# Transformer Architecture

compiled by D.Gueorguiev, 4/19/2025 from various sources

Input Embedding

Inputs

Multi-head attention

Feed forward

Add and Normalize

Output Embedding

Masked Multi-head attention

Add and Normalize

Outputs

(shifted right)

Add and Normalize

Multi-head attention

Add and Normalize

Feed forward

Add and Normalize

Linear

Softmax

Output Probabilities

Positional Encoding

Positional Encoding

Nx

Nx

Figure: Transformer Architecture (from the original paper *“Attention is All you need”*)

The original Transformer model is a stack of 6 layers. The output of layer is the input of layer until the final prediction is reached. There is a 6 layer encoder stack on the left and 6 layer encoder stack on the right. On the left the inputs enter the encoder side through an Attention sub-layer and Feed-forward network sub-layer. On the right the target outputs go into the Decoder side through two Attention sub-layers and a Feed-forward network sub-layer. Notice that there is no vestige of Recurrent networks in this architecture – Recurrence have been abandoned. Attention replaces Recurrence which requires an increasing number of operations as the distance between two words increases. The Attention mechanism is a “word-to-word” operation. The Attention mechanism will find how each word is related to all other words in a sequence, including the word being analyzed itself.

Let us examine one popular example which is the following sentence:

The cat sat on the mat.

Attention will run dot products between word vectors and determine the strongest relationships of a word among all other words, including itself (“cat” and “cat”).

The

cat

sat

on

the

mat

The

cat

sat

on

the

mat

The attention mechanism will provide a deeper relationship between words and produce better results.

For each Attention sub-layer, the original Transformer model runs not one but eight Attention mechanisms in parallel to speed up the calculations. This process is named “Multi-head Attention” and it provides:

* Analysis of sequences
* Elimination of recurrence
* Parallel execution
* Each attention head learns a different aspect of the same input sequence

## The Encoder Stack

The layers of the encoder and decoder of the original Transformer model are stacks of layers. Each layer of the Encoder stack has the following structure

Encoder layer 6

Encoder layer 5

Encoder layer 4

Encoder layer 3

Encoder layer 2

Encoder layer 1

Figure: Encoder layer Figure: Stack of 6 Encoder layers (instances)

Input Embedding

Inputs

Multi-head attention

Feed forward

Add and Normalize

Add and Normalize

Positional Encoding

Nx

The original Encoder architecture remains the same for all of the 6 layers of the Transformer model.

Each Encoder layer contains two main components (aka sub-layers) – Multi-headed Attention mechanism and a fully connected position-wise Feed-forward network. Notice that a residual connection surrounds each main sub-layer . These connections transport the unprocessed input of a sub-layer (Attention or Feed-forward) to a normalization function layer. By doing such bypass we make sure that key information such as positional encoding is not lost along the way. The normalized output of each layer is thus:

Though the structure of each of the six layers (instances) of the Encoder is identical, the content of each Layer is not strictly identical to the previous Layer. For example, the embedding sublayer is only present at the bottom level of the stack. The five layers (instances of the Encoder) do not contain an embedding layer and this guarantees that the encoded input is stable through all the layers.

Also, the multi-head attention sublayers perform the same functions from layer 1 to 6. However, they do not perform the same tasks. Each layer learns from the previous layer.

### Embedding and Positional Encoding

In any NLP model including the Transformer there are two important pieces of information which needs to be constructed for each word in the input sequence: the meaning of each word and its position of the input sequence. The Embedding layer encodes the meaning of the word and the Position encoding layer represents the position of the word. The Transformer combines these two encodings by combining them.

Transformers, unlike RNN, are inherently parallel - that is all words in a sequence are input in parallel. This means that the position information is lost and has to be added back separately.

Let us consider the following example input sequence:

The **black** cat sat on the couch and the **brown** dog slept on the rug.

Let us denote with the dimension of the word embedding vector. Usually, we are talking about large number of dimensions such as , , . So both the word **black** and **brown** are represented by a word embedding vector with size . [Vaswani et al](https://arxiv.org/abs/1706.03762) are using unit sphere projections to represent positional encoding that will remain limited in magnitude but nevertheless useful in accounting for the position in each word into the corresponding word embedding vector.

For each position in the dimensions of the word embedding vector we generate a pair of values which represent the projection on the unit sphere for the frequency corresponding to the position .

Here the projection is applied only to the even indices while the projection is applied only to the odd indices in the word embedding vector. Thus, the positional encoding vector is obtained by interleaving with . So for we can write:

where the binary operator denotes sequence interleaving along its index .

