VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY

UNIVERSITY OF SCIENCE

INTRODUCTION TO

NATURAL LANGUAGE PROCESSING

(CSC15006 – 21CNTTHUC)

CHINESE-TO-VIETNAMESE MACHINE TRANSLATION

For ancient texts using T5-translate-vietnamese-nom model

LECTURER	NGUYỄN HỒNG BỬU LONG
TEACHING ASSISTANT	LƯƠNG AN VINH
STUDENT	21127518 NGUYỄN VŨ MINH KHÔI
	21127621 ÂU DƯƠNG KHANG

Contents

I. Transformer:

1. Transformer:

Transformer is a deep learning model introduced in 2017, mainly used in the fields of *natural language processing (NLP) and computer vision (CV).*

Like recurrent *neural networks (RNNs)*, Transformers are designed to process sequential data, such as natural language, for tasks such as statistical machine translation or automatic summarization.

2. Formation process:

However, unlike RNNs, Transformers do not require sequential data to be processed in order. For example, if the input data is a natural language sentence, the Transformer does not need to process the beginning of the sentence before the end. Due to this feature, Transformers allow for multiple parallel computations and thus reduce training time.

Transformers were introduced in 2017 by a group of authors at Google Brain and are increasingly becoming the model of choice for NLP problems to replace RNN models such *as Long Short-Term Memory (LSTM).* Transformers have *the ability to train in parallel*, allowing for training on larger datasets. This opened the door to the development of pre-trained models such *as BERT (Bidirectional Encoder Representations from Transformers)* and *GPT (Generative Pre-trained Transformer).* These models are trained with large language datasets such as the Wikipedia Corpus and Common Crawl, and can be fine-tuned for specific tasks.

3. How it works:

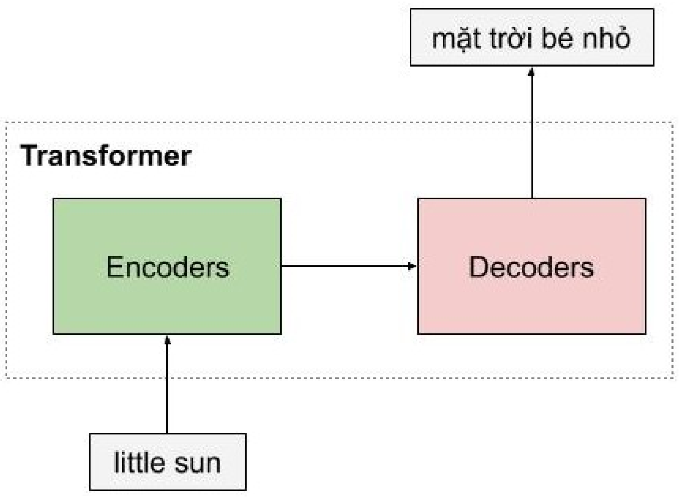
The main idea of Transformer is still to apply the Attention mechanism, but at a more complex level.

4. Overview of the Transformer model:

4.1. Model architecture

Like other machine translation models, the overall architecture of the Transformer model consists of two main parts: Encoder and Decoder. The Encoder is used to learn the representation vector of the sentence with the expectation that this vector carries the perfect information of that sentence. The Decoder performs the function of converting the representation vector into the language to be translated.

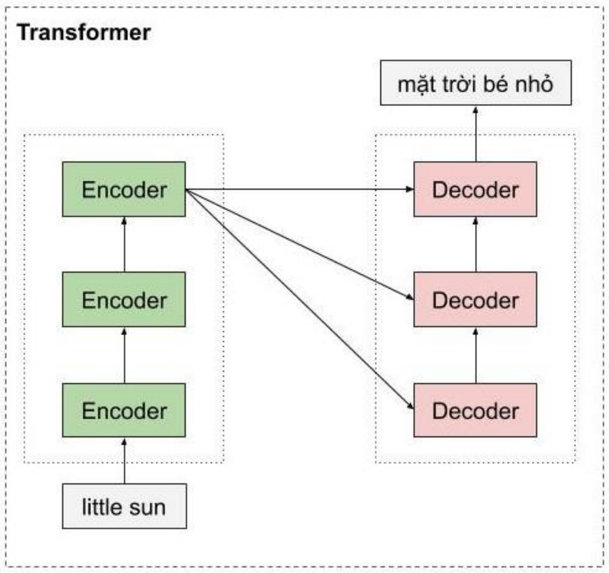
In the example below, the Encoder of the Transformer model receives an English sentence, and encodes it into a semantic representation vector of the sentence **"little sun"**, then the Decoder model receives this representation vector, and translates it into the Vietnamese sentence **“mặt trời bé nhỏ".**

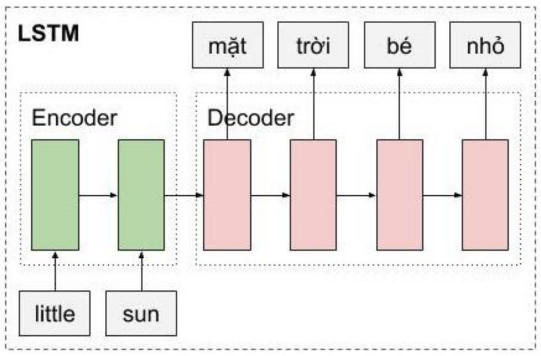


4.2. Advantages

One of the advantages of Transformer is that this model has the ability to process words in parallel. As you can see, the Encoders of the Transformer model are a type of *feedforward neural nets*, including many other Encoder layers, each of which processes words simultaneously.

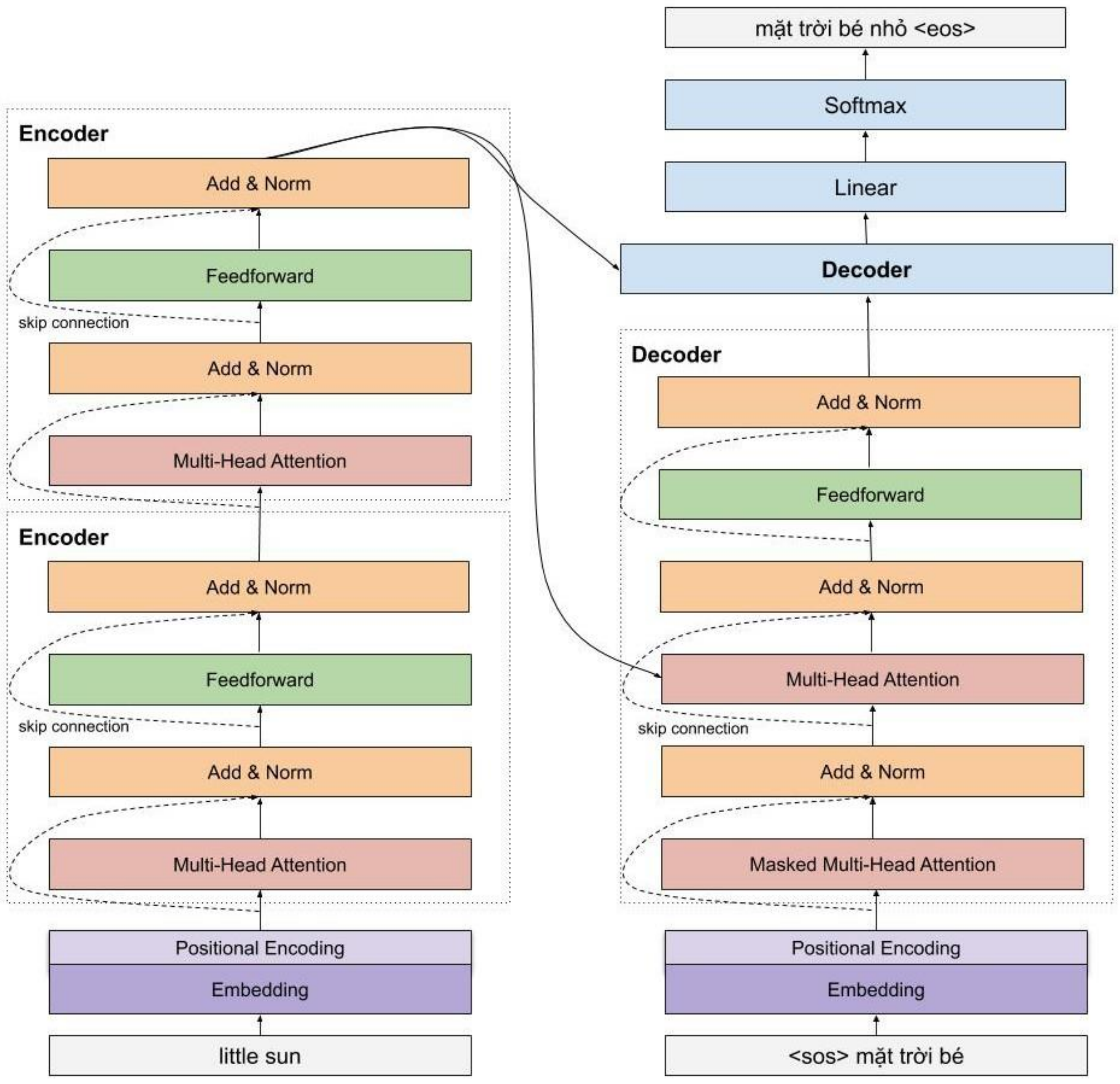
Meanwhile, with the LSTM model, the words must be processed sequentially. In addition, the Transformer model also processes the input sentence in 2 directions without having to stack another LSTM image like in the *Bidirectional LSTM* architecture.





