

Self-Attention as the Core Prediction Algorithm in Large Language Models: A Linear Algebra and Geometric Perspective

Author Raymond Jonathan Dwi Putra Julianto - 13524059^{1,2}

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia

¹13524059@stei.itb.ac.id, ²annayuliati69@gmail.com

Abstract— Large Language Models (LLMs) have become a central technology in modern natural language processing, enabling machines to generate coherent and context-aware text. At the core of these models lies the self-attention mechanism, which allows tokens within a sequence to dynamically interact and contribute to contextual representation. This work examines self-attention as the core prediction algorithm in LLMs from a linear algebraic and geometric perspective.

Keywords— Self-Attention, Linear Algebra, Large Language Models, Transformer, Next-Token Prediction

I. INTRODUCTION

Automation technologies and Large Language Models (LLMs) have rapidly advanced and are increasingly adopted by industries to improve operational efficiency and reduce economic cost. Modern LLMs are developed for a wide range purposes, from general-purpose language understanding to highly specialized tasks. At the core of these models lies self-attention, a fundamental algorithm that enables the model to dynamically assign relevance to each token in a sequence when predicting the next token.

Unlike statistical or matrix-factorization approaches such as Latent Semantic Analysis (LSA), which analyze global co-occurrence patterns to identify similarity between texts, self-attention is capable of capturing fine-grained contextual relationships within a sentence. For example, although the sentences “He eat rice yesterday” and “Yesterday, he ate rice” contain similar words, their meanings and syntactic structures differ. Self-attention distinguishes these differences by incorporating positional information and computing token interactions through geometric operations in high-dimensional vector spaces.

Self-attention, also referred to as intra-attention, is a mechanism that evaluates the positional and contextual relationships among tokens within a single sequence. This mechanism has proven highly effective for numerous natural language processing tasks, including reading comprehension, abstractive summarization, textual entailment, and learning task-independent sentence

representations. As a result, self-attention has become a critical component in modern LLM architecture due to its ability to model contextual meaning dynamically.

This paper examines the role of self-attention as the core algorithm responsible for next-token prediction in LLMs. A simplified demonstration is provided to illustrate how self-attention processes input sequences and generates predictions based on token relevance. Due to computational limitations, the implementation presented in this work does not match the scale or capabilities of real-world LLMs, but it captures the essential linear algebraic and geometric principles underlying their operation.

II. LITERATURE REVIEW

A. Matrix

A matrix is a fundamental concept in linear algebra used to represent and manipulate structured collections of data. Matrix is defined by a rectangular array of elements arranged in rows and columns. A matrix of size $m \times n$

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}$$

Which a_{ij} denotes the element in the i -th row and j -th column.

There several fundamental operations are defined on matrices :

- 1) Addition and subtraction Matrix

Two matrices may be added or subtract if and only if they have the same dimensions:

$$(A \pm B)_{ij} = a_{ij} \pm b_{ij}$$

- 2) Scalar Multiplication Matrix

A matrix can be multiply a scalar value. Each element of the matrix is multiplied by a scalar value:

$$(kA)_{ij} = k \cdot a_{ij}$$

- 3) Matrix Multiplication

A matrix multiplication is a multiplication of matrix by another matrix. If $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times p}$, then their product $C \in \mathbb{R}^{m \times p}$ is defined as :

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$$

4) Transpose Matrix

Transpose Matrix is reflecting the matrix across its main diagonal, effectively converting its rows into columns and its columns into rows. The transpose of a matrix A is denoted as A^T .

B. Vectors

A vector is a mathematical object that has both magnitude and direction. Vectors are commonly represented as an ordered list of numerical components. A vector \mathbf{v} in an n -dimensional real space \mathbb{R}^n can be expressed as :

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

There are several fundamental operations defined on vector :

1) Addition and Subtraction Vectors

Two vectors of the same dimension can be added and subtract component-wise. Given vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$:

$$\mathbf{u} \pm \mathbf{v} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} \pm \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} u_1 \pm v_1 \\ u_2 \pm v_2 \\ \vdots \\ u_n \pm v_n \end{bmatrix}$$

2) Scalar Multiplication Vector

A vector may be scaled by multiplying each component by a scalar $c \in \mathbb{R}$:

$$c\mathbf{v} = \begin{bmatrix} cv_1 \\ cv_2 \\ \vdots \\ cv_n \end{bmatrix}$$

3) Magnitude (Norm) of Vector

Magnitude is represented a length or magnitude of a vector. The norm represents by :

$$\|\mathbf{v}\| = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}$$

4) Inner Product (Dot Product)

