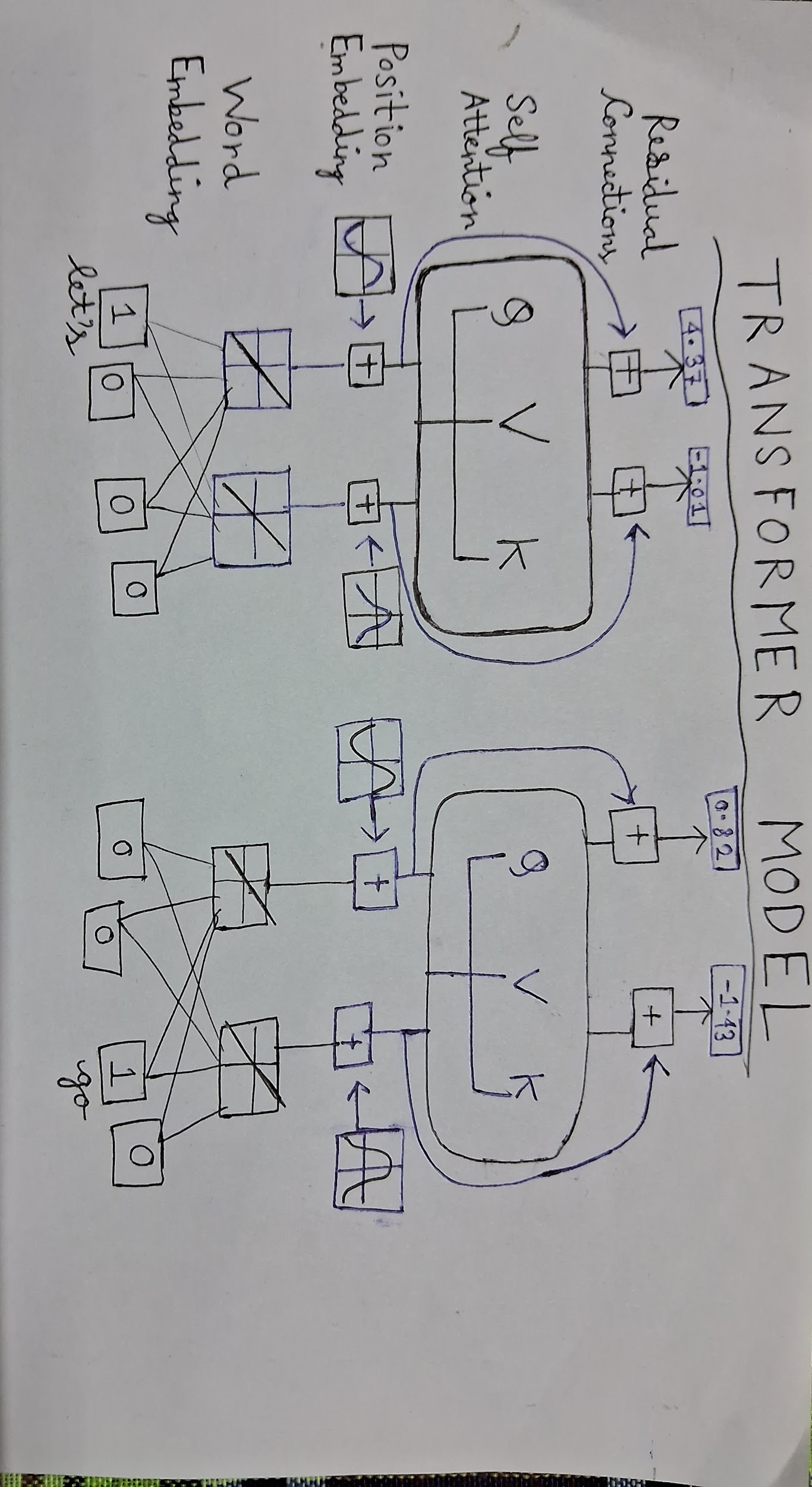
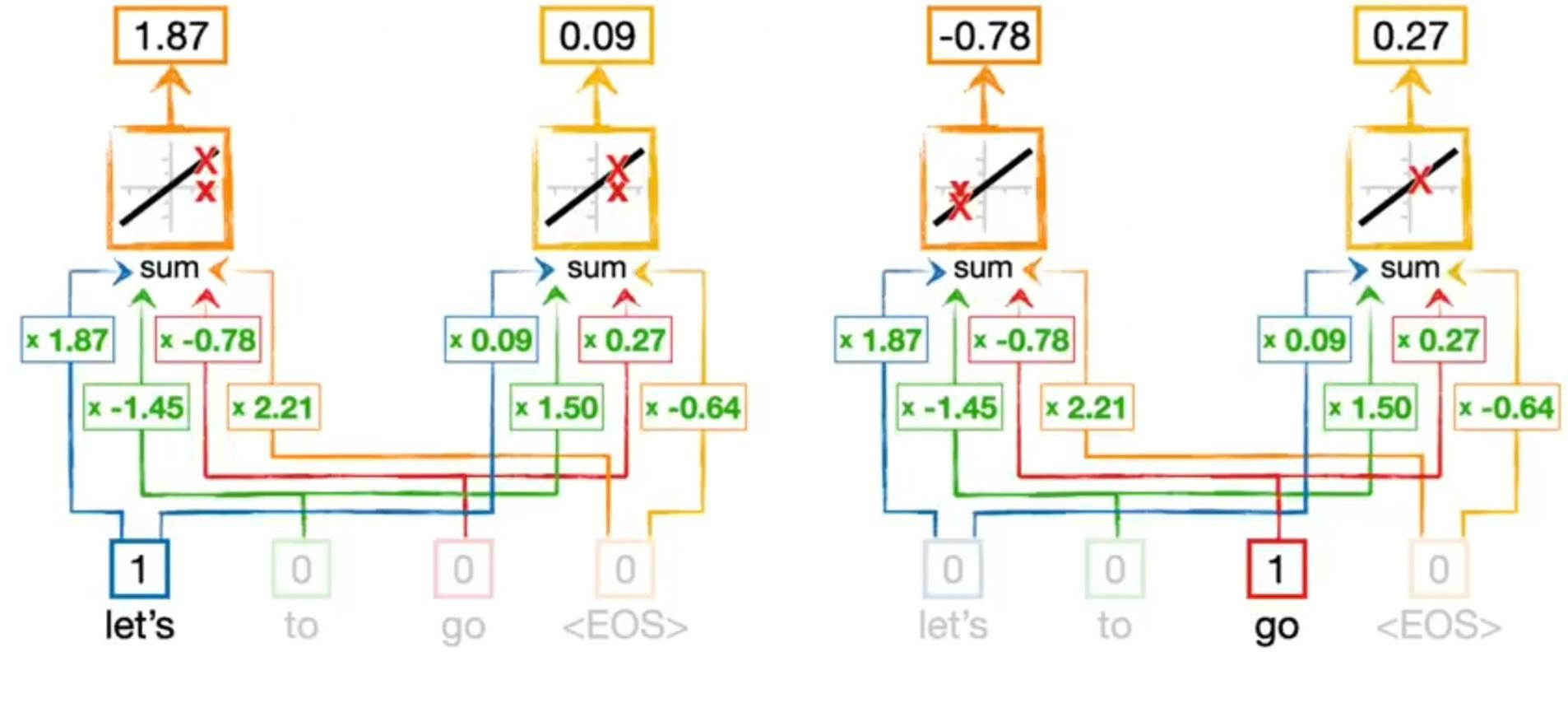
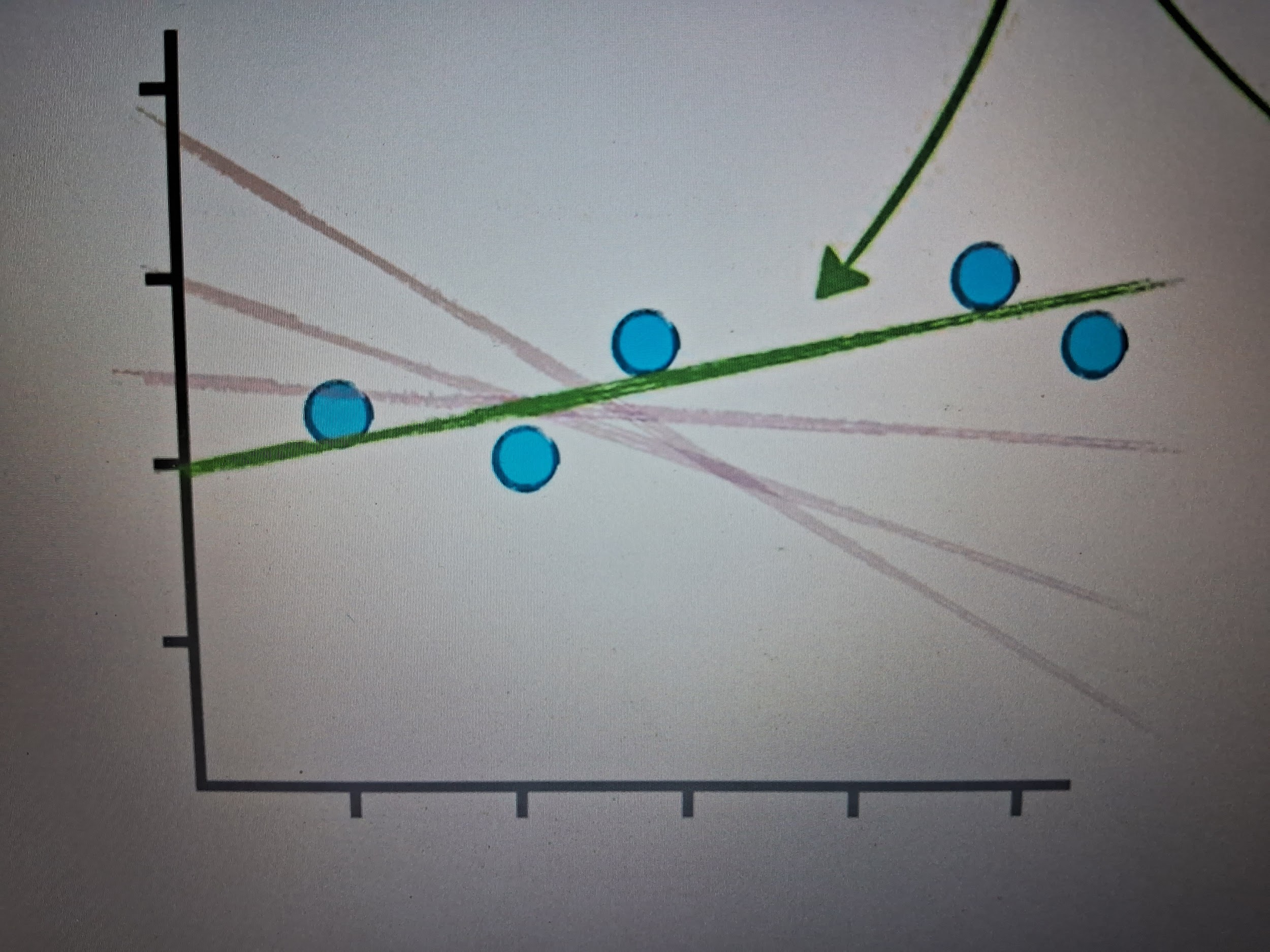
Transformers are a class of deep learning models that have gained significant popularity in natural language processing tasks.The Transformer model is core of chatgpt's architecture. It was introduced in the paper "Attention Is All You Need" by Vaswani in 2017.Basic structure of this model is —



It is a neutral network model.Specifically one application is Transformer can translate a simple english sentence to spanish sentence or any other language. Ex : **Let's** **go** to **ir Vamos**.We have to find the way to turn words into numbers.