4.3. Overview

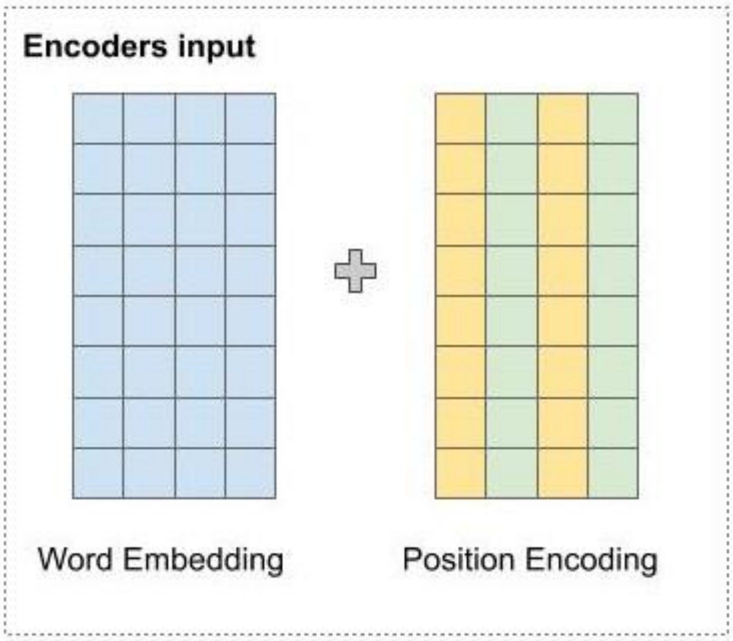
A general and detailed look at some extremely important parts such as *Sinusoidal Position Encoding*, *Multi-Head Attention* of the Encoder, and the Decoder's architecture is very similar to that of the Encoder.



5. Embedding Layer with Position Encoding:

5.1. Proposed method Sinusoidal Position Encoding:

The method proposed by the author does not suffer from the limitations we just mentioned. The positions of the words are encoded by a vector of the size of the word embedding and are added directly to the word embedding.



Specifically, at even positions, the author uses the sine function, and at odd positions the author uses the cosine function to calculate the value at that direction.

$$p_t^i = f(t)^i = \begin{cases} sin(w_k * t) & \text{if } i = 2k \\ cos(w_k * t) & \text{if } i = 2k+1 \end{cases}$$

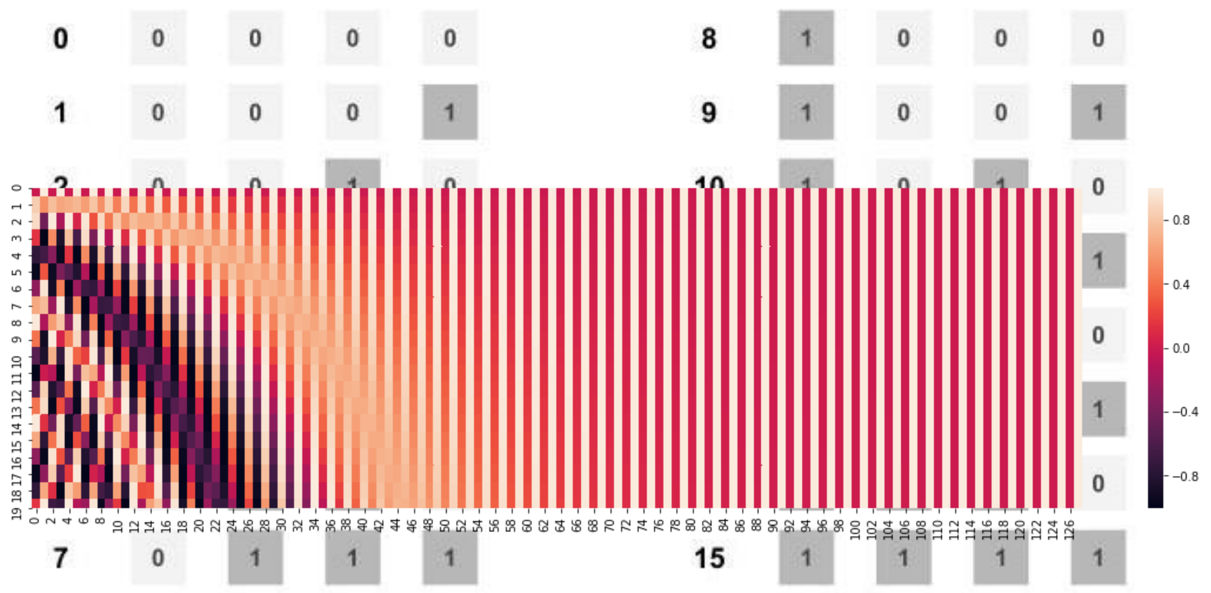
In which:

$$w_k = \frac{1}{10000^{2k/d}}$$

In the figure below, the author's way of calculating Position Encoding is illustrated. Suppose we have a word embedding with 6 dimensions, then Position Encoding also has 6 dimensions. Each line corresponds to a word. The value of the vectors at each position is calculated according to the formula in the figure below.

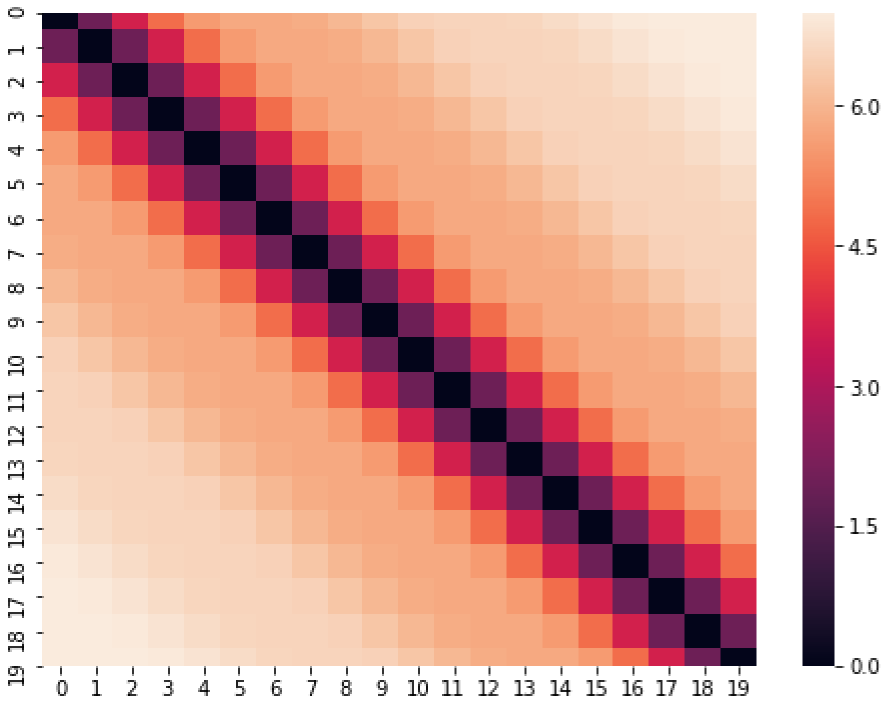
	0	1	2	3	4	5
0	sin($\frac{1}{1000^{0/512}}$ x 0)	cos($\frac{1}{1000^{0/512}}$ x 0)	sin($\frac{1}{1000^{2/512}}$ x 0)	cos($\frac{1}{1000^{2/512}}$ x 0)	sin($\frac{1}{1000^{4/512}}$ x 0)	cos($\frac{1}{1000^{4/512}}$ x 0)
1	sin($\frac{1}{1000^{0/512}}$ x 1)	cos($\frac{1}{1000^{0/512}}$ x 1)	sin($\frac{1}{1000^{2/512}}$ x 1)	cos($\frac{1}{1000^{2/512}}$ x 1)	sin($\frac{1}{1000^{4/512}}$ x 1)	cos($\frac{1}{1000^{4/512}}$ x 1)
2	sin($\frac{1}{1000^{0/512}}$ x 2)	cos($\frac{1}{1000^{0/512}}$ x 2)	sin($\frac{1}{1000^{2/512}}$ x 2)	cos($\frac{1}{1000^{2/512}}$ x 2)	sin($\frac{1}{1000^{4/512}}$ x 2)	cos($\frac{1}{1000^{4/512}}$ x 2)

To explain how the positional representation proposed by the author can encode positional information of a word, Imagine we have numbers from 0-15. We can see that the rightmost bit changes the fastest every 1 number, and then the 2nd rightmost bit, changes every 2 numbers, and so on for the other bits.



