Transformers

Dr.Jawwad Ahmad Shamsi
Systems Research Laboratory
School of Computing,
National University of Computer and Emerging Sciences, Karachi Campus, Pakistan

April 29, 2024

1 Introduction

As discussed earlier, a sequence to sequence network, is a type of network, which takes a series of input and generates a series-based output. It essentially consist of an encoder and a decoder.

An encoder is a type of neural network, which takes an input and maps it into an encoding scheme. A decoder takes the encoded scheme as an input and decodes it to determine the desired output.

1.1 Examples of Seq 2 seq networks

There are many applications (and types) of seq 2 seq networks

Type	Example
one to many	Image captioning
many to one	Sentiment analysis
many to many	Machine Translation

In general, a seq 2 seq network follows an encoder decoder architecture.

Let's study the image captioning system in a little more detail .

It takes a fixed size vector as input and generates a sequence of any length as output.

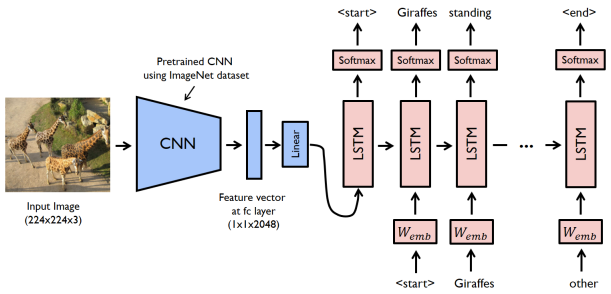

Figure 1: Image Captioning
[Jal]

Another example is sequence to vector model which takes a variable input and generates a fixed size output. A sentiment analysis system is an example of such a system [Rob].

A machine translation system is an example of variable size input and variable size output.

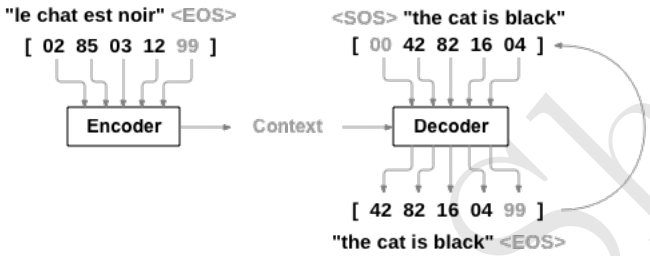

Figure 2: Machine Translation
[Rob]

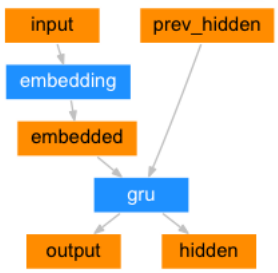

Figure 3: Encoder Network
[Rob]

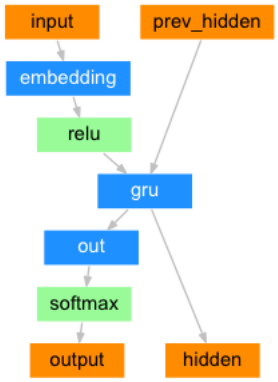

Figure 4: Decoder Network
[Rob]

An encoder in all the above mentioned examples is a system, which encodes the input in a given format, whereas a decoder generates the output in the desired format.

1.2 RNN vs LSTM

1. Slow in training
2. long sequences vanish or explode

LSTM solves the second problem
However, the first problem persists. LSTMs are slow and are difficult to train
LSTM are sequential. therefore they are slow.

2 Transformers

A transfotmer is a non-recuurence model. That is, unlike RNNs/LSTM/GRU which relies on recurrence, a transformer is not based on recurrence. A transformer architecture could be well understood from figure 5. It consists of an Encoder unit and a Decoder Unit. A transformer consists of six such units.