The dot product of two vectors \mathbf{u} and \mathbf{v} in \mathbb{R}^n is defined as:

$$\mathbf{u} \cdot \mathbf{v} = u_1 v_1 + u_2 v_2 + \cdots + u_n v_n = \sum_{i=1}^n u_i v_i$$

The dot product produces a single scalar value and serves as a fundamental measure of similarity between two vectors. Geometrically, the dot product relates to the angle θ between the vectors, and is given by:

$$\mathbf{u} \cdot \mathbf{v} = \|\mathbf{u}\| \|\mathbf{v}\| \cos \theta$$

This relationship shows that the dot product increases when two vectors point in similar directions and decreases when they point in opposite directions.

C. Matrix Transformations

A matrix transformation is a fundamental concept in linear algebra and refers to a function that maps vector from one vector space to another through matrix multiplication. Given a transformation matrix M and an input vector \mathbf{a} , the transformed vector \mathbf{a}' is defined as:

$$\mathbf{a}' = M\mathbf{a}$$

D. Self-Attention Mechanism

Self-Attention (also called intra-attention) is an attention mechanism that relates different positions within a single sequence in order to compute a new representation of that sequence. In contrast to traditional recurrent or convolutional architectures, which process tokens primarily based on their position in the sequence, self-attention allows each token to directly attend to all other tokens, regardless of their distance. This enables the model to capture long-range dependencies more efficiently.

Scaled Dot-Product Attention

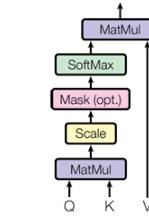


Figure 1. Diagram Process of Scaled Dot-Product Attention [Source : <https://arxiv.org/pdf/1706.03762.pdf>]

Multi-Head Attention

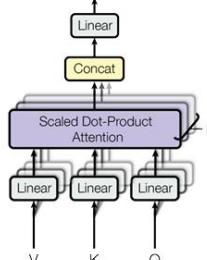


Figure 2. Multi-Head Attention consists of several attention layers running in parallel [Source : <https://arxiv.org/pdf/1706.03762.pdf>]

1) Query, Key, and Value Vector

In the general Transformer framework, an attention mechanism can be viewed as a function that maps three sets of vectors, such as queries, keys, and values, to a new set of output vectors. For each query vector, the mechanism compares it to all key vectors to measure how compatible or similar they are. The result of this comparison is a set of attention weights, which are then used to compute a weighted sum of the value vectors. Intuitively, the keys determine where the model should attend, the values determine what information is retrieved, and the queries represent what each position is currently looking for in the sequence.

Formally, let

- $Q \in \mathbb{R}^{n_q \times d_k}$ be the matrix of query vectors,
- $K \in \mathbb{R}^{n_k \times d_k}$ be the matrix of key vectors, and
- $V \in \mathbb{R}^{n_k \times d_v}$ be the matrix of value vectors.

2) Scaled Dot-Product Attention

Here, n_q denotes the number of query vectors, while n_k represents the number of key and value vectors, which is typically equal to the sequence length. The parameter d_k corresponds to the dimensionality of the query and key vectors, and d_v denotes the dimensionality of the value vectors. Within the Transformer architecture, the self-attention operation is defined using scaled dot-product attention as follows:

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

This formulation describes how contextual relevance between tokens is computed through matrix operations. The product QK^T captures similarity relationships between query and key vectors across the entire sequence, where each element reflects the degree of alignment between a query at one position and a key at another. From a geometric perspective, these dot products correspond to angular similarity in a high-dimensional vector space, with vectors pointing in similar directions producing larger values.

To maintain numerical stability as the vector dimensionality increases, the similarity scores are normalized by the factor $\sqrt{d_k}$. This scaling prevents excessively large values from dominating the softmax operation. The subsequent application of the softmax function transforms the scaled similarities into attention weights that sum to one, effectively determining how strongly each token attends to others when forming contextual representations.

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

where each row of A is a probability distribution over all keys for a given query: every entry is non-negative and the entries in each row sum to 1. Finally, the output of the attention mechanism is obtained by multiplying these weights with the value matrix:

$$\text{Output} = AV.$$

For each query position i , the resulting output vector is therefore a weighted linear combination of all value vectors, with weights determined by how similar the corresponding keys are to the query. Geometrically, each output lies in the convex hull of the value vectors, so the mechanism can be viewed as selecting a “context vector” by averaging values according to their relevance.

