin}} G(\Theta)$$

Attempt 1: **Gradient descent**.

$$\begin{aligned} \Theta_{t+1} &= \Theta_t - \eta \nabla_{\Theta} G(\Theta_t) \\ &= \Theta_t - \eta \frac{1}{M} \sum_{m=1}^M \sum_{k=1}^{|V|} \mathbf{1}_{\{y_k^m=1\}} \nabla_{\Theta} (\ln(F(X^m, \Theta))). \end{aligned}$$

Attempt 2: **Stochastic gradient descent**. At each time step, randomly select some m such that $1 \leq m \leq M$, then set

$$\Theta_{t+1} = \Theta_t - \eta \nabla_{\Theta} \left(\sum_{k=1}^{|V|} \mathbf{1}_{\{y_k^m=1\}} \ln(F(X^m, \Theta)) \right).$$

Conclusion (paraphrased from [JM])

An **autoregressive** LLM accepts a sequence of tokens and seeks to predict the next token.

Conclusion (paraphrased from [JM])

An **autoregressive** LLM accepts a sequence of tokens and seeks to predict the next token.

Transformers are networks based on **multi-head attention**. A multi-head attention computation takes an input vector $\mathbf{x}_i$ and maps it to an output $\mathbf{h}_i$ by adding in vectors from prior tokens, weighted by how relevant they are for the processing of the current word.

Conclusion (paraphrased from [JM])

An **autoregressive** LLM accepts a sequence of tokens and seeks to predict the next token.

Transformers are networks based on **multi-head attention**. A multi-head attention computation takes an input vector $\mathbf{x}_i$ and maps it to an output $\mathbf{h}_i$ by adding in vectors from prior tokens, weighted by how relevant they are for the processing of the current word.

A **transformer block** consists of a residual stream in which the input from the prior layer is passed up to the next layer, with the output of different components added to it.

Conclusion (paraphrased from [JM])

An **autoregressive** LLM accepts a sequence of tokens and seeks to predict the next token.

Transformers are networks based on **multi-head attention**. A multi-head attention computation takes an input vector $\mathbf{x}_i$ and maps it to an output $\mathbf{h}_i$ by adding in vectors from prior tokens, weighted by how relevant they are for the processing of the current word.

A **transformer block** consists of a residual stream in which the input from the prior layer is passed up to the next layer, with the output of different components added to it.

Transformer blocks are **stacked** to make deeper and more powerful networks.

Conclusion (paraphrased from [JM])

An **autoregressive** LLM accepts a sequence of tokens and seeks to predict the next token.

Transformers are networks based on **multi-head attention**. A multi-head attention computation takes an input vector $\mathbf{x}_i$ and maps it to an output $\mathbf{h}_i$ by adding in vectors from prior tokens, weighted by how relevant they are for the processing of the current word.

A **transformer block** consists of a residual stream in which the input from the prior layer is passed up to the next layer, with the output of different components added to it.

Transformer blocks are **stacked** to make deeper and more powerful networks.

Language models can be built out of stacks of transformer blocks, with a **language modeling head** at the top, which applies an unembedding matrix to the output $\mathbf{H}$ of the top layer to generate the logits, which are then passed through a softmax to generate word probabilities.

References

- BCB** Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate." ICLR, 2015.
- JM** Jurafsky, Daniel and James H. Martin. "Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, with Language Models." 3rd edition draft, 2026.
- LSP** Liu, Peter J., Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Łukasz Kaiser, and Noam Shazeer. "Generating Wikipedia by Summarizing Long Sequences." ICLR, 2018.
- VSP** Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention Is All You Need." NeurIPS, 2017.