In the author's proposed formula, we also see that the sine and cosine functions have a frequency graph and this frequency decreases in increasing dimensions. See the figure below, in dimension 0, the continuously changing value corresponds to continuously changing color, and this changing frequency decreases in increasing dimensions.

So we can feel that the author's representation is quite similar to the way integers are represented in binary, so we can represent word positions in the same way.



We can also see the distance matrix of the position vectors as shown below. Obviously, the distance vectors represent the distance between two words. The farther apart the two words are, the greater the distance.

In addition, a property of the author's proposed method is that it allows the model to easily learn the relative relationship between words. Specifically, the position representation of the word **"t" + offset** can be converted to the position representation of the word **"t"** by a linear transformation based on the rotation matrix.

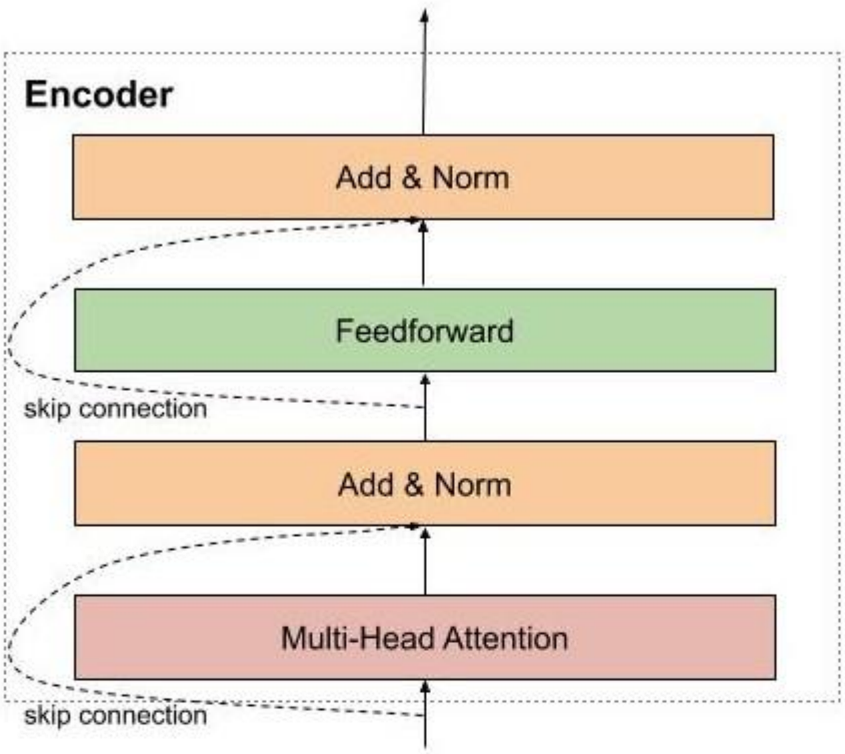
To easily visualize how the author's proposed method works, we can imagine the sine and cosine functions as the second and minute hands on a clock. With these two hands, we can represent 3600 positions. And at the same time, we can immediately understand why the representation of the word **"t" + offset** and the word t can be easily converted to each other.

6. Encoder:

6.1. Encoder:

The encoder of the Transformer model can include many similar Encoder layers. Each encoder layer of the Transformer includes 2 main components: Multi - Head Attention and Feed Forward Network, in addition to *Skip connection and Normalization layer*.

In these 2 main components, you will be more interested in Multi - Head Attention because it is a new layer introduced in this article, and it is what makes the difference between the LSTM model and the Transformer model that we are studying.

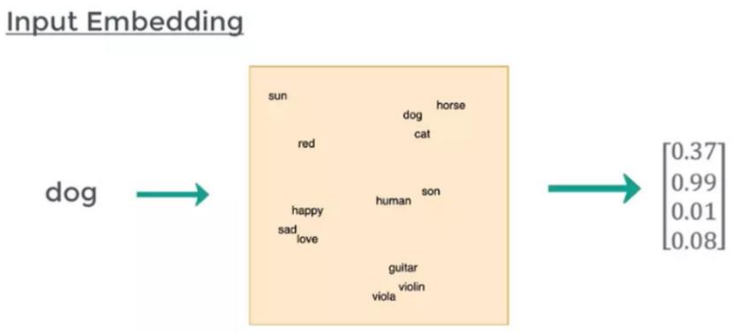


The first encoder will receive the word representation matrix which has been added with positional information through Positional Encoding. Then, this matrix will be processed by Multi-Head Attention. Multi-Head Attention is essentially Self-Attention, but to make the model able to process more phrases in the sentence, the author simply uses multiple Self-Attentions.

6.2. Components of Encoder:

6.2.1. Input embedding:

Computers do not understand words but can only read numbers, vectors, matrices; therefore, we must represent words in vector form, called Input Embedding. This ensures that words with similar meanings have similar vectors. Currently, there are quite a few pretrained word embeddings such as *GloVe, Fasttext, gensim Word2Vec,..*

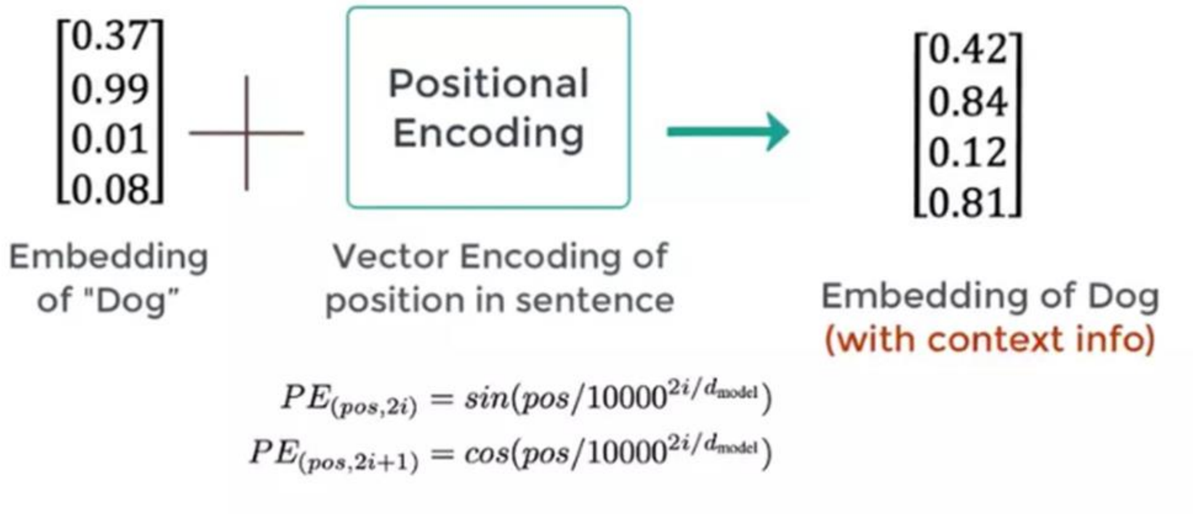


6.2.2. Positional encoding:

Word embeddings help us to represent the semantics of a word to some extent, however, the same word in different positions of the sentence has different meanings. That is why Transformers has an additional Positional Encoding part to inject more information about the position of a word.