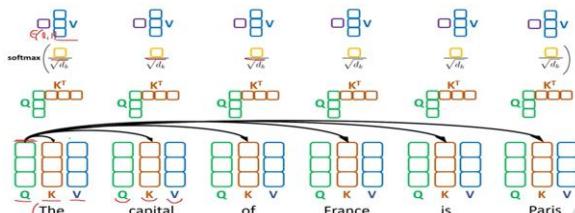


Figure 3. Self-attention over a sentence with Q , K , and V vectors for each token [Source : https://www.youtube.com/watch?v=u8pSGp_0Xk]

3) Multi-Head Attention

While a single scaled dot-product attention head is already capable of computing contextual representations, using only one attention head forces the model to focus on a single type of relationship at a time. In practice, different aspects of a sentence, such as syntactic structure, coreference, and long-range semantic links, may require the model to look at the sequence from different “perspectives” in vector space.

To address this, the Transformer architecture employs multi-head attention. Instead of computing one attention output, the model computes several attention heads in parallel, each with its own learned projection matrices. Formally, given the input matrices Q , K , and V the i -th head is defined as

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V),$$

which, $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$,
 $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, and
 $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$

are learned parameter matrices for the i -th head. Each head therefore operates in its own projected subspace, allowing it to capture a different type of dependency among tokens.

The outputs of all heads are then concatenated and linearly transformed to form the final multi-head attention output:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

which, h is the number of heads and
 $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$ is another learned projection matrix.

By using multiple heads with reduced dimensionality for each head, the model can jointly attend to information from different representation subspaces at different positions, without increasing the overall computational cost too much compared to a single-head attention with full dimensionality. In other words, multi-head attention enriches the expressiveness of self-attention while still remaining efficient and fully based on linear algebra operations.

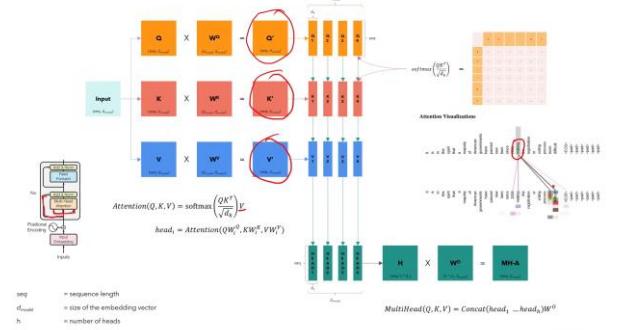


Figure 4. Multi-head attention mechanism illustrating parallel attention heads. [Source : <https://www.youtube.com/watch?v=bCz4OMemCcA>]

4) Positional Encoding and Word Order

By design, the self-attention mechanism is permutation-invariant: if the input tokens are shuffled, the pairwise interactions computed by attention do not inherently encode which token comes first or last. However, natural language is highly sensitive to word order. Two sentences that contain the same words, such as “He eats rice yesterday” and “Yesterday, he eats rice”, can convey different structures and emphasis. Therefore, the model must be given explicit information about the position of each token in the sequence.

To address this, the Transformer introduces positional encodings, which are added to the token embeddings before they are passed into the attention layers. For each position i in the sequence, a positional vector P_i is defined, and the final input representation is

$$X_i = E_i + P_i,$$

where E_i is the embedding of the i -th token and P_i encodes its position. In the original formulation, these positional encodings are constructed using sinusoidal functions of different frequencies so that the model can infer relative positions from linear combinations of these signals. Other variants, such as learned positional embeddings or rotary position encodings, follow the same principle: attach position-dependent information to each token.

Because positional encodings are added before the linear projections that generate Q , K , and V , the resulting query, key, and value vectors depend not only on the identity of the token but also on where it appears in the sequence. Consequently, the attention scores in QK^T are influenced by both lexical content and positional structure, allowing the model to differentiate between sentences that use identical words in different orders.

E. Role of Self-Attention in Next-Token Prediction

Building upon the self-attention mechanism described in the previous subsection, Large Language Models use stacked layers of masked self-attention to perform next-token prediction in an autoregressive manner

In Large Language Models, self-attention is the core mechanism used to build contextual representations that drive next-token prediction. These models are typically trained in an autoregressive manner: given a prefix (x_1, x_2, \dots, x_t) , the model learns to approximate the conditional probability

$$P(x_{t+1} | x_1, x_2, \dots, x_t).$$

To ensure that the prediction at position t only depends on tokens up to t , the Transformer uses masked self-attention in its decoder-style layers. A causal mask is applied so that, during the computation of attention scores, each position is allowed to attend only to itself and to earlier positions, but never to future tokens. Within each layer, masked multi-head self-attention aggregates information from all permitted previous positions into a

context vector for every token. Stacking several such layers yields a rich, context-sensitive representation h_t for the final position in the sequence.