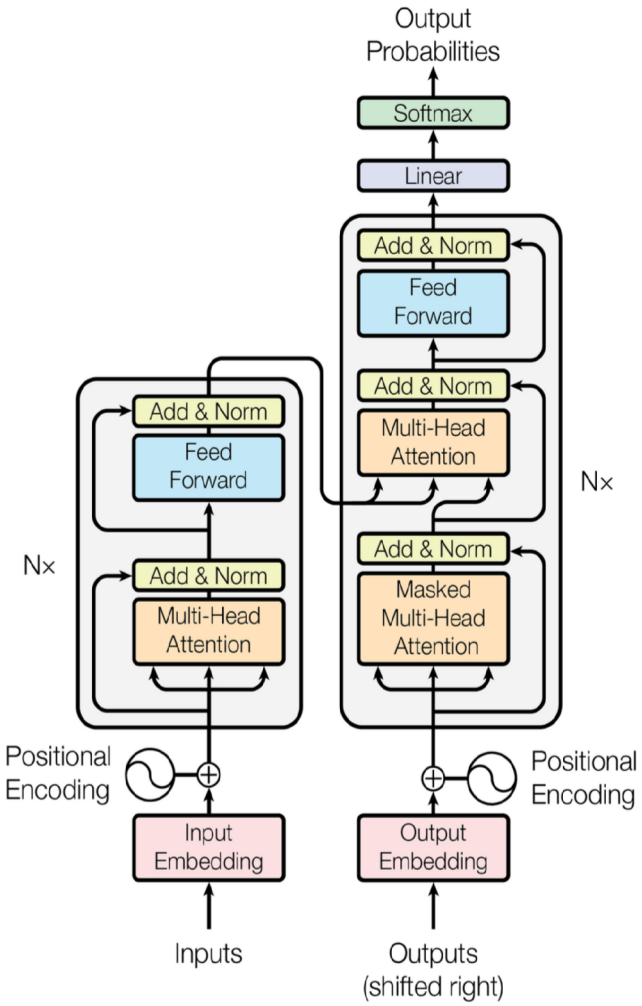

Figure 5: Transformers Architecture
[Max]

Figure 5 illustrates the basic architecture of transformers. The encoder consists of two blocks, i.e., a multi-head attention and a feed forward block. The decoder contains an additional block.

Let's study these blocks (units) in detail. Let's begin with an example of English to Urdu Translation.

2.1 Encoder

As per the research paper, there are six encoder units stacked together. Each unit has an Attention block and a Feed Froward block. There is an additional

process of converting word embeddings into positional encoding.

Since a transformer is a non-recurrent neural network, it cannot capture context related to positions. This information has to be captured through positional encoding.

1. Positional Encoding: The main purpose of positional encoding is to add context of positions in a sentence. This unit takes word embeddings as input and outputs positional embeddings. The process of positional encoding is explained in section 2.6

2. Multi-head Attention: This unit is responsible for generating attention vectors (weights). Multi-head implies that multiple head weight vectors are generated. The basic attention mechanism of a transformer is based on scaled dot-product attention. Details are mentioned in section. 2.8.

3. Add and Norm : The Add layer adds the two matrices, whereas the normalizaiton layer normalizes the matrices with respect to mean and standard deviation.

4. Feed Forward: A feed forward unit is connected to the attention block.It is used to transform the attention vector generated by the attention block into a form which is acceptable to the next encoder block or decoder block. The feed forward layer uses a linear layer and an activation function.

2.2 Decoder block

The purpose of the decoder is to predict the next word. During the training phase, we feed the output urdu sentence to the decoder. This is done by computing input embedding and adding the positional vector. Following are the main blocks on the decoder side:

1. Masked multi-head attention: This is the attention block on the decoder side. It is based on scaled dot-production. It generates attention vector for every word in the output language, i.e, Urdu. This attention vector will represent that how much each word in the output sentence is related to every word in the same sentence. However, it also involves masking. Masking means that decoder will mask the embedding for words, which have not been generated by the decoder.

2. Encoder Decoder Attention Block: There is another attention block. It takes input from the encoder and input from the above attention block. That is, it takes input from the encoder as well as the decoder side and determines attention vector from input to output. This blocks outputs attention vectors with respect to both English and Urdu. The output of this block is attention vectors for every English and Urdu sentences. These vector provides relationships with respect to both the languages.

3. Feed Forward Block: The purpose of this block is to make the output vector more digestable either to the next decoder block or to the linear layer.

4. Linear Layer: The linear layer used to expand dimensions for the output language.

5. Softmax Layer: This layer transforms the output into probability distributions. The word with the highest probability is selected.

The encoder decoder model is executed over several time steps until we reach the end of sentence.

2.3 Self Attention / Multi-head Attention

In the transformer, the self-attention mechanism allows each token to assess it's significance with every word in the entire sequence. The multi-head attention generates several heads , allowing an enriched mechanism.

2.4 Training

For training, we use sentences in both the languages. These sentences are translations of each other. For the English to Urdu Translation, we input the English sentence on the Encoder side, whereas on the decoder side, we input the Urdu sentence. However, the decoder input will be masked and shifted to the right by one position

The main reason for shifting is to ensure that our model does not learn to copy the decoder input.

If the sequence is not shifted right then the model will learn copying the decoder input. This is because for position j (character/word) in the encoder sequence, the model will learn to copy the decoder input for position j.

The shifting of the decoder input will ensure that we will train our model such that given the encoder sequence and the decoder sequence from 1, ..., j-1, the model can predict the jth sequence.

This process ensure that our model does not learn copy pasting of the translated sequence, instead it learns to predict the next word in the sequence

During training the first word of the decoded sentence is filled with a 'start of sentence' token. The end of sentence token is also attached to mark the end of the sentence.

For loss function, we will consider the target sentence (i.e., the Urdu sentence without shifting and with the end of token sentence.

Further, if during the training process, a deocder estimates incorrectly, even then the corrected word is given in the input and the incorrect word is not considered. This phenomenon is called **teacher forcing**. The rationale is that similar to a teacher, which tries to teach a student by giving hints.

