Understanding the Mathematical Foundations of the Transformer Model in "Attention Is All You Need"

Version 0.1

**Sept 27, 2022**

Authors:

Mohammed Khalil mk3441@att.com

Contents

[Document Control 3](#_Toc172710729)

[1 Introduction 4](#_Toc172710730)

[1.1 High-Level Architectural Description 4](#_Toc172710731)

[**Calculating Q, K, V** 11](#_Toc172710732)

[1.1.1 Calculating the Gradient of the Loss function with Respect 13](#_Toc172710733)

[1.1.2 Gradient Optimization for 13](#_Toc172710734)

[1 Input and learned matrices. 24](#_Toc172710744)

[**Mean Squared Error (MSE) Loss** 26](#_Toc172710745)

[2 Acronyms 26](#_Toc172710746)

Document Control

Below are the publication date, author’s initials, and reason for publication for the current and any previous versions of this document.

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Publication Date** | **Author** | **Reason for Publication** |
| 0.1 | <27 SEPTEMPER 2022> | MK,PV,GB,HM | NWDAF architecture description, phases, requirements |
| 0.2 |  |  |  |
| 1.0 |  |  | Draft Ready |
| 2.0 |  |  | Internal Inspection Ready <This is the version that is used for the formal inspection> |
| 3.0 |  |  | Internal Inspection Complete <Only bump to this version once all issues have been resolved and agreed. Intermediate versions are numbered 2.1, 2.2 as needed. V3.0 should not contain any change bars> |
| 4.0 |  |  | Approved <V4.0 is only used if the FRS is formally approved by a customer> |
|  |  |  |  |

# Introduction

The "Attention Is All You Need" paper by Vaswani et al. introduced the Transformer model, a novel architecture designed for sequence-to-sequence tasks such as translation. The Transformer model revolutionized natural language processing by eliminating the need for recurrent or convolutional structures, instead relying entirely on attention mechanisms. This shift enables the model to capture global dependencies in input sequences more efficiently.

This document provides an in-depth explanation of the mathematical foundations underpinning the Transformer, focusing on key concepts like the attention mechanism, multi-head attention, positional encoding, and feed-forward networks.

## Mathematical Foundation

### Hadamard product

to vector

# 

# **=**

### Similarity Concept

#### Dot Product



### Jacobian Matrix

#### Single Value Scalar Function

f(x) =

**= 2****x**

The first derivative of the function f of the independent variable x is the slope of the function f at certain value of x. Or how fast the function f changing at this particular x value. So if we change the value at x by  x, so x will cause a change of the output f(x). The slope of f(x) at x when it changes by x measures the sensitivity of changing the function f(x) when we change x by x.

**Slope of f(x) at x**

**x**

**x+ x** 1

**x**1

if we extend the concept of derivative to the concept of gradient, gradient means the derivative of scalar function to multiple independent variables (,, ….) feeding to the function:

### **Multiple Value Scaler Function**

the gradient is given as =  **=**

is the partial derivative of the function f for each one of the independent variables of the vector . is the sensitivity or the rate function f changes in

**Example** f(,) =

**=** **2**, measure the sensitivity or the rate the function f changes with respect to

**=** **2** , measure the sensitivity or the rate the function f changes with respect to

**=**

### **Jacobian Matrix**

Instead of dealing with a single scaler function, we use a vector valued function. So instead of having scaler output we have a vector output. The figure below shows the output of 3 scaler functions on the top of each other:

***(, ) =***

***(, ) =***

***(, ) =***

X1

X2

***(,)***

***(,)***

*(*

***(,)***

The Jacobian matrix is expressed as follows:

***(***

Since the gradient of the function f with respect to is expressed as:

=  **=**

So, the Jacobian matrix J is stack of the transpose of each in the top of each other. Jacobian matrix measures the sensitivity or the rate of change of each function for each variable in vector **.**

**J = =**

is defined as the sensitivity of function  output to the change of .