This contextual vector is then passed through a linear output layer followed by a softmax function to obtain a probability distribution over the vocabulary:

$$P(x_{t+1} | x_1, \dots, x_t) = \text{softmax}(W_o h_t),$$

which W_o is a learned output projection matrix. The model can then select the next token by taking the argmax of this distribution or by sampling according to a chosen decoding strategy.

Through this process, self-attention provides the context aggregation step that determines which previous tokens are most relevant for predicting the next one. Because it can flexibly capture long-range dependencies, positional relationships, and nuanced interactions between tokens, self-attention has become the fundamental building block that enables Transformer-based Large Language Models to generate coherent and contextually appropriate text.

III. IMPLEMENTATION

A. Programming Language and Supporting Tools

Python is chosen due to its simplicity, readability, and extensive support for scientific computing and machine learning libraries. These characteristics make Python well-suited for implementing and analyzing mathematical models such as self-attention.

```
import torch
import torch.nn as nn
```

The main library used in this implementation is PyTorch, which provides efficient tensor operations and automatic differentiation. PyTorch is employed to perform all vector and matrix operations required in the self-attention mechanism. Although the implementation relies on library functions, the underlying mathematical operations strictly follow the theoretical formulation of self-attention discussed in previous chapters.

B. Text Preprocessing and Tokenization

Before self-attention can be applied, textual input must be converted into a numerical representation. The preprocessing stage includes lowercasing, removal of special characters, and tokenization.

```
def simple_tokenize(text: str) -> List[str]:
    text = text.lower().replace("<br />", " ")
    text = re.sub(r"^[^a-z0-9\s]", " ", text)
    tokens = text.split()
    return tokens
```

Each sentence is transformed into a sequence of tokens. These tokens are then mapped to integer indices using a vocabulary dictionary. Padding is applied to ensure a fixed sequence length.

```
def text_to_ids_and_mask(text: str, vocab: dict,
                        max_len: int) -> Tuple[List[int], List[int]]:
    tokens = simple_tokenize(text)
    ids = [vocab.get(tok, vocab["unk"]) for tok
           in tokens][:max_len]
    mask = [1] * len(ids)
    if len(ids) < max_len:
```

```

pad_len = max_len - len(ids)
ids += [vocab["<pad>"]] * pad_len
mask += [0] * pad_len
return ids, mask

```

The resulting output consists of:

- an index vector $x \in \mathbb{N}^n$,
- a binary attention mask $m \in \{0,1\}^n$.

C. Token and Positional Embedding Construction

Each token index is mapped to a dense vector using an embedding layer:

```

class IMDBSelfAttnClassifier(nn.Module):
    def __init__(self, vocab_size, d_model=256,
max_len=256, n_heads=4):
        super().__init__()
        self.token_emb = nn.Embedding(vocab_size, d_model, padding_idx=0)
        self.pos_emb = nn.Embedding(max_len, d_model)

```

The layer `token_emb` maps each token index to a dense vector in a d_{model} -dimensional space. Mathematically, this corresponds to an embedding function:

$$f: \mathbb{N} \rightarrow \mathbb{R}^{d_{model}}$$

Each token in the input sequence is therefore represented as a vector in a high-dimensional Euclidean space. To preserve the sequential order of tokens, positional embeddings are introduced through the `pos_emb` layer. For a sequence of length n , the final input representation is computed as:

$$X = E + P$$

where:

- $E \in \mathbb{R}^{n \times d_{model}}$ is the token embedding matrix,
- $P \in \mathbb{R}^{n \times d_{model}}$ is the positional embedding matrix.

This addition operation is a vector-space translation that injects positional information without changing the dimensionality of the embedding space. The combined embedding matrix X serves as the input to subsequent self-attention layers.

D. Scaled Dot-Product Self-Attention Mechanism

After the construction of token and positional embeddings, the model applies the self-attention mechanism to enable each token to dynamically incorporate contextual information from other tokens in the sequence. In this work, self-attention is implemented using the scaled dot-product attention formulation, which constitutes the core computational component of the model. Let

$$X \in \mathbb{R}^{n \times d_{model}}$$

denote the embedding matrix obtained from the previous stage, where n represents the sequence length and d_{model} denotes the embedding dimension. Each row of X corresponds to a vector representation of a token in the input sequence.