The whole process of training can be understood from figure 6

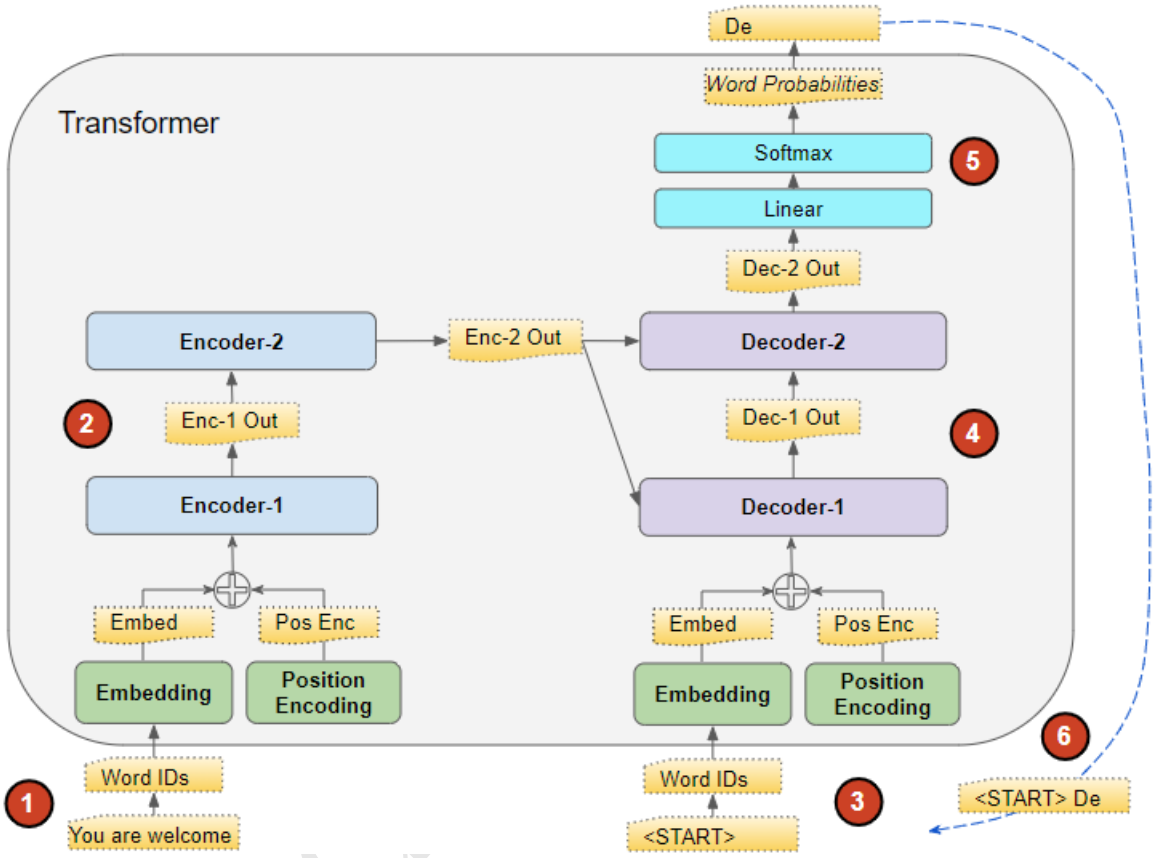

Figure 6: Transformer Training
[Dosa]

1. The sequence of words at the input layer is converted into Embeddings (such as word2vec). This sequence is then transformed with Positional Encoding and fed to the Encoder. Since the transformer is not a recurrent network, it cannot learn positions.

2. At the encoding site, there is a stack of encoders (N=6 in the original paper), which produces an encoded representation.

3. At the Decoder site, the words which have been generated or predicted so far are sent to the decoder as a sequence. This is converted into embeddings with positional encoding. For the very first word, a start of sentence tag is sent. In this way, the decoder input is considered as shifted right.

4. A stack of decoders process this. The stack takes input from the encoded layer output (encoded representation). This produces the encoded representation of the output or the target sequence.

5. The decoder outputs probabilities. The total number of nerons at the output layer is the number of unique words in the corpus. In this manner, a sequence is generated at the output sequence.

6. AT the output, loss function is used to compare the predicted words with the actual output. This is used during back-propagation.

2.5 Inference or Testing

Inference or testing the transformer the machine translation problem is trivial. On the decoder side, we begin with the start of the sentence token and keep adding the next word with the highest probability [Max19].

The testing (or inference) part is similar except that instead of sending the actual sequence, we share the predicted (So far) sequence.