## 

## **High-Level Architectural Description**

Input sequence X of shape (n,

* Number of tokens (n): 2
  + "Hello"
  + "world"

**input Embeddings**: Assume each token is embedded into a vector of dimension = m

**weight matrices** , , and

**Query, Key, Value Matrices**: Using learned weight matrices , , and

= = .

### **Positional Encoding**

Transformers process input sequences in parallel, treating each token independently. Without positional information, the model would have no sense of word order, which is essential for understanding language. Positional encoding provides a way to include this order information.

* **Positional encoding** is added to the **input embeddings**.
* These **position-aware embeddings** are then used to generate the **Query (Q), Key (K), and Value (V)** vectors through linear transformations.
* This approach ensures that the positional information is integrated into the Q, K, and V vectors, allowing the self-attention mechanism to be aware of the positions of the tokens in the sequence.

Q

### Sinusoidal Positional Encoding:

* Uses sine and cosine functions to encode positions.

PE(pos,2i) =

PE(pos,2i+1) =

* pos is the position in the sequence, i is the dimension, and d is the dimensionality of the encoding.
* This method ensures that each position has a unique encoding and that similar positions have similar encodings.
* Enables the model to generalize to longer sequences than it was trained on, since the encoding is periodic.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| pos | PE(pos,0) | PE(pos,1) | PE(pos,2) | PE(pos,3) |
| 0 |  |  |  |  |
| 1 |  |  |  |  |
| 2 |  |  |  |  |
| 3 |  |  |  |  |
| 4 |  |  |  |  |

Example:

* Sequence: ["The", "cat", "sat", "on", "the", "mat"]
* Sequence length (seq\_len): 6
* Embedding Dimensionality (d) : 4

Each word in the sequence is represented as a 4-dimensional vector:

​=[0.1,0.2,0.3,0.4]

And the positional encoding for position 0 is

=[=**0.0**, =**1.0**, =**0.0**,=**1.0**]

Then the input to the transformer for "The" at position 0 would be: +=[0.1,1.2,0.3,1.4]

)

* **Rows**: Each row i in the attention weights matrix corresponds to a token xi in the input sequence. The values in this row represent how much attention token xi pays to every other token xj (including itself) in the sequence. Or what is the similarity of each token xi to the other token xj including itself.
  + For example, the first row [a11,a12,…,a1n] represents the attention distribution of the first token over all tokens. Or the similarity of the first token over all tokens.
* **Columns**: Each column j in the attention weights matrix corresponds to the amount of attention all tokens pay to the token xj. Or what is the similarity of all tokens to xj. Or what is the similarity of the first token a11 to itself, to the second token a12, to the n token a1n.
  + For example, the first column [a11,a21,…,an1] represents how much attention all tokens in the sequence pay to the first token. Or what is the similarity of all tokens to the first token. Or what is the similarity of first token a11 to itself, the similarity of the second token a21 to the first one, the similarity of the third token a31 to the first one, etc.

Example

Input matrix X

Weight matrix for queries and keys

Calculate Q , K

Calculated scaled scores.

Applying SoftMax in each row

SoftMax ( = = =

SoftMax ( = =  =

* **First Row** ([0.623,0.377]: Represents how much the 1st token pays attention to the 1st token (0.623) and the 2nd token (0.377). or the similarity of the 1st token to itself, and similarity of the 1st token to the 2nd .
* **Second Row** ([0.623,0.377]): Represents how much the 2nd token pays attention to the 1st token (0.623) and the 2nd token (0.377).
* **First Column** ([0.633,0.377]): Represents how much attention the 1st token (0.623) and the 2nd token (0.377) pay to the 1st token. Or the similarity of the 1st token to the first token and 2nd token.
* **Second Column** ([0.377,0.623]): Represents how much attention the 1st token (0.377) and the 2nd token (0.623) pay to the 2nd token. Or the similarity of the 2nd token to the 1st token (0.377) and the 2nd token (0.623).

***Z = attention\_weights . V***

* **z11​**: This element is the weighted sum of the 1st feature across all tokens, using the attention weights from the 1st token.
* **z12** This element is the weighted sum of the 2nd feature across all tokens, using the attention weights from the 1st token.