For and the word **black** in the example above we get:

for ( )

for ( )

Chart

Description automatically generated

With and the word **brown** in the example above we get:

for ( )

for ( )

Chart

Description automatically generated

**Definiton**: *Cosine similarity*

If it means the vectors are opposite in terms of semantic value, indicates they are semantically the same and indicates the vectors are uncorrelated in terms of semantic value.

If and then

Note that if we denote with the word embedding vector of the word **black** and with the word embedding vector of the word **brown** where both are computed with *word2vec* then

The encoding of the position has lower similarity value than the similarity value of the word embedding for the same words. The problem is how to add the positional encoding to the word embedding vectors.

The authors of the Transformer chose to implement positional encoding into the words by simply adding the word vectors with the positional encoding vectors . The new vector will be denoted with (which stands for Word Embedding with Positional Encoding) as shown on the Figure below.

“black”

+

=

“brown”

+

=

If we apply the cosine similarity function to the word vectors with positional encoding for **black** and **brown** we will get :

Thus, we have

Initially the word embedding vectors for **black** and **brown** indicate that the two words are very closely related semantically. The positional encoding of those two words showed that those are related semantically but had larger semantic distance compared to the word embeddings and finally the semantic distances between the sum of the and was found to be between the other two. This result can be interpreted as an assertion that the word vectors with positional encoding retain the information from the word embedding and positional encoding in them.

We may lose information of the word embedding after application of the positional encoding to it.

One way to preserve this information is to scale the embedding vector before we add the positional encoding to it as shown below:

### Encoder Self-Attention

Self-attention refers to the input sequence which pays attention to *itself*.

Attention input parameters are *Query*, *Key* and *Value*. All three parameters are similar structurally with each word in a sequence represented by a vector.

The input sequence is fed into the Input Embedding and Position Encoding, which produce encoded representation of each word in the input sequence that captures the position and meaning of each word. This is fed to all three parameters, Query, Key and Value in the Self-Attention in the first Encoder which then also produces an encoded representation of each word in the input sequence that now incorporates the attention scores for each word as well. As this passes through all Encoders in the stack, each Self-Attention module also adds its own attention scores into each word’s representation.

### Multi-Head Attention

We divide the dimension of the word vectors into heads. We denote the new vector size for each of the heads with .

Let us denote with the number of words in the input . With we will denote each individual word in . With we denote the matrix of word vectors with positional encoding added. Each row in that matrix represents word vector with positional encoding corresponding to an input word. Obviously is of size .

The output of each attention head is a matrix with size . The output of a multi-attention head is where the symbol denotes concatenation along the rows of the 8 matrices.

Attention for each word can be characterized with 3 parameters – a query vector , key vector , and a value vector . With multi-head attention the query, key and value vectors have size . For all words in the input we compose 3 matrices – the query matrix , the key matrix and the value matrix where each row is a corresponding vector for each word in the input:

; ;

Scaled dot product Attention is computed as:

A diagram of a software algorithm

Description automatically generatedFigure: Scaled dot product attention

Here the function represents the softmax function.

Formally, the standard softmax function is defined as:

where is matrix given with .

Here , obviously, is a square matrix which is symmetric. Each element of the Attention matrix represents the attention between the words and .

To obtain those 3 attention parameters we need to train the model for each attention head with each sub-network *query* , *key*, and *value* corresponding and their corresponding weight matrices , and . These three subnetworks are depicted on the Figure below.

A diagram of a machine

Description automatically generated

Figure adapted from Wikipedia [*Attention*](https://en.wikipedia.org/wiki/Attention_(machine_learning))*.*

On the Figure the sentence is sent to three parallel streams. On the right end a single context vector emerges as a result of applying the attention mechanism to the word . The single head word embedding size is . The neuron count for each of the three subnetworks is .

denotes the row word vector (including the positional encoding) for the word .

denotes the matrix of the word embeddings of all words in the sentence. In general the dimensions of are . The attention head includes three single layer subnetworks each having neurons. The weight matrices for each of the three subnetworks are , and , all sized as .

The query component is a vector of size corresponding to single word, the key and the value matrices are sized . The function represents the softmax function. The result of applying the softmax function to is the soft weights row vector with size . Multiplying with results in the context row vector with size .

With multi-head attention we split each word vector into chunks with size .

# References

[1] [Attention is All You Need , Ashish Vaswani et al, Google Brain, U of Toronto, 2017](https://arxiv.org/abs/1706.03762)

[2] [The Annotated Transformer, Harvard NLP, 2018](https://nlp.seas.harvard.edu/2018/04/03/attention.html)

[3] [The Annotated Transformer, Alexander M. Rush, Harvard U., 2018](https://aclanthology.org/W18-2509.pdf)

[4] [Attention in Machine Learning, Wikipedia](https://en.wikipedia.org/wiki/Attention_(machine_learning))