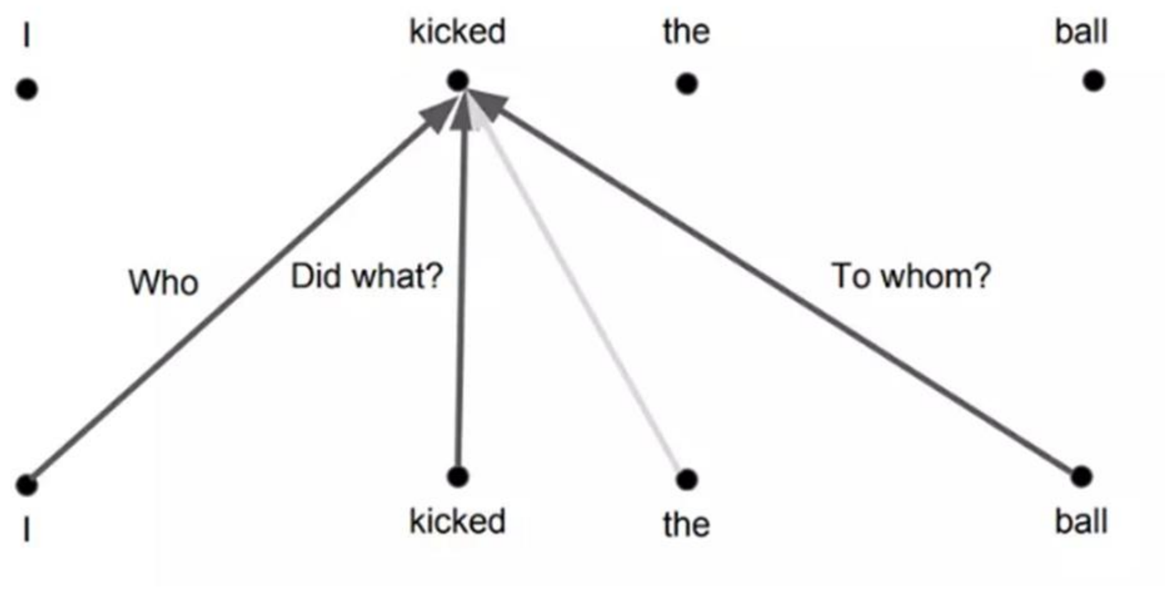
$$PE_{(pos,2i)} = \sin\left(pos/10000^{2i/d_{\text{model}}}\right)$$
$$PE_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d_{\text{model}}}\right)$$

Whereas pos is the position of the word in the sentence, PE is the value of the i-th element in the embeddings with the model length. Then we add the **PE vector** and the **Embedding vector:**


[0.37 0.99 0.01 0.08]
Embedding of "Dog"
Positional Encoding
Vector Encoding of position in sentence
[0.42 0.84 0.12 0.81]
Embedding of Dog
(with context info)
PE(pos,2i) = sin(pos/10000^(2i/dmodel))
PE(pos,2i+1) = cos(pos/10000^(2i/dmodel))

6.2.3. Self - attention:

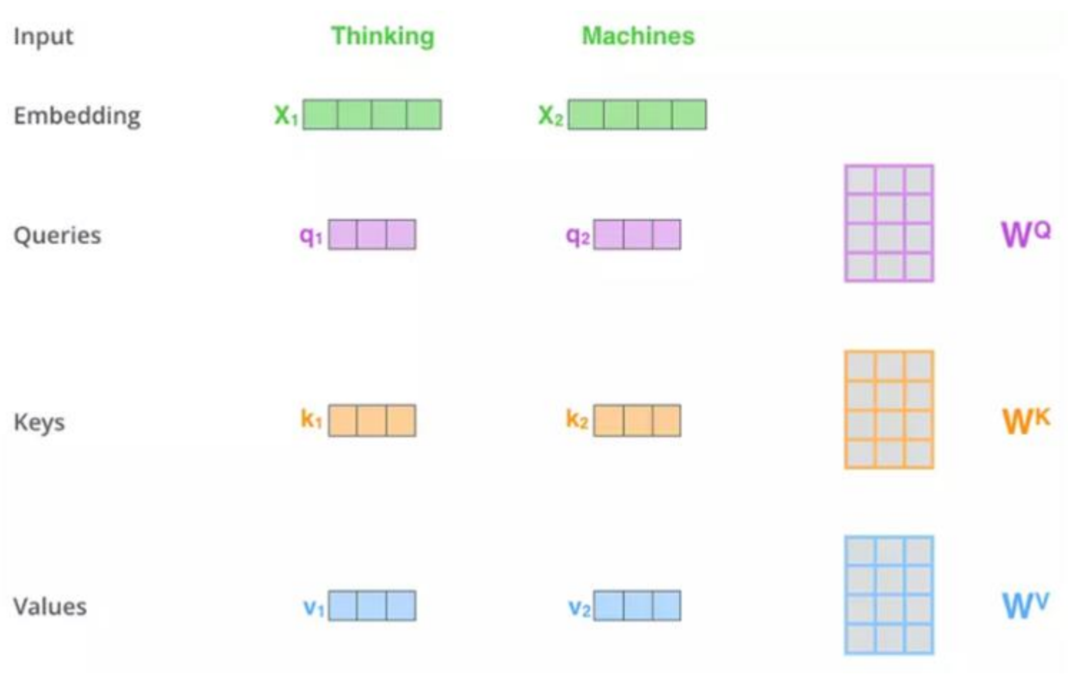
Self - attention is the mechanism that helps Transformers "understand" the relationship between words in a sentence. For example, how is the word **"kicked"** in the sentence **"I kicked the ball"** related to other words? Obviously, it is closely related to the word **"I"** (subject), **"kicked"** is itself so it will always be "strongly related" to **"ball"** (predicate). In addition, the word **"the"** is a preposition so the connection with the word **"kicked"** is almost non-existent. So Self - attention extracts these "relationships" by:



Returning to the overall architecture above, we can see that the input of the Multi-head Attention module (essentially Self-attention) has 3 arrows, which are the 3 vectors **Queries (Q)**, **Keys (K)** and **Values (V)**. From these 3 vectors, we will calculate the **vector attention Z** for a word according to the following formula:

$$Z = \text{softmax}\left(\frac{Q \cdot K^T}{\sqrt{Dimension\ of\ vector}\ Q, K \text{ or } V}\right) \cdot V$$

This formula is quite simple, it is done as follows. First, to get 3 vectors Q, K, V, input embeddings are multiplied with 3 corresponding weight matrices (tuned during training) **W_Q, W_K, W_V.**

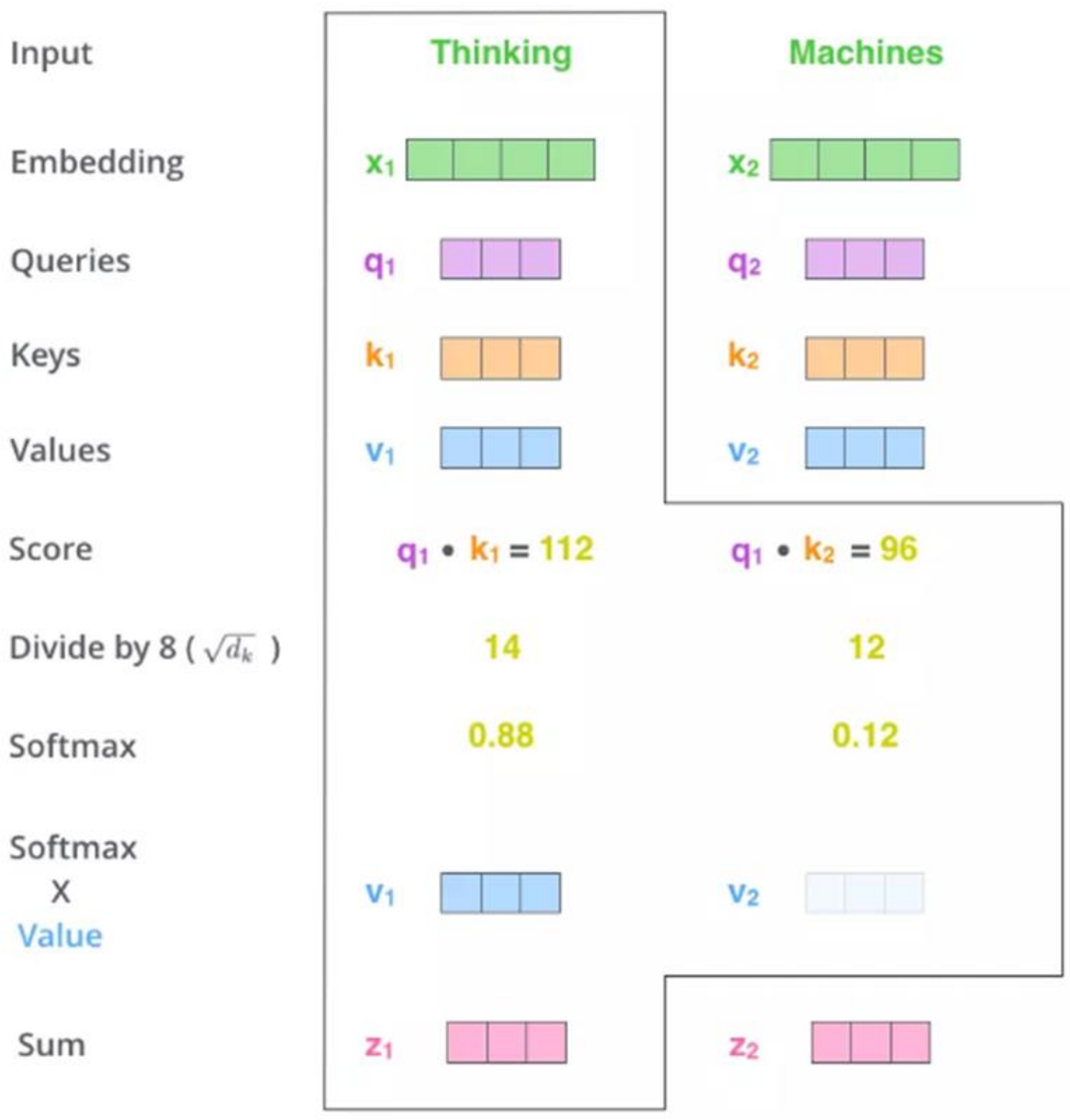


At this point, the K vector acts as a key representing the word, and Q will query the K vectors of the words in the sentence by convolution with these vectors. The purpose of convolution is to calculate the relevance between the words. Accordingly, 2 related words will have a large "Score" and vice versa.