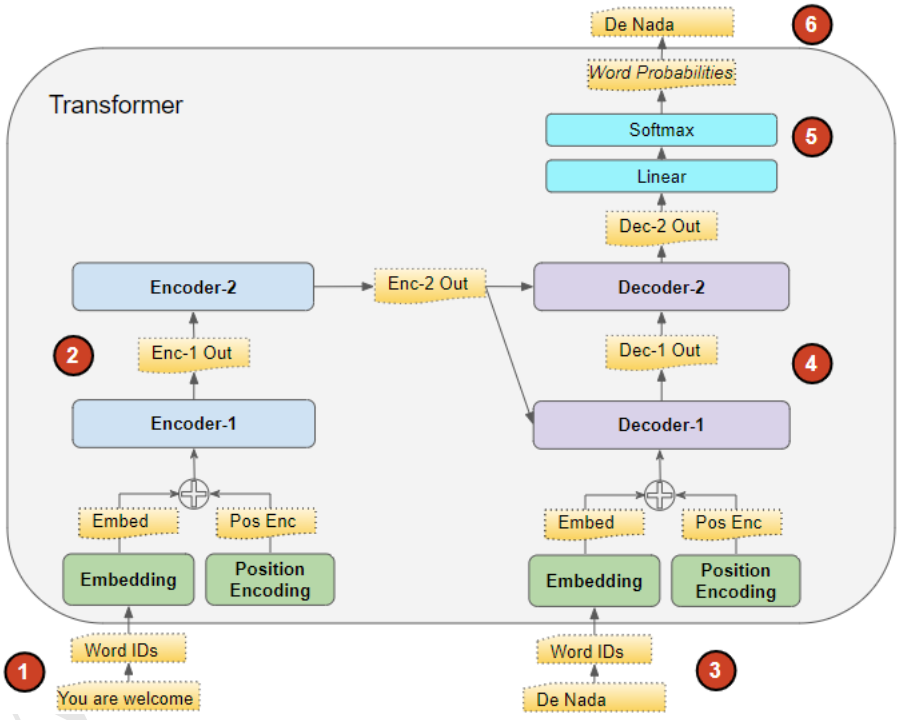

Figure 7: Transformer Inference
[Dosa]

Detail steps are mentioned below:

1. Input the English sentence (encoder output) with the start of the sentence in the decoder. It will output the first word in the decoder sequence (Urdu).

2. Add the first word (output above) in the sequence of decoder

3. Now input the encoded sentence with the above mentioned decoder sequence (start of the sentence and the first word) to predict the next word in the sequence.

4. iterate by appending the decoder sequence and predicting the next word in the decoder sequence by considering the complete encoder sequence and the current state of the decoder sequence.

2.6 Positional Encoding

It captures the positional information of words in a sentence. Each position has a unique value in the sentence. For long sequences, indices can grow, therefore a vector is used instead of an index.

This layer outputs a matrix, which is used for positional encoding. Each row of the matrix represents positional information of the sequence.

For instance,

AJ's dog is a cutie [position 2] AJ looks like a dog [position 5]

$$\text{P(k, 2i)} = \sin\left(\frac{k}{n^{2i/d}}\right) \quad (1)$$

$$\text{P(k, 2i+1)} = \cos\left(\frac{k}{n^{2i/d}}\right) \quad (2)$$

L: length of the sequence

k: It is the position of the object in the input sequence, $0 \leq k < L$

d: It is the dimension of the output embedding space

P(k, j): This denotes the position function, which is used to compute positional embedding. (k,j) denotes input sequence to index (k,j) of the positional matrix

n: It is a user defined scalar. The original paper uses the value of 10,000 .

i: It is use to map column indices $0 \leq i < d/2$

Both sine and cosine functions are used for mapping.

Example Sentence: I am a robot d=4 n=100

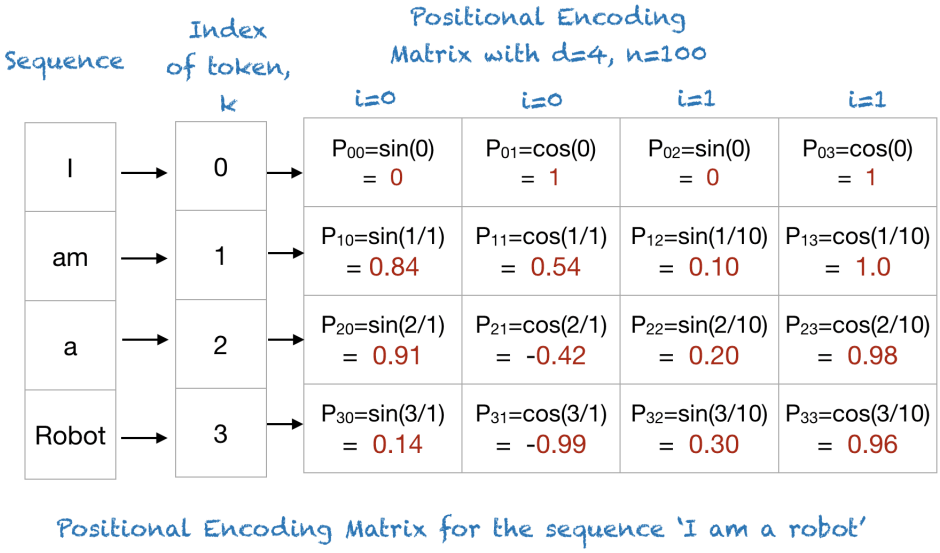

Figure 8: Encoder Network
[Sae]

The figure computes values based on radian degrees.