In self-attention, the matrix X is linearly projected into three distinct vector spaces known as Query (Q), Key (K), and Value (V). These projections are mathematically defined as:

$Q = XW_Q, K = XW_K, V = XW_V$, where $W_Q, W_K, W_V \in \mathbb{R}^{d_{model} \times d_k}$ are learnable weight matrices. In the implementation, these projections are handled internally by PyTorch's MultiheadAttention module, ensuring consistency with the standard theoretical formulation.

The relevance between tokens is computed by taking the dot product between query and key vectors, producing a similarity matrix:

$$S = QK^T.$$

Because the magnitude of dot products increases with the dimensionality of the vector space, the similarity scores are scaled by the factor $\sqrt{d_k}$ to improve numerical stability:

$$\tilde{S} = \frac{QK^T}{\sqrt{d_k}}.$$

The scaled similarity matrix is then normalized using the softmax function:

$$A = \text{softmax}(\tilde{S}),$$

where $A \in \mathbb{R}^{n \times n}$ is referred to as the attention matrix. Each entry A_{ij} represents the degree of attention that token i assigns to token j . This matrix encodes global token-to-token relationships within the sequence.

In the code, the entire attention computation is performed by the following call:

```

Z, A = self.attn(X, X, X,
key_padding_mask=key_padding_mask)

```

Since the same matrix X is used as the query, key, and value input, this operation corresponds to self-attention. The output of the attention mechanism is obtained through a weighted summation of value vectors:

$$Z = AV.$$

The resulting matrix $Z \in \mathbb{R}^{n \times d_{model}}$ represents a new contextualized embedding, where each token vector is influenced by all other tokens in the sequence. From a geometric perspective, this operation can be interpreted as projecting each token vector into a new position in the embedding space based on its similarity to other vectors.

To preserve information and stabilize training, a residual connection followed by layer normalization is applied:

$$X_{\text{out}} = \text{LayerNorm}(X + Z).$$

This operation ensures that the transformed representation remains close to the original embedding while maintaining consistent scale across dimensions.

After the attention operation, a position-wise feed-forward network is applied independently to each token vector. This network introduces nonlinearity and further refines the representation without altering the dimensionality of the embedding space. A second residual connection and normalization step are then applied, ensuring stable propagation of information through the network.

To prevent padded tokens from influencing the attention computation, an attention mask is used. This mask ensures that only valid tokens contribute to the attention matrix, preserving the correctness of the linear algebra operations with respect to the actual sequence length.

E. Stacked Self-Attention Blocks

To enhance the representational capacity of the model, the self-attention mechanism is applied multiple times by stacking two identical self-attention blocks within the classifier architecture. This design is implemented in the model as follows:

```
self.block1 =
ScaledDotProductSelfAttention(d_model=d_model,
n_heads=n_heads, p_drop=0.1)

self.block2 =
ScaledDotProductSelfAttention(d_model=d_model,
n_heads=n_heads, p_drop=0.1)
```

Each block performs scaled dot-product self-attention followed by residual connections, normalization, and a position-wise feed-forward network, as described in the previous section. The output of the first self-attention block serves as the input to the second block.

From a linear algebra perspective, stacking self-attention blocks can be interpreted as applying successive transformations to the embedding matrix. Let $X^{(0)}$ denote the initial embedding matrix produced after the token and positional embedding stage. The transformations performed by the stacked self-attention blocks can be expressed as:

$$\begin{aligned} X^{(1)} &= \text{SelfAttention}(X^{(0)}), \\ X^{(2)} &= \text{SelfAttention}(X^{(1)}). \end{aligned}$$

Each application of self-attention refines the vector representation of tokens by incorporating increasingly rich contextual information. While the first block primarily captures direct relationships between tokens, the second block operates on already contextualized vectors, allowing the model to represent higher-order dependencies within the sequence.

Geometrically, this process can be viewed as iteratively projecting token vectors into new subspaces within the same high-dimensional embedding space. Each projection adjusts the position of vectors based on their relationships with other vectors, resulting in representations that better encode global structure and semantic context.

The use of stacked self-attention blocks does not alter the dimensionality of the embedding space. Instead, it progressively refines the orientation and relative distances between vectors. This characteristic preserves the linear algebraic structure of the representation while increasing its expressive power.