The second step, the "Scale" step, is simply dividing the "Score" by the square root of the dimension of Q/K/V (in the figure, divided by 8 because Q/K/V are 64-D vectors). This helps the "Score" value not depend on the length of the Q/K/V vector

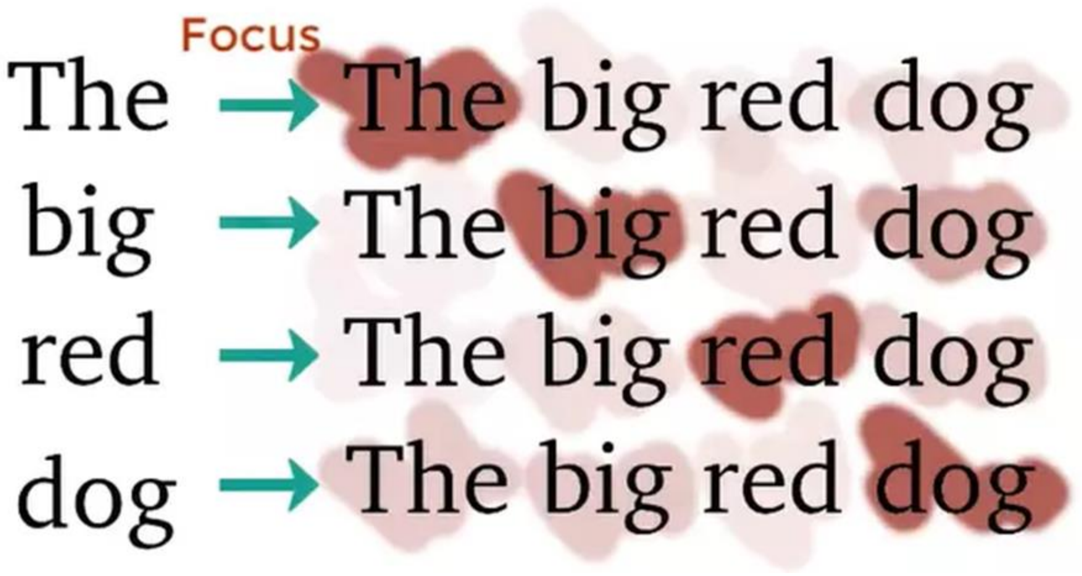
The third step Softmaxes the previous results to obtain a probability distribution over the words.

The fourth step, we multiply that probability distribution with the V vector to remove unnecessary words (small probability) and retain important words (large probability). In the final step, the vectors V (which have been multiplied by the Softmax output) are added together, creating the attention vector Z for a word. Repeating the above process for all words gives us the attention matrix for a sentence.



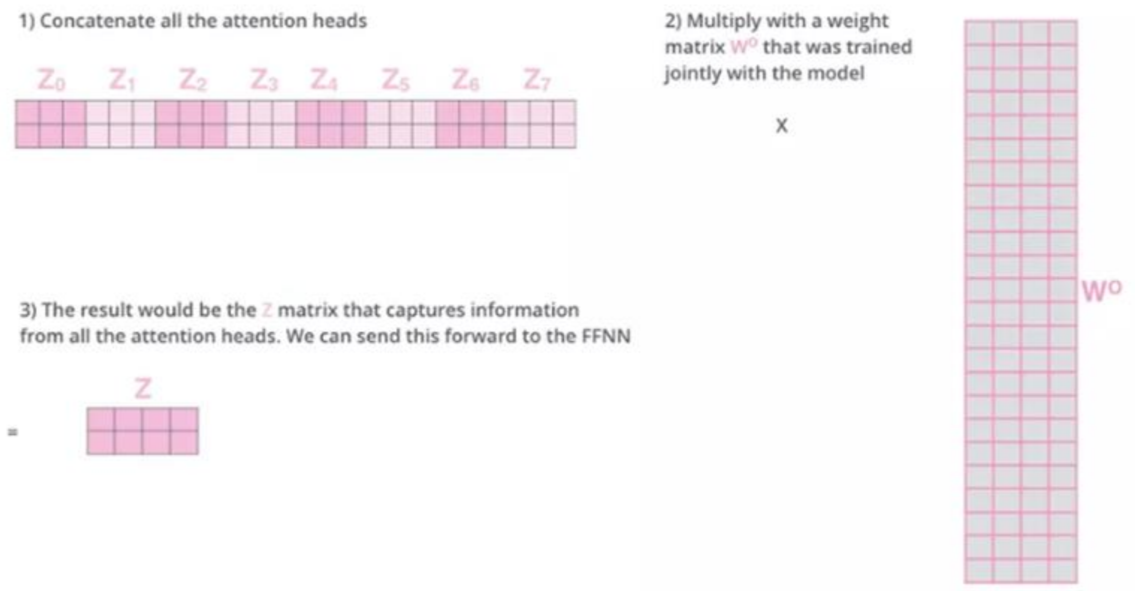
6.2.4. Multi - head attention:

The problem with Self-attention is that a word's attention will always be on itself. This makes sense because it is obvious that "it" must be most related to "it". For example:



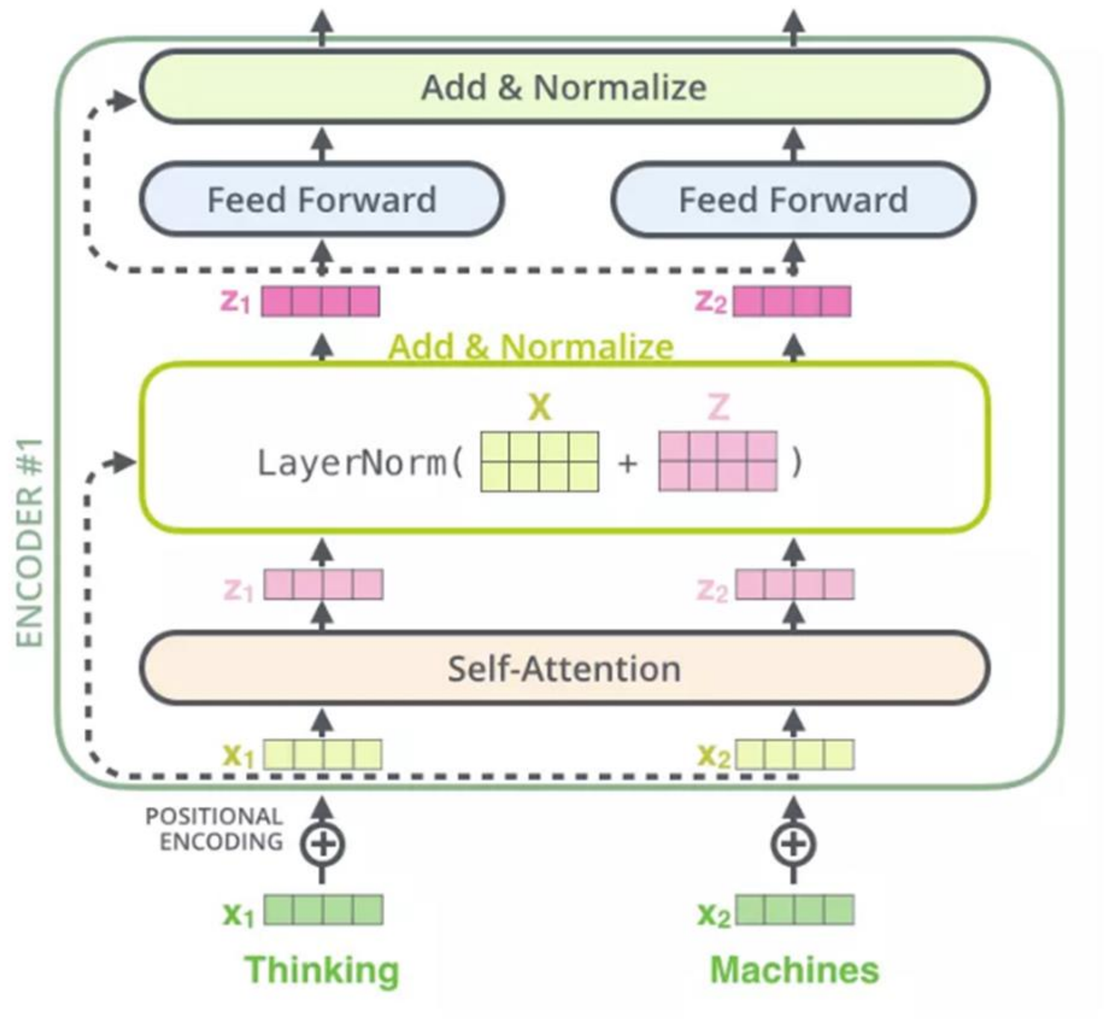
But we don't want this, what we want is the interaction between **DIFFERENT** words in the sentence. The author introduced a more advanced version of Self-attention called Multi-head attention. The idea is very simple: instead of using 1 Self-attention (1 head), we use many different Attentions (Multi-head) and maybe each Attention will pay attention to a different part of the sentence.