2.7 The Process

Let us understand the mechanism of encoding and decoding through these steps:

1. Input Sentence
2. Convert into embedding
3. add positional encoding
4. The attention mechanism creates attention vectors. Interactions with self may have high weights. But it is not much of a use. So create an average attention vector for every word.Multi-head attention because multiple vectors are created.
5. The attention vector is passed through a network , one vector at a time. Since each vector is independent, we can use parallelization. All words can be passed at the same time into the encoder block.The output will be a set of encoded vectors for every word.
6. On the decoder side, we obtain the embedding of the Urdu word to obtain meaning.
7. Add positional embedding to compute the context.

8. This is passed to the attention block. It has been termed as masked attention. This is because while generating the next word, we can use all the words of the English sentence but only the previously generated words of the french sentence. (There will not be any learning if we are going to use all the words of the French sentence. It will just be spit out). Matrix mask these later words by transforming them to zeros so the attention network cant use them

9. The next attention block, which is an encoder-decoder attention block generates a similar attention mechanism for every english and Urdu words.

10. This attention vector is passed to a feed forward layer and softmax layer to predict the next word.

2.8 Scaled-dot Attention

Transformer uses self attention mechanism. In such a mechanism, the input sequence learns attention weights (attention) for it's own input sequence (self). In other words, it is an attention mechanism, which relates different positions of a single sequence and determines attention representation of the same sequence.

The self-attention mechanism of transforemers are based on scaled-dot attention.

We will now learn the scaled-dot attention mechanism of transformers. The mechanism is based on query Q, Keys k, and Values V. This is similar to the concept of a web search. When we query a web search engine, it will map the query to a set of keys. Each key can consists of title of the page, description etc. whereas the search engine will select web pages with best results or best values. The same notion is applied to the attention mechanism of transformers. Figure 9 shows this concept. In transformers attention (weights) are used to match keys against values.

In the original transformers paper, keys are used to define the attention weights to look at the data and the values are used as the information that will actually be obtained [Ada21].

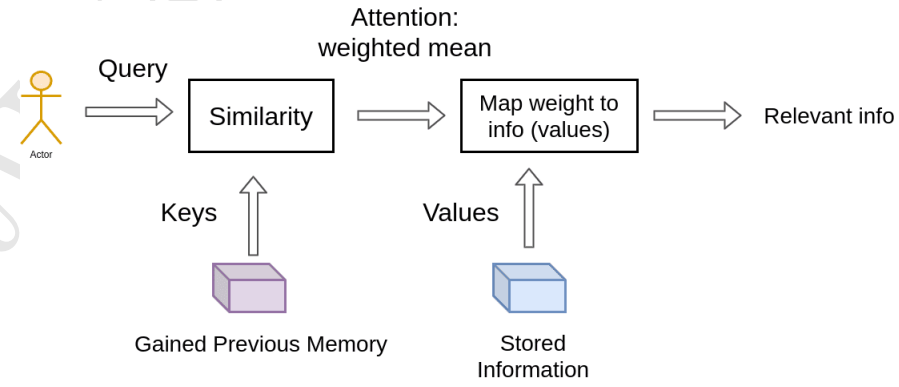

Figure 9: Query vs Key vs Value
[Ada21]

Figure 11 illustrates the scaled dot attention mechanism of transformers.

Let us consider an example. Hello I like you. A trained self-attention layer will associate the word like with the words I and you with a higher weight than the word Hello. Linguistically, the three words share a subject-verb-object relationship as well [Ada21].

Self-attention
Probability score matrix

	Hello	I	Like	you
Hello	0.8	0.1	0.05	0.05
I	0.1	0.6	0.2	0.1
Like	0.05	0.2	0.65	0.1
you	0.2	0.1	0.1	0.6

Softmax(Attention)
equation

Figure 10: Self Attention Example

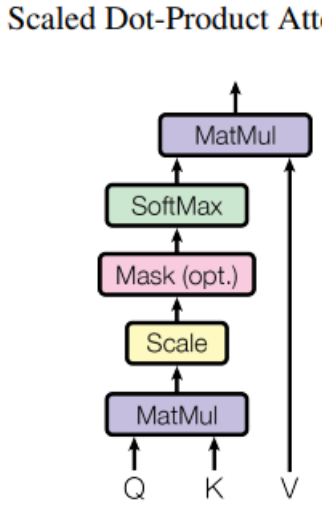

Figure 11: Scaled dot attention
[Vas+17]

Attention mappings can be defined as a set of query Q and a set of key K value V pairs.

Let us assume tha q and k denotes vectors of dimension d_k, containing the queries and keys, respectively, and v denoting a vector of dimension d_v containing the values. Therefore, Q, K, and V denotes matrices of queries, keys and values, respectively.

We can describe scaled.dot attention as follows

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3}$$

We initialize q, and k as independent random variables with mean =0 and variance =1 , then their dot product,

$$q \cdot k = \sum_{i=1}^{d_k} u_i v_i \tag{4}$$

has mean 0 and variance d_k

Since we would prefer these values to have variance =1 , we divide by $\sqrt{d_k}$.