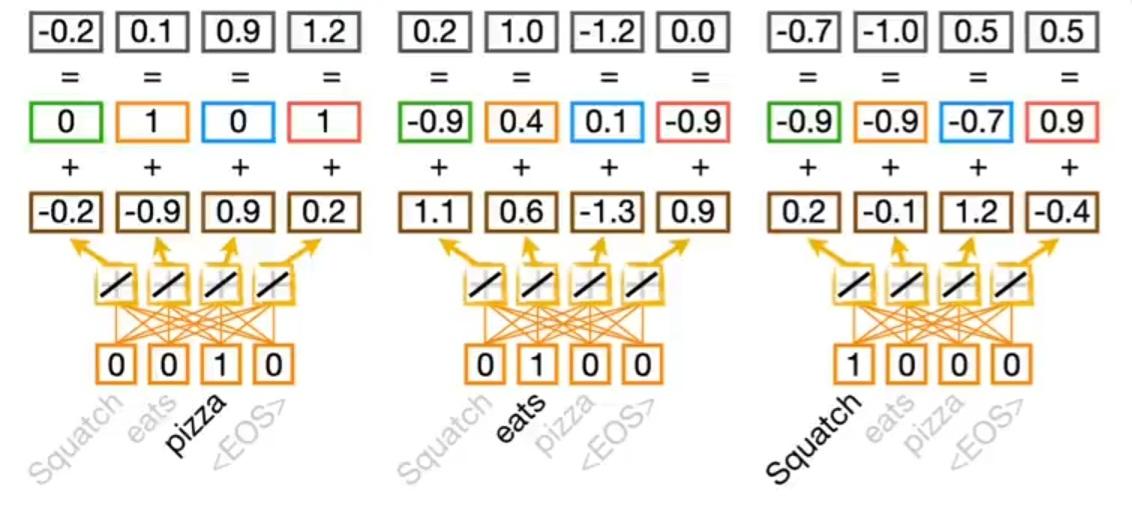
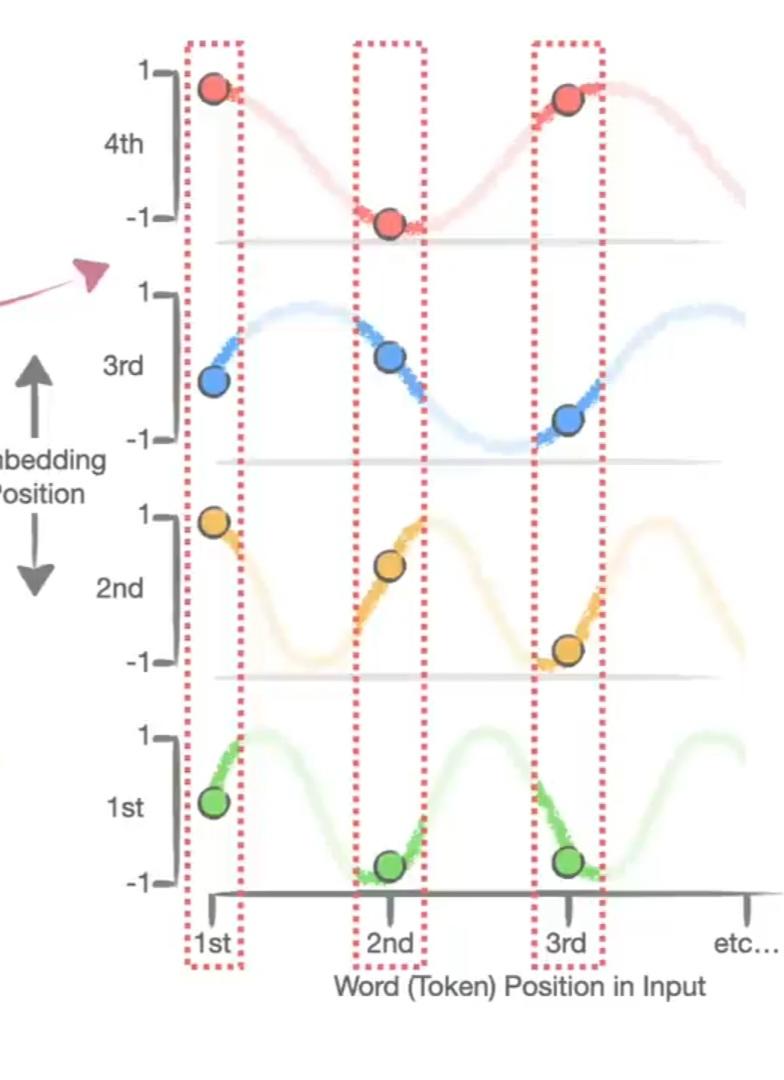
1)WORD EMBEDDING : The commonly used method is ***Word Embedding.***There are some vocabulary inputs like let's ,go,to,<EOS> which stands for END OF SENTENCE.There are some activation functions and weights of each input.First we put the numbers in weights ,then multiply with weights connecting to the activation functions.

Here 1 is input for let's in the left side.1 is multiplied by weight 1.87 and weight 0.09 to to produce the activation function graphs f(x) = x . So the output numbers for let's are 1.87 and 0.09.Similarly,on the right side,go is transformed into numbers -0.78 and 0.27.

* The weights in the network for let's are exactly the same for the network of go.This gives the **flexibility** to handle input sentences with different lengths.
* All the weights are determined by **BACKPROPAGATION**.It is a function like *y=mx+c .*It takes random values for m and c and changes until it found the optimum values.