F. Sentence Representation and Classification

After passing through the stacked self-attention blocks, each token in the input sequence is represented by a contextualized vector that incorporates information from the entire sequence. To perform classification at the sentence level, these token-level representations must be aggregated into a single fixed-size vector. Let

$$X \in \mathbb{R}^{n \times d_{model}}$$

denote the output matrix of the final self-attention block, where each row corresponds to a contextualized token

representation. Because input sequences may contain padding, an attention mask is used to ensure that only valid tokens contribute to the aggregation process

In the implementation, sentence-level representation is obtained using masked average pooling:

```
mask = attention_mask.unsqueeze(-1)
summed = (X * mask).sum(dim=1)
lengths = mask.sum(dim=1).clamp(min=1e-6)
pooled = summed / lengths
```

Mathematically, this operation computes the average of the token vectors while excluding padded positions:

$$\bar{z} = \frac{1}{n} \sum_{i=1}^n z_i,$$

where z_i denotes the contextualized representation of the i -th token and n is the number of valid tokens in the sequence.

From a linear algebra perspective, this pooling operation can be interpreted as a projection from a matrix space to a vector space, mapping the sequence representation from $\mathbb{R}^{n \times d_{model}}$ to a single vector in $\mathbb{R}^{d_{model}}$. This vector serves as a compact summary of the entire sentence.

The resulting pooled vector is then passed to a feed-forward classification network:

```
self.classifier = nn.Sequential(
    nn.Dropout(0.2),
    nn.Linear(d_model, d_model // 2),
    nn.ReLU(),
    nn.Dropout(0.2),
    nn.Linear(d_model // 2, 1),
)
```

This classifier consists of a sequence of linear transformations and nonlinear activations. The first linear layer reduces the dimensionality of the sentence vector, followed by a nonlinear activation function to introduce expressive power. The final linear layer maps the representation to a single scalar value.

The output of the classifier is interpreted as a logit, which is subsequently transformed using the sigmoid function:

$$p = \sigma(\text{logit}),$$

where $p \in [0,1]$ represents the predicted probability of the positive class. In this study, the classifier is applied to sentiment analysis on movie reviews, where the output probability indicates whether a given sentence expresses positive sentiment.

G. Attention Matrix Extraction and Interpretation

One important advantage of the self-attention mechanism is its interpretability. Unlike traditional text representation methods that rely solely on global statistics, self-attention explicitly produces an attention matrix that describes how tokens interact with one another. In this study, the attention matrix is extracted and analyzed to better understand the internal behavior of the model.

During inference, the self-attention block returns not only the contextualized token representations but also the attention weights:

```
probs, logits, A = model(input_ids,
attention_mask)
```

The matrix A represents the attention weights produced by the final self-attention block. For analysis purposes, a single attention head is selected and extracted:

```
A_matrix = A[0, 0]
```

```
A_cut = A_matrix[:len(tokens), :len(tokens)]
```

Mathematically, the attention matrix is defined as:

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

where each entry A_{ij} indicates the contribution of the j -th token to the contextual representation of the i -th token.

From a linear algebra perspective, the attention matrix can be interpreted as a context-dependent weighting matrix. Each row of A forms a probability distribution over the tokens in the sequence, and the multiplication AV produces a weighted combination of value vectors. This process allows each token to dynamically adjust its representation based on its relationship with other tokens.

Geometrically, the attention matrix captures angular and directional relationships between vectors in the embedding space. Tokens whose query and key vectors form smaller angles (i.e., higher similarity) receive larger attention weights. As a result, tokens that are semantically related or contextually important tend to influence each other more strongly, even if they are distant in the sequence.

This behavior highlights a fundamental difference between self-attention and classical matrix-factorization approaches such as Latent Semantic Analysis (LSA). While LSA relies on static co-occurrence statistics and global frequency patterns, self-attention computes relevance dynamically for each input sequence. Consequently, the attention matrix adapts to word order and contextual meaning rather than relying on fixed similarity scores.

By visualizing and examining the attention matrix, it becomes possible to observe which tokens contribute most significantly to the representation of other tokens. This provides concrete evidence that the model performs context-aware processing rather than simple frequency-based matching.

IV. RESULTS AND DISCUSSION

A. Experimental Setup

The proposed self-attention-based sentiment classification model is evaluated using textual data derived from the IMDB Dataset of 50K Movie Reviews – Kaggle. This dataset consists of 50,000 English-language movie reviews collected from the Internet Movie Database (IMDB), evenly divided into positive and negative sentiment classes. The dataset is widely used as a benchmark for sentiment analysis tasks and provides a diverse range of review lengths and writing styles.

In this study, the dataset is utilized to train and evaluate the model's ability to predict sentiment polarity based on natural language input. The use of the IMDB Dataset of 50K Movie Reviews ensures that the experimental results are grounded in a realistic and standardized sentiment analysis scenario.