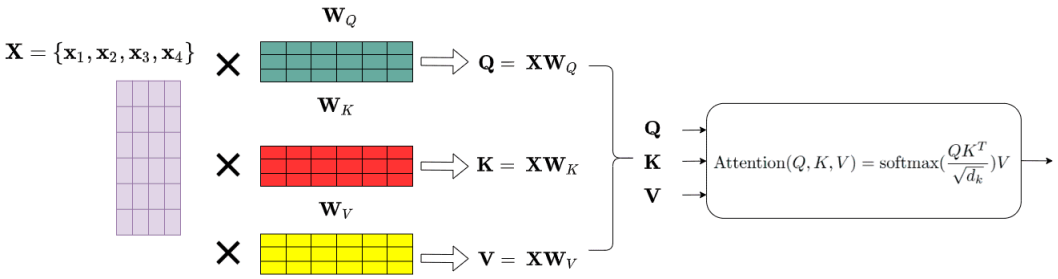

Figure 12: Key-query-value
[Ada21]

It is an attention mechanism where the dot products are scaled down by a factor of $\sqrt{d_k}$

In general, attention mappings are denoted as function of query and a set of key-value pairs. Transformers use a Scaled Dot-Product Attention to obtain the context vector.

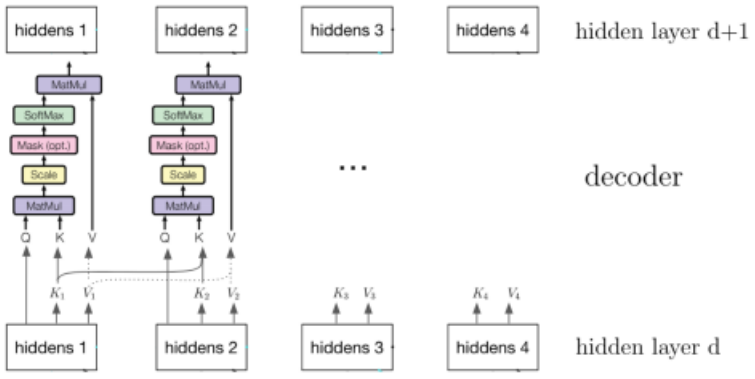

Figure 13: Scaled dot attention Explained details
[GB]

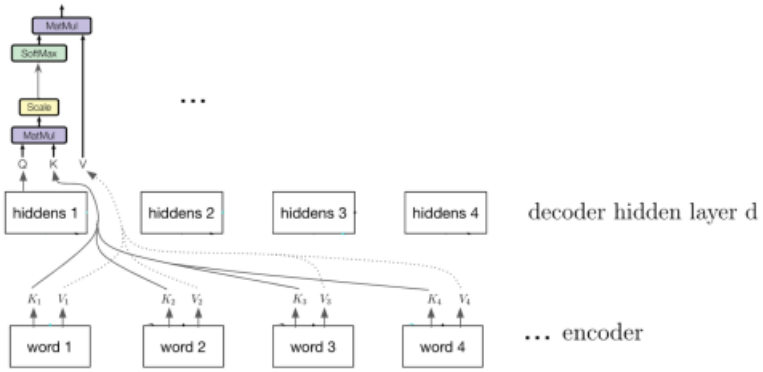

Figure 14: Scaled dot attention Explained
[GB]

Figure 13 and 14 illustrates the concept of self-attention in transformers.