Example

​=[0 1]

​=[1 0]

​=[1 1]

X=  =

Assume ​ = 2

PE(pos,0) = = sin (pos)

PE(pos,1) = = cos (pos)

pos = 0

PE (0,0) = sin (0) = 0

PE (0,1) = cos (0) = 1

Pos = 1

PE (1,0) = sin (1) = 0.840

PE(1,1) = cos (1) = 0.540

Pos = 2

PE (2,0) = sin (2) = 0.909

PE(1,1) = cos (2) = -0.41

1. **Assume Transformation Matrices:**

**Weight matrix for Q ()**

**Weight matrix for K ()**

**Weight matrix for V ()**

1. **Calculating Q, K, V**

**Query matrix Q:**

**Q = (X + P) .**

**Q =**

**Key matrix (K)**:

**K = (X + P) .**

**K =**

**Key matrix (V)**:

***V= (X + P) .***

**V =**

1. **Attention Scores**

**=**

**=**

**=**

1. **softmax of Attention Scores**

**=**

1. **Output matrix Z**

**Z=Attention Weights . V**

**Z=**

**T=**

### **Gradient Optimization for**

1. **Forward pass**

* **Compute Queries, Keys, and Values,** Given the input sequence X:

* **Calculated scaled scores:**

* Apply the softmax function to get attention weights:
* Compute the output of the attention mechanism:

***Z=A⋅V***

* **Feed-Forward Network:** The output Z from the attention mechanism is then passed through a feed-forward neural network

1. **Compute Loss:** The output from the Transformer layer is compared to the target values using a loss function (e.g., cross-entropy loss for classification tasks).
2. **Backward Pass (backpropagation:** Calculate Gradients
3. Gradient of Loss with Respect to Z:
4. Gradient of Loss with Respect to V :
5. Gradient of Loss with Respect to Attention Weights:

1. Gradient of Loss with Respect to Scores:

; where ***f = softmax***

1. Gradient of Loss with Respect to Q and K:

1. Gradient of Gradient Loss with Respect to  ​ :
2. **Update Parameters:** Using the gradients calculated in the backward pass, the parameters​, ​, and​ are updated using an optimization algorithm like stochastic gradient descent (SGD)

**The derivations of**

# **Q =**

# **K** =

# **V** =

Then

Next step is to evaluate the derivations of

#### **Gradient L with Respect to Q (** **)**

Given the loss L, the goal is to compute the gradient ​. We'll use the chain rule to find this gradient, considering the dependencies through the attention mechanism.

​ = **=**

1. **Gradient of the loss with respect to the scores:**

Let scores be the matrix of attention scores. We need the gradient of the loss with respect to these scores:

1. **Scores depend on Q and K:**

The scores (S) are computed as:

So, we need to differentiate the scores with respect to Q:

**Score Matrix Calculation:** The score matrix S is calculated as:

**First, compute**

**S = =**

**Partial Derivative Calculation**

Let's focus on calculating the partial derivatives of each element in S with respect to each element in Q.

Partial Derivatives for ​ =

=  =  = 0

Partial Derivatives for ​ =

= = = 0

Partial Derivatives for ​ =

= 0  = 0 =

Partial Derivatives for ​ =

= 0 = 0 =

To map this 4x4 gradient matrix to a form related to we need to recognize that:

1. The gradient of ​ and ​ with respect to ​ and corresponds to the first row of
2. The gradient of ​ and ​ with respect to ​ and corresponds to the second row of .

Thus, the 4x4 gradient matrix essentially breaks down into a form that directly corresponds to the elements of :

This shows how the gradient matrix maps directly to , confirming that the gradient of the score matrix S with respect to the query matrix Q is indeed .

**Summary:** By carefully examining the partial derivatives and constructing the gradient matrix, we see that the 4x4 gradient matrix can be simplified to match the structure of . Each element in the gradient matrix corresponds directly to the elements of , reflecting the influence of Q on the score matrix S. Thus, = .