All textual inputs are preprocessed using the same pipeline described in Section III, including tokenization, numerical encoding, and padding to a fixed maximum sequence length of 256 tokens. During evaluation, the trained model produces the following outputs:

1. A positive sentiment probability,
2. A binary sentiment classification result, and

3. A self-attention weight matrix extracted from the final attention layer.

These outputs enable both quantitative evaluation of prediction accuracy and qualitative analysis of how the self-attention mechanism distributes importance across tokens in a given input sequence.

B. Experimental Result

The experimental results obtained from testing the trained self-attention model on several input sentences derived from the IMDB sentiment analysis domain. The objective of these experiments is to evaluate both the sentiment prediction performance and the behavior of the self-attention mechanism when processing different types of textual input.

The model was trained on the IMDB Dataset of 50K Movie Reviews and achieved a test accuracy of 86.18%, indicating that the model is capable of generalizing well to unseen data. After training, the model was evaluated using manually constructed test sentences with varying sentiment characteristics.

Several representative examples are shown below:

1. Long Positive Review

Input:

“I really enjoyed this movie because the story was engaging, the characters felt realistic, and the overall experience was emotionally satisfying from beginning to end.”

Result:

The model produces a very high positive sentiment probability (>90%).

Observation:

The attention weights are strongly concentrated on sentiment-bearing tokens such as enjoyed, engaging, realistic, and satisfying. Tokens related to narrative quality and emotional experience receive higher attention values compared to neutral structural words.

2. Long Negative Review

Input:

“I wanted to like this movie, but the plot was boring, the pacing was slow, and the characters failed to keep my attention throughout the entire film.”

Result:

The predicted positive sentiment probability is very low (<10%).

Observation:

Although the sentence begins with a mildly positive phrase (wanted to like), the attention mechanism shifts focus toward negative sentiment tokens such as boring, slow, and failed. The contrastive structure introduced by the word but plays a significant role, with later clauses receiving stronger attention weights.

3. Mixed / Ambiguous Sentiment

Input:

“The movie started with an interesting premise and strong visuals, but as the story progressed it

became predictable and less engaging, even though some scenes were still enjoyable.”

Result:

The model outputs a moderate positive probability (approximately 40–60%).

Observation:

The attention matrix reveals that positive tokens (interesting, strong visuals, enjoyable) and negative tokens (predictable, less engaging) both receive notable attention weights. However, no single sentiment dominates the sequence.

4. Subjective Positive Opinion with Minor Negation

Input:

“Although the film is not perfect and has a few pacing issues, I found it to be an entertaining and heartfelt experience that I would gladly watch again.”

Result:

The predicted sentiment is clearly positive.

Observation:

While the phrase not perfect introduces a mild negative signal, the attention mechanism assigns higher weights to the concluding clause (entertaining, heartfelt, watch again). This suggests that the model learns to emphasize final evaluative judgments, which are often decisive in IMDB-style reviews.

5. Subtle Negative Review Without Strong Negative Keywords

Input:

“The movie was not terrible, but it never managed to become truly interesting, and I struggled to stay engaged until the end.”

Result:

The model predicts a low positive probability, classifying the review as negative.

Observation:

Despite the absence of explicit harsh language, the attention mechanism correctly focuses on phrases such as never managed, struggled, and stay engaged. The model does not misinterpret the phrase not terrible as a positive indicator, showing robustness against superficial negation patterns.

C. Discussion

The experimental results demonstrate that the self-attention mechanism effectively identifies and prioritizes sentiment-relevant tokens within long and complex input sequences. Across all examples, attention weights dynamically shift toward words and phrases that carry strong semantic and emotional meaning, while less informative tokens receive lower importance.

Notably, the model performs well on:

- Long-form reviews with multiple clauses,
- Contrastive sentence structures,
- Subtle or implicit negative sentiment.

These behaviors are consistent with the theoretical properties of self-attention discussed in previous chapters, where each token representation is updated through

weighted combinations of all other tokens in the sequence. The resulting attention matrices provide an interpretable representation of how sentiment information is distributed and aggregated in high-dimensional vector spaces.

V. CONCLUSION

The self-attention mechanism emerges as the core algorithm underlying next-token prediction in Large Language Models when viewed from a linear algebraic and geometric perspective. Through vector space representations, matrix transformations, and similarity-based weighting, self-attention can be understood as a mathematically structured and interpretable mechanism.