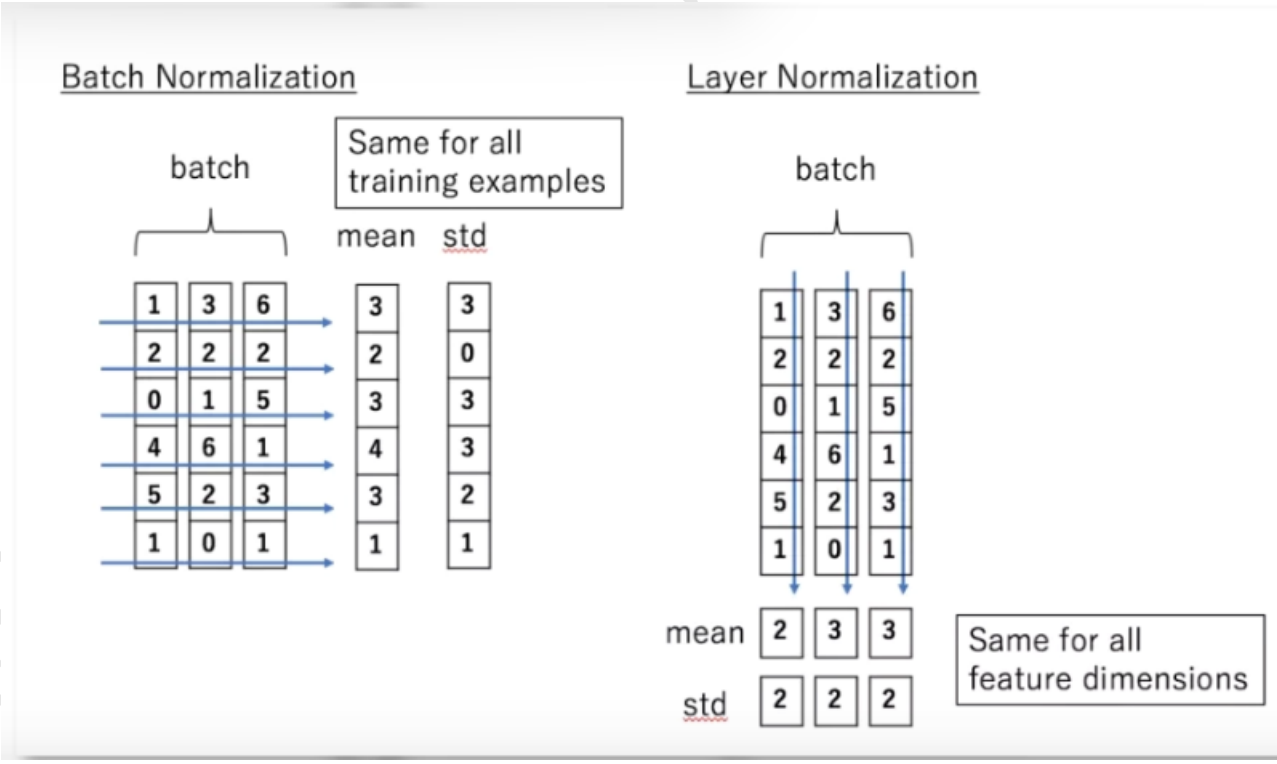

Figure 15: Layer Normalization
[Sha]

Figure 15 illustrates the concept of layer normalization.

2.9 Single-head Attention

Q,K,V are abstract vectors. They extract different components of an input word.

We have Q,K,V vectors for every single word.

we use these to compute attention vectors for every word using

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

Figure 16: Scaled dot attention Explained details [GB]

The transformer mechanism utilizes attention at three instances [Unk]:

1. : Encoder-decoder attention layers. The values are taken from the previous decoder layer, whereas the queries and keys from the encoder output. This attention instance learns weights for each position in the decoder with respect to each position in the input sequence.

2. The self-attention layer contained in the encoder generates random attention matrix values for query, key, and value. Further iterations allow learning of these attention weights. Subsequent layers of encoders takes input from the previous encoder layers.

3. : Self-attention in the decoder is similar. All the values of queries, keys, and values are taken from the previous layer. The attention weights are masked up to the values for the position decoded so far.

2.10 Multi-head Attention

The concept of Multi-head attention can be understood from figures 17, 18 19 and 20 The self-attention mechanism discussed above gives more weightage to the own word. However, our main task is to learn weights with other words in the same sentence. If W^Q W^K and W^V denotes weighted matrices for query, key, and value then W_i^Q W_i^K and W_i^V for the ith head. Here head denotes an iteration for a set of weights. We generate multiple heads in parallel (because each head is independent) (see figure 21. These heads are generated multiple times in order to take a combined and concatenated affect of the attention weights. In the original transformers paper, eight heads have been used [Vas+17].

Linearity in the multi-head means a bunch of neurons connected without the activation function. The linearity fulfills two purposes [Int]:

1. Mapping Input to Output

2. Reduce dimensions

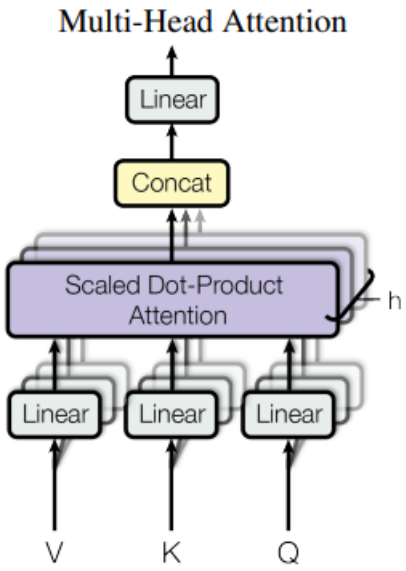

Figure 17: Multi Head Attention
[Vas+17]

$$\text{head} = \text{Attention}(QW^Q, KW^K, VW^V)$$

Figure 18: Head in an attention model
[Lih19]

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

Figure 19: ith Multi-head
[Lih19]

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$

Figure 20: Concatenation in Multi-head
[Lih19]