Since each "head" will produce a separate attention matrix, we have to concat these matrices and multiply them with the Wo weight matrix to produce a single attention matrix (weighted sum). And of course, this weight matrix is also tuned during training.


1) Concatenate all the attention heads
Z_0 Z_1 Z_2 Z_3 Z_4 Z_5 Z_6 Z_7
2) Multiply with a weight matrix W^O that was trained jointly with the model
x
W^O
3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN
Z
=

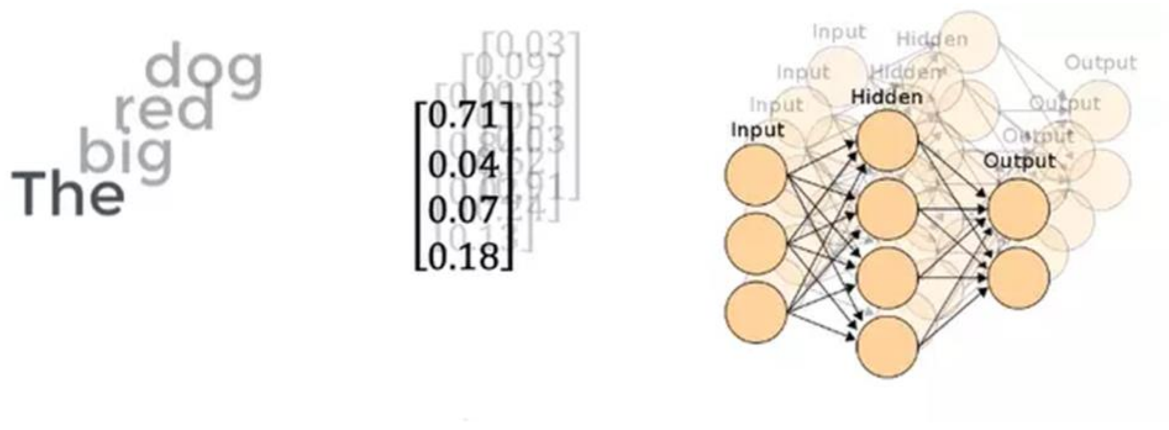
6.2.5. Residuals:

As can be seen in the overview model above, each sub-layer is a residual block. Similar to residual blocks in Computer Vision, skip connections in Transformers allow information to pass directly through the sublayer. This information **(x)** is added to its attention **(z)** and Layer normalization is performed.



6.2.6. Feed forward:

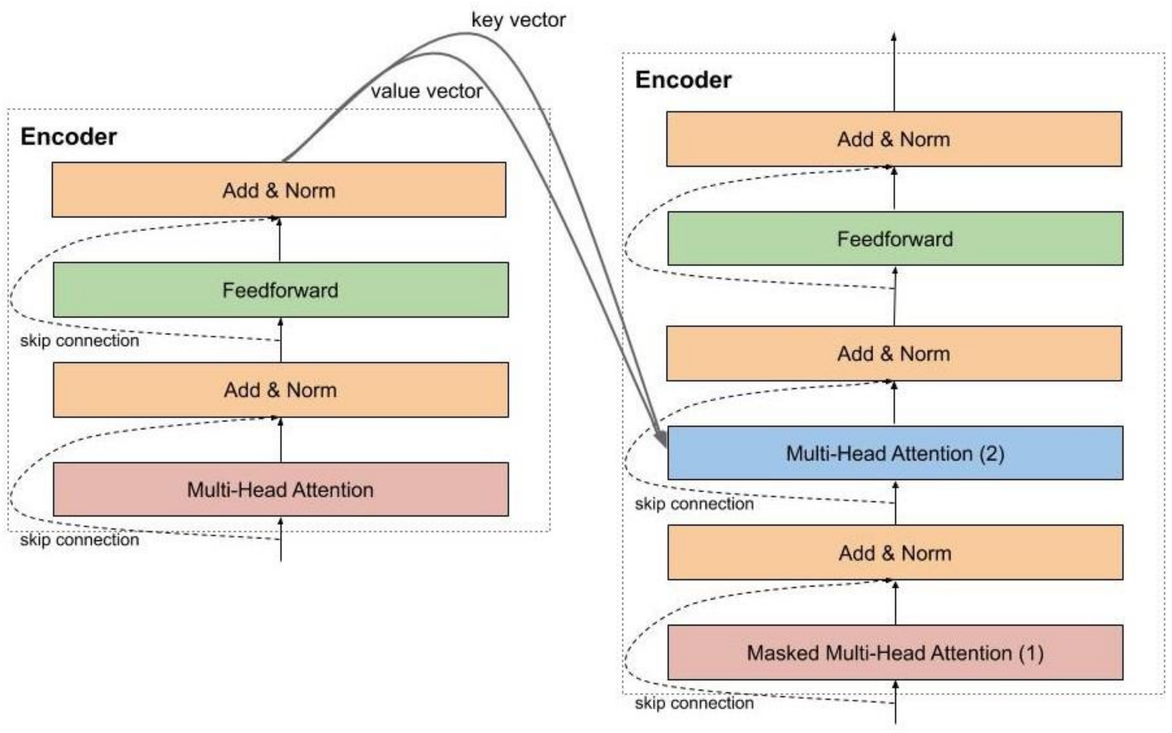
After being Normalized, the **Z vectors** are passed through the fully connected network before being pushed to the Decoder. Since these vectors do not depend on each other, we can take advantage of parallel computation for the entire sentence.



7. Decoder:

7.1. Decoder:

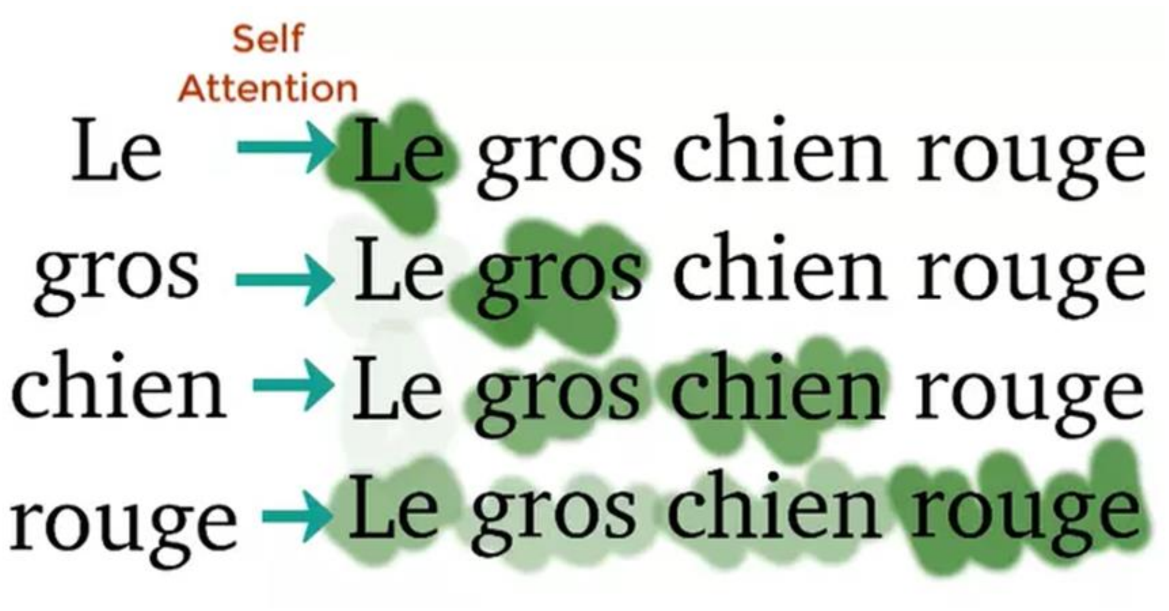
The Decoder performs the function of decoding the vector of the source sentence into the sentence to be translated, so the Decoder will receive information from the Encoder which are 2 vectors of key and value. The architecture of the Decoder is very similar to the Encoder, except that there is an additional Multi-Head Attention in the middle used to learn the relationship between the word being translated and the words in the source sentence.