Through a detailed review of linear algebra fundamentals this work establishes a clear theoretical foundation for understanding how self-attention operates. The formulation of Query, Key, and Value vectors demonstrates how token representations are linearly projected into multiple subspaces, while scaled dot-product attention illustrates how geometric similarity in high-dimensional space determines contextual relevance. The use of softmax normalization further ensures that attention weights form valid probability distributions, enabling meaningful weighted combinations of value vectors.

A simplified implementation of self-attention was presented to demonstrate these concepts in practice. Although the model does not operate at the scale of real-world Large Language Models, it faithfully captures the essential mathematical structure of self-attention. Experimental evaluation using the IMDB Dataset of 50K Movie Reviews shows that the model is capable of handling long and complex input sentences, including contrastive structures, mixed sentiment, and subtle negations. The attention matrices extracted during inference provide explicit evidence that the model dynamically prioritizes sentiment-relevant tokens rather than relying on word frequency alone.

From a geometric viewpoint, self-attention can be interpreted as an iterative projection and refinement process in a high-dimensional embedding space, where token vectors are repositioned based on their angular and directional relationships to other tokens. This interpretation highlights a fundamental difference between self-attention and classical matrix-factorization methods such as Latent Semantic Analysis, which rely on static global statistics and do not adapt to input-specific context.

In conclusion, self-attention serves as a compelling example of how linear algebra and geometry form the mathematical backbone of modern machine learning systems. By connecting abstract mathematical concepts to a concrete implementation and experimental analysis, this work demonstrates the relevance of algebraic and geometric reasoning in understanding and designing contemporary language models. The insights presented in this paper reinforce the importance of linear algebra as a foundational discipline in the study of artificial intelligence and large-scale language modeling.

VI. APPENDIX

GitHub:[Alpaomega1136/Self-Attention-in-a-Linear-Algebra-and-Geometric-Perspective](https://github.com/Alpaomega1136/Self-Attention-in-a-Linear-Algebra-and-Geometric-Perspective)

VII. ACKNOWLEDGMENT

The author is deeply thankful to God Almighty for providing strength, determination, and opportunity to bring this paper to completion. The author also expresses deep appreciation to Ir. Rila Mandala, M.Eng., Ph.D., the lecturer of the IF2123 Linear Algebra and Geometry course, for his dedicated guidance and support, which have been a source of inspiration throughout his teaching journey with the students.

REFERENCES

- [1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention Is All You Need,” *arXiv preprint arXiv:1706.03762*, 2017. [Online]. Available: <https://arxiv.org/abs/1706.03762> [Accessed: 8-Dec-2025].
- [2] R. Munir, “Review Matriks,” *Aljabar dan Geometri*, STEI-Institut Teknologi Bandung, 2025. [Online]. Available: <https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2025-2026/Algeo-01-Review-Matriks-2025.pdf> [Accessed: 8-Dec-2025].
- [3] R. Munir, “Vektor di Ruang Euclidean – Bagian 1,” *Aljabar dan Geometri*, STEI-Institut Teknologi Bandung, 2025. [Online]. Available: <https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2025-2026/Algeo-11-Vektor-di-Ruang-Euclidean-Bag1-2025.pdf> [Accessed: 8-Dec-2025].
- [4] R. Munir, “Ruang Vektor Umum – Bagian 4,” *Aljabar dan Geometri*, STEI-Institut Teknologi Bandung, 2025. [Online]. Available: <https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2025-2026/Algeo-18-Ruang-vektor-umum-Bagian4-2025.pdf> [Accessed: 8-Dec-2025].
- [5] DataMListic, “Transformer Self-Attention Mechanism Visualized,” *YouTube*, 2022. [Online]. Available: https://www.youtube.com/watch?v=u8pSGp_0Xk [Accessed: 8-Dec-2025].
- [6] GeeksforGeeks, “Self-Attention in NLP,” last updated Aug. 23, 2025. [Online]. Available: <https://www.geeksforgeeks.org/nlp/self-attention-in-nlp/> [Accessed: 8-Dec-2025].
- [7] “Self Attention in Transformers | Transformers in Deep Learning,” *YouTube*, n.d. [Online]. Available: https://www.youtube.com/watch?v=SO2_3YS6e-k [Accessed: 8-Dec-2025].
- [8] Lakshmi Narayanan, “IMDB Dataset of 50K Movie Reviews,” *Kaggle*, n.d. [Online]. Available: <https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews> [Accessed: 10-Dec-2025].

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 24 Desember 2024



Raymond Jonathan Dwi Putra Julianto
13524059