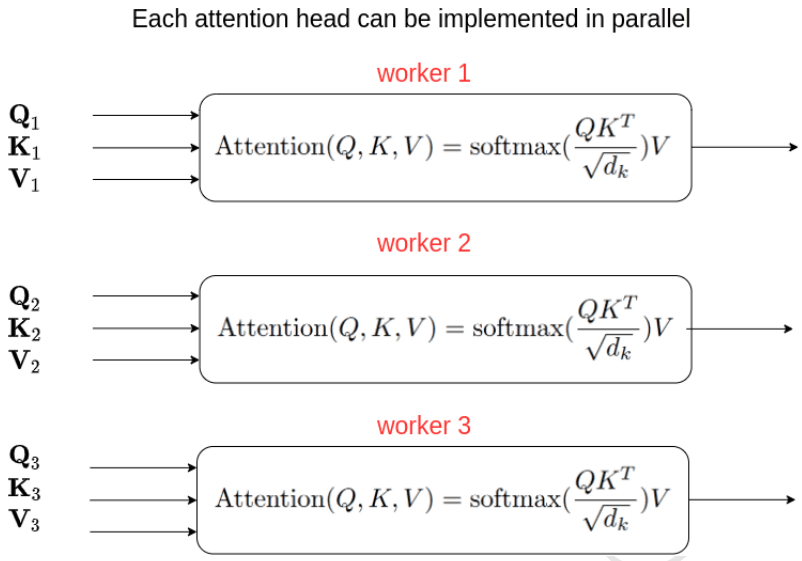

Figure 21: Parallelism in Multi-head attention model [Ada21]

Figure 22 illustrates the concept of computing multi-head attention.

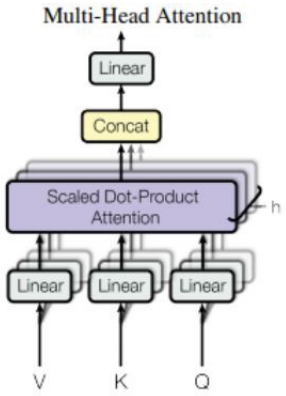

Figure 22: Multi-head-Attention Model [Lih19]

For multi-head attention, we have multiple weight matrices, i.e., one for each head.

Since the Feed forward Neural Network is only expecting one attention vector per word, we use another weighted matrix W^Z to compute final attention vector per word.

Multi-head attention ensures that different parts parts of the sequence are attended differently. It provides improved learning for positional information as each head will attend different segments of the input and will learn different contextual information.

2.11 Residual Connection

Transofrmers also has residual connections. Residual connection serves two main purposes:

1. Knowledge Preservation During forward propagation the input gets modified considerably by the time it reaches the last layer. This may result in loss of information as compared to the input layer in the system. Residual connection will ensure that old information will not get lost. We can add two pieces of information to get the final output.

2. Vanishing Gradient Problem

After every layer, we apply some form of normalization. this smoothens out loss surface making it easier to optimize. We prefer layer normalization.

By leveraging parallelism, we can pass all the words together in our encoder block and it will output all the words together.

2.12 Types of Transformers

Encoder Only

Decoder Only

Encoder Decoder

Further reading [Dosb].

References

[Vas+17] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[Lih19] Anusha Lihala. "Attention and its different forms". In: *Towards Datascience* (2019).

[Max19] Maxime. "What is a Transformer". In: *Inside Machine Learning* (2019).

[Ada21] Nikolas Adaloglou. "Transformers in Computer Vision". In: *https://theaisummer.com/* (2021).

[Dosa] Ketan Doshi. *Transformers Explained Visually (Part 1): Multi-head Attention, deep dive.* URL: `https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452`. (accessed: 01.05.2022).

[Dosb] Ketan Doshi. *Transformers Explained Visually (Part 3): Multi-head Attention, deep dive*. URL: https://towardsdatascience.com/transformers-explained-visually-part-3-multi-head-attention-deep-dive-1c1ff1024853. (accessed: 01.05.2022).

[GB] Roger Grosse and Jimmy Ba. *Neural Networks and Deep Learning*. URL: https://www.cs.toronto.edu/~rgrosse/courses/csc421_2019/. (accessed: 04.05.2022).

[Int] Hedu - Math of Intelligence. *A Visual Guide to Neural Networks*. URL: https://www.youtube.com/watch?v=mMa2PmYJlCo. (accessed: 01.05.2022).

[Jal] JalFaizy. *Automatic Image Captioning using Deep Learning (CNN and LSTM) in PyTorch*. URL: https://www.analyticsvidhya.com/blog/2018/04/solving-an-image-captioning-task-using-deep-learning/. (accessed: 01.01.2022).

[Max] Maxine. *What is a Transformer*. URL: https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04. (accessed: 01.05.2022).

[Rob] Sean Robertson. *NLP FROM SCRATCH: TRANSLATION WITH A SEQUENCE TO SEQUENCE NETWORK AND ATTENTION*. URL: https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html. (accessed: 01.01.2022).

[Sae] Mehreen Saeed. *Positional Encoding in Transformers*. URL: https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/. (accessed: 01.05.2022).

[Sha] Palash Sharma. *Keras Normalization Layers- Batch Normalization and Layer Normalization Explained for Beginners*. URL: https://machinelearningknowledge.ai/keras-normalization-layers-explained-for-beginners-batch-normalization-vs-layer-normalization/. (accessed: 01.05.2022).

[Unk] Unknown. *Self Attention in NLP*. URL: https://www.geeksforgeeks.org/self-attention-in-nlp/. (accessed: 01.05.2022).