7.2. Example and Decode process:

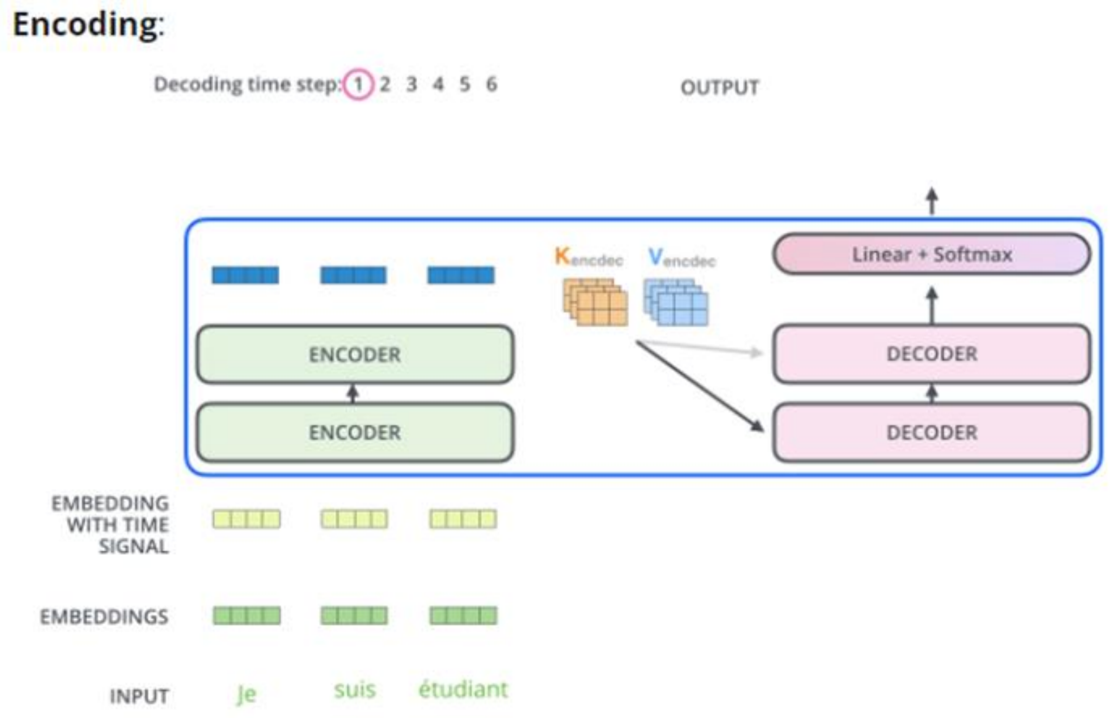
7.2.1. Masked Multi-head Attention

Suppose we want Transformers to perform English-France translation, then the Decoder's job is to decode the information from the Encoder and generate each French word based on the **PREVIOUS WORDS**. So, if we use Multi-Head Attention on the whole sentence like in the Encoder, the Decoder will "see" the next word it needs to translate. To prevent that, when the Decoder translates to the i-th word, the following part of the French sentence will be **masked** and the Decoder is only allowed to "see" the part it has translated before, to predict the next word.

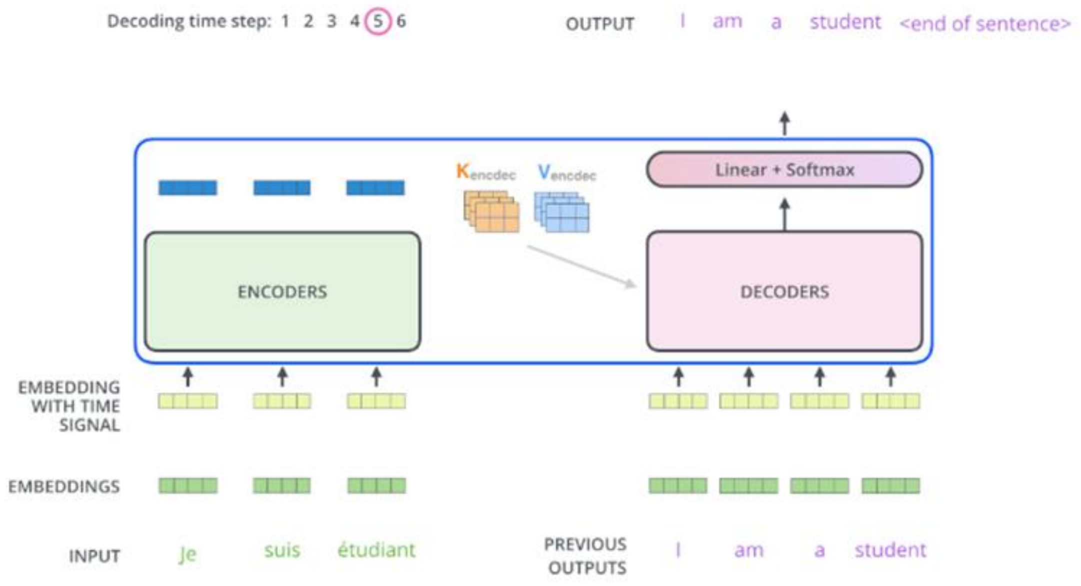

Self Attention
Le → Le gros chien rouge
gros → Le gros chien rouge
chien → Le gros chien rouge
rouge → Le gros chien rouge

7.2.2. Decode process

The Decode process is basically the same as Encode, except that the Decoder decodes each word and the Decoder's Input (French sentence) is **Masked**. After the Masked Input is passed through the Decoder's Sub - layer **#1**, it will not be multiplied by 3 weight matrices to create Q, K, V anymore but only multiplied by 1 weight matrix WQ. K and V are taken from the Encoder along with Q from the Masked Multi - Head Attention and fed into Sub - layers **#2** and **#3** similar to the Encoder. Finally, the vectors are pushed into the Linear layer (a Fully Connected network) followed by Softmax to produce the probability of the next word. The two figures below visually describe the Transformers Encode and Decode process.



Decoding:



Model sizes In Section 3.6 we also showed how scaling up the baseline model size improved performance. However, using smaller models can be helpful in settings where limited computational resources are available for fine-tuning or inference. Based on these factors, we train models with a wide range of sizes:

Base. This is our baseline model, whose hyperparameters are described in Section 3.1.1. It has roughly 220 million parameters.

Small. We consider a smaller model, which scales the baseline down by using dmodel = 512, dff = 2,048, 8-headed attention, and only 6 layers each in the encoder and decoder. This variant has about 60 million parameters

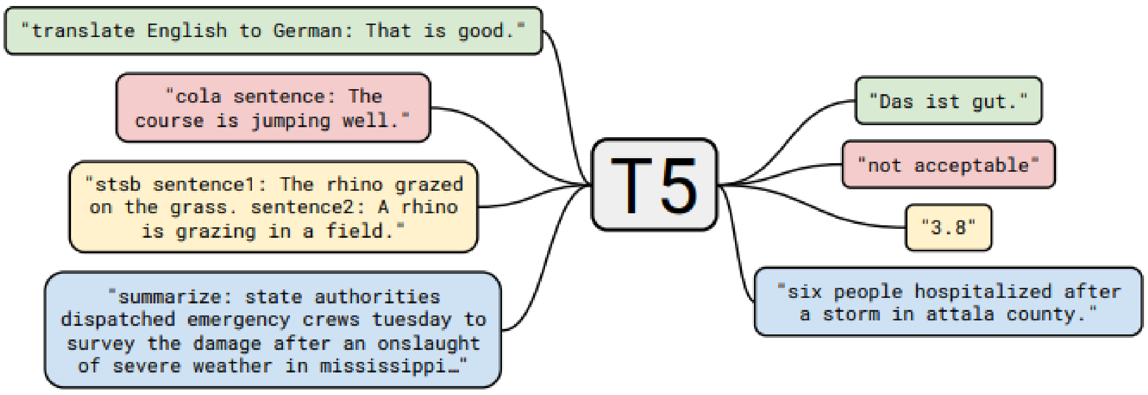
II. T5: Text-To-Text Transfer Transformer

1. T5:

T5 was borned as part of the evolution in transfer learning NLP, where big data learning methods are becoming popular and effective in improving the performance of models on various NLP tasks.