1. **The chain rule:**

Using the chain rule, the gradient of the loss with respect to Q is:

Substitute the partial derivative of scores with respect to Q:

To simplify, assume ​​=1 for clarity (or include it in the scaling factor):

##### **Numerical Example**

Assume the following matrices for Q, K, and scores:

Score = S =

Assume a hypothetical gradient of the loss with respect to the scores:

**Gradient Calculation**

1. **Compute:**
2. **Apply the chain rule:**
3. **Matrix multiplication:**

=

=

#### **Gradient L with Respect to K ( )**

Given the loss L, the goal is to compute the gradient ​. We'll use the chain rule to find this gradient, considering the dependencies through the attention mechanism.

1. **Gradient of the loss with respect to the scores:**

Let scores be the matrix of attention scores. We need the gradient of the loss with respect to these scores:

1. **Scores depend on Q and K:**

The scores (S) are computed as:

So, we need to differentiate the scores with respect to K:

**Score Matrix Calculation:** The score matrix S is calculated as:

**First, compute**

**S =** **=**

Let's focus on calculating the partial derivatives of each element in S with respect to each element in Q.

Partial Derivatives for ​ =

= = = 0

Partial Derivatives for ​ =

= 0 =0 =

Partial Derivatives for ​ =

= = = 0

Partial Derivatives for ​ =

= 0 =0 =

Gradient Matrix

Now, we can organize these partial derivatives into the gradient matrix:

To map this 4x4 gradient matrix to a form related to we need to recognize that:

1. The gradient of ​ and ​ with respect to ​ and corresponds to the first row of
2. The gradient of ​ and ​ with respect to ​ and corresponds to the second row of .

Thus, the 4x4 gradient matrix essentially breaks down into a form that directly corresponds to the elements of :

This shows how the gradient matrix maps directly to, confirming that the gradient of the score matrix S with respect to the query matrix K is indeed

**Summary:** By carefully examining the partial derivatives and constructing the gradient matrix, we see that the 4x4 gradient matrix can be simplified to match the structure of . Each element in the gradient matrix corresponds directly to the elements of , reflecting the influence of Q on the score matrix S. Thus, =

1. **The chain rule:**

Using the chain rule, the gradient of the loss with respect to Q is:

**, =**

Substitute the partial derivative of scores with respect to Q:

To simplify, assume ​​=1 for clarity (or include it in the scaling factor):

##### **Numerical Example**

Assume the following matrices for Q, K, and scores:

Score = S =

Assume a hypothetical gradient of the loss with respect to the scores:

**Gradient Calculation**

1. **Compute :**
2. **Apply the chain rule:**

=

=

#### **Gradient L with Respect to S** **( )**

**In general:**

) =

##### **Gradient A with Respect to S ( )**

# **Score Matrix Calculation:** The score matrix S is calculated as:

# 

# First, compute

# **S = =**

# A = softmax(S) **=** softmax

A=

# **J=**

**If i = n an∂ j = m**

**If i** **≠ n 0r j ≠ m**

# **J=**

# 

#### **Gradient L with Respect to V ( )**

Given the loss L, the goal is to compute the gradient ​. We'll use the chain rule to find this gradient, considering the dependencies through the attention mechanism.

**= ;** where

##### Gradient L with Respect to Z ( )

I**nput Sequence Length n**:

* n is the length of the input sequence, i.e., the number of tokens (words, subworld’s, characters, etc.) in the sequence being processed by the Transformer model.
* For example, if the input sentence is "The quick brown fox jumps over the lazy dog," n would be 9 if each word is treated as a token.

The goal is to find Loss​ for each element in Z.

1. Loss Function:
2. Isolate the ith Term :

**Loss = (**

1. We are interested in terms involving :

**Loss =**

1. To find the gradient of the loss with respect to we use the chain rule of differentiation. Since the other terms do not depend on to they disappear in differentiation:

##### Summary of the Procedure for

This derivation is a scalar L by vector derivation:

=

The Calculation of the Gradient of the Loss function with Respect is summarized as follows:

* ( + ()
* =
* () = 0

* = =

##### **Numerical Example**

Given the output matrix Z:

Z =

And the target matrix T:

T =

We want to calculate

# **Element-wise differences**

Z-T = =

# **Compute** **​ (where n=3)**

=  0.6667

# **Detailed calculation for**

# **The result in the gradient matrix**

This matrix represents the gradient of the loss function with respect to the output Z, indicating how the loss changes with respect to each element of Z. This information is used to update the model parameters during training to minimize the loss.

##### Gradient L with Respect to Z ( )

Given Z=AV,

where:

* A is an m×n matrix,
* V is an n×p matrix,
* Z is an m×p matrix.

we want to find the derivative of

Z= A.V=

### **Matrix Notation**

In matrix notation, each element of is a dot product of the rows of and the columns of

### **Partial Derivative Calculation**

() = (

To find the derivative of ​ with respect to ​, we observe that involves ​, but only when k=p Therefore:

### **Partial Derivative Calculation for Matrices 2x2**

A Matrix

V Matrix

Calculate Z

Z =

**=**

Partial Derivatives for ​ =

= = 0 =

Partial Derivatives for ​ =

= 0 = =0

Partial Derivatives for ​ =

=  = 0 =

Partial Derivatives for ​ =

= 0 = = 0

Gradient Matrix

Now, we can organize these partial derivatives into the gradient matrix:

**Since**

**=**

= =

**Then**

##### **Numerical Example for**

A Matrix

T Matrix

V Matrix

Partial Derivatives for ​ = = 0.1x1 + 2x03 =0.1+0.6 = 0.7

= = 0 =

Partial Derivatives for ​ = = 0.1x2 +0.2 x4 = 0.2+0.8 =1

= 0.0 ==0.1 =0.0 =0.2

Partial Derivatives for ​ = =0.3x1+0.4 x3=0.3+1.2=1.5

= =0.3 = 0.0 =

Partial Derivatives for ​ = =0.3x2+0.4x4=0.6+1.6 =2.2

= 0.0 = =0.3 = 0.0 =0.4

# Input and learned matrices.

 Input sequence X of shape (n,

* Number of tokens (n): 2
  + "Hello"
  + "world"

 Learned weight matrices , , of shape (dmodel,dk)(d\_{\text{model}}, d\_k)(dmodel​,dk​)

#### Derivation of

**= ;** where

I**nput Sequence Length n**:

* n is the length of the input sequence, i.e., the number of tokens (words, subworld’s, characters, etc.) in the sequence being processed by the Transformer model.
* For example, if the input sentence is "The quick brown fox jumps over the lazy dog," n would be 9 if each word is treated as a token.

The goal is to find Loss​ for eac h element  in Z.

1. Loss Function:

1. Isolate the ith Term :

**Loss = (**

1. We are interested in terms involving :

**loss =**

1. To find the gradient of the loss with respect to we use the chain rule of differentiation. Since the other terms do not depend on to they disappear in differentiation:

##### Summary of the Procedure for

The Calculation of the Gradient of the Loss function with Respect is summarized as follows:

* ( + ( )
* ( ) = 0
* +

Z = Attention Weights x V

**T=**

**Mean Squared Error (MSE) Loss**

The MSE loss is given by:

# Acronyms

CSC Communication Service Customer

NWDAF 5G Network Data Analytical Function

AF Application Function

NF Network Function

CSMF Communication Service Management Function

CSP Communication Service Provider

DN Data Network

MNO Mobile Network Operator

NOP Network Operator

NSaaS Network Slice as a Service

NSaasC Network Slice as a Service Customer

NSaaSP Network Slice as a Service Provider

NSMF Network Slice Management Function

NSC Network Slice Customer

NSSMF Network Slice Subnet Management Function

NSP Network Slice Provider

SLA Service Level Agreement

SLS Service Level Specification

TN Transport Network