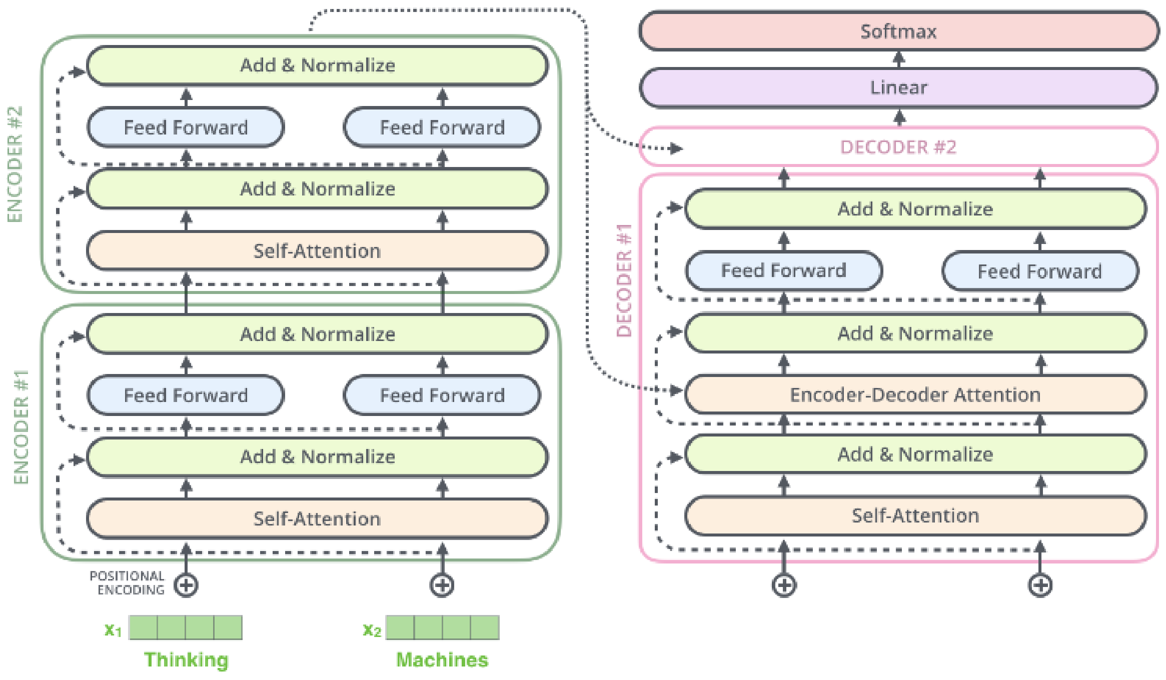
2. Baseline model:

2.1.T5 framework:


"translate English to German: That is good."
"cola sentence: The course is jumping well."
"stsb sentence1: The rhino grazed on the grass. sentence2: A rhino is grazing in a field."
"summarize: state authorities dispatched emergency crews tuesday to survey the damage after an onslaught of severe weather in mississippi…"
T5
"Das ist gut."
"not acceptable"
"3.8"
"six people hospitalized after a storm in attala county."

Many tasks are cast into this framework: machine translation, classification task, regression task (for example, predict how similar two sentences are, the similarity score is in range 1 to 5), other sequence to sequence tasks like document summarization (for example, summarising articles from CNN daily mail corpus).

2.2. Model structure:



The model structure is just a standard sort of vanilla encoder-decoder transformer.

2.3. Pretrained dataset (The Colossal Clean Crawled Corpus (C4)):

Using Common Crawl's web extracted text with some simple heuristic filtering:

_ Retains lines that ended in a terminal punctuation mark.

_ Discards any page with fewer than 5 sentences and only retained lines containing at least 3 words.

_ Removes line with the word javascript and any pages that had a curly bracket.

_ Deduplicate the data set, discards all but one of any three-sentence span occurring more than once in the data set.

After cleaning, the dataset is called “Colossal Clean Crawled Corpus” (C4), with about 750 GB of clean English text data. C4 is used to train the T5 model.

2.4. Downstream tasks:

The goal is to measure the T5 model's general language learning ability across diverse downstream tasks. With those:

_ GLUE and SuperGLUE: Text classification tasks to test general language understanding.

_ CNN/Daily Mail: Text summarization task.

_ SQuAD: Question answering task.

_ WMT: Machine translation task from English to German, French, and Romanian.

_ Model Fine-tuning: The model is fine-tuned on each task by combining all datasets into a single task.

Model Fine-Tuning: The model is fine-tuned on a task-by-task basis by pooling all datasets into a single task.

2.5. Input and Output format:

In order to train a single model on the diverse set of tasks described above, paper mentions that they cast all of the tasks we consider into a “text-to-text” format—that is, a task where the model is fed some text for context or conditioning and is then asked to produce some output text. This framework provides a consistent training objective both for pre-training and fine-tuning.

_ For example, for machine translation, the input is “translate English to German: That is good.” and the output is “Das ist gut.”

_ Task Prefix: Add a task-specific prefix to the input string to specify the task to be performed.

Text Classification Task: The output is a single word representing the target label.

Unwanted Output Problem: If the model produces output that does not match any of the possible labels, the result is counted as false.

STS-B Task Refinement: Convert the STS-B regression task into a 21-class classification problem by rounding and mapping similarity scores to text strings.

Winograd Task: Transform the ambiguous pronoun problem into a “predicting the representative noun” problem, by highlighting the pronoun and asking the model to predict the representative noun for that pronoun.

3. T5-Vietnamse-Nom model:

Vietnamese Nôm, or Chữ Nôm, was an ancient writing system in Vietnam before the 20th century. It evolved from Chinese characters but adapted to Vietnamese sounds and vocabulary. Nôm was used by scholars for literature and communication. The script visually differed from Chinese characters and expressed Vietnamese concepts with semantic and phonetic components. Today, Nôm is a specialized field, and efforts are made to preserve its knowledge. Though modern Vietnamese uses the Latin alphabet, Nôm remains an integral part of Vietnam's cultural heritage.

State-of-the-art lightweights pretrained Transformer-based encoder-decoder model for Vietnamese Nom translation.

Model trained on dataset Luc-Van- Tien’s book, Tale Of Kieu book, “History of Greater Vietnam” book, “Chinh Phu Ngam Khuc” poems, “Ho Xuan Huong” poems, Corpus documents from chunom.org, sample texts coming from 130 different books (Tu-Dien-Chu-Nom-Dan Giai).

The model is trained and supports bidirectional translation between Vietnamese Nôm script and Vietnamese Latin script, enabling the translation from Nôm to Vietnamese Latin script and vice versa.

III. Fine-tuning & Evaluation results

1. Dataset: NetFlix-200k and tqdn

The NetFlix-200k and tqdn dataset is a dataset for machine translation use cases. It consists of 226361 rows of sentences in Chinese and Vietnamese. There are no duplicate rows within the dataset.

There are 2 features:

- **source:** a string containing Chinese.

- **target:** a string containing Vietnamese.

The dataset has already been split into 3 splits train-test-validation at a ratio of about 0.8-0.1-0.1.

2. Finetuning process

The model is finetuned on the train split of the dataset (181088 rows) using HuggingFace’s *transformers Trainer* instance. Some notable training parameters are:

- Learning rate = 2e-5

- Batch size = 32

- Eval batch size = 8

- Weight decay = 0.01

- Train epochs = 3 (Due to the huge size of the dataset, to save resources we can only finetune on a small epoch size)

3. Evaluation results

In natural language processing, model performance is evaluated using various metrics, each with its unique strengths. **BLEU**, a widely used metric, measures n-gram overlap between generated and reference texts, while **SacreBLEU** standardizes BLEU for consistent benchmarking. **ROUGE** emphasizes recall, ideal for summarization, while **METEOR** balances precision, recall, and semantic matching for greater linguistic flexibility.

TER assesses translation effort by calculating the minimum edits needed, offering interpretability, and **ChrF**, focusing on character-level n-grams, excels in capturing nuances in morphologically complex languages. Together, these metrics provide a comprehensive framework for evaluating text generation tasks.