The process of optimising the weights is also called **Training.**

2) POSITIONAL EMBEDDING : Word order also matters.Ex : Squatch eats Pizza .At first ,we find the set of numbers by Word Embedding.Then we add a new set of numbers which correspond to word order to the embedding values for each word.The numbers for word order comes from **sine** and **cosine** squiggles.The y axis coordinates of green squiggle gives the first numbers for all three words Squatch,eats,Pizza .Similary, yellow,green and Red squiggles give the 2nd,3rd and 4th word positioning numbers for all three input words .

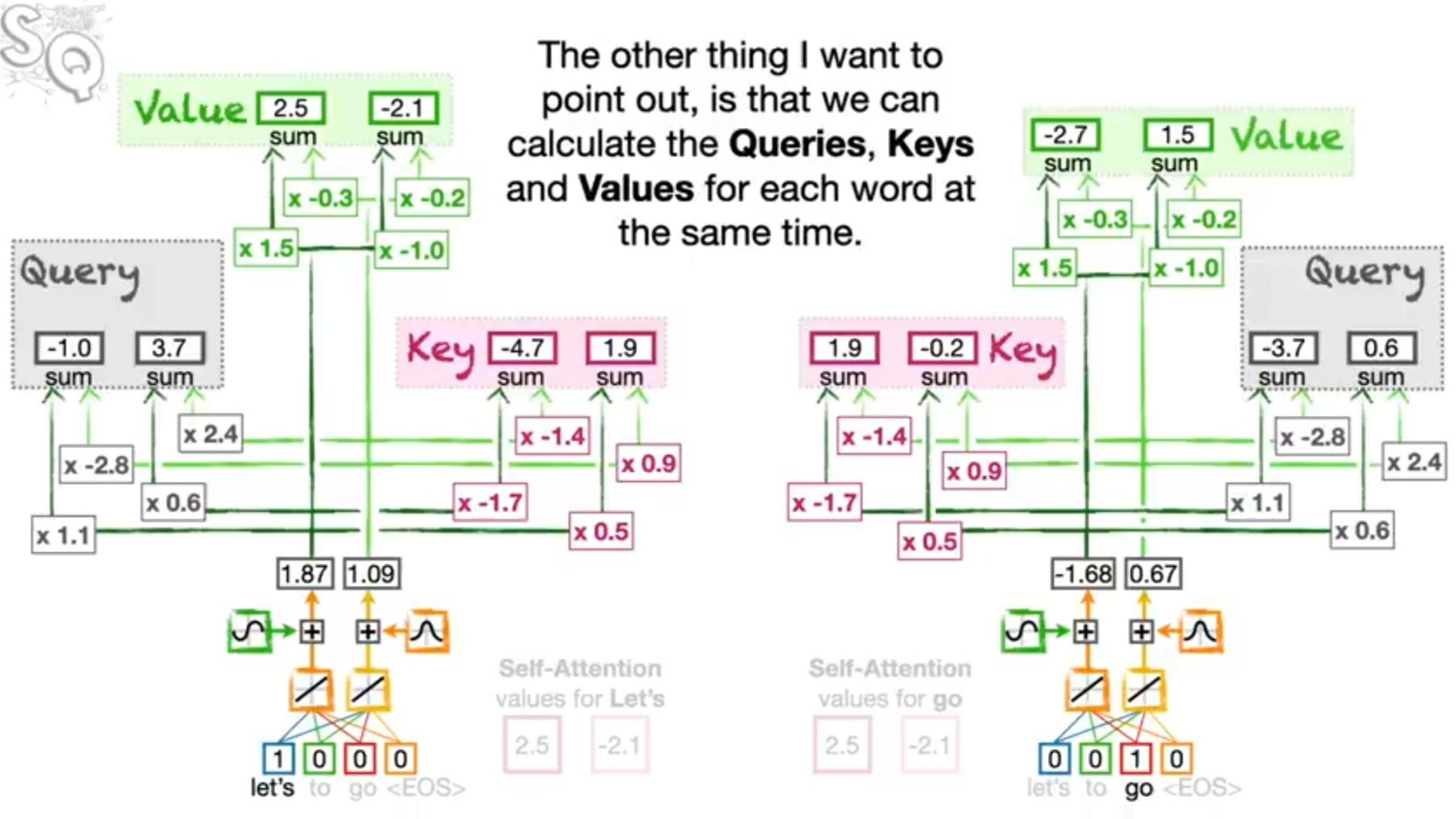
*Add the positional values to embedding values.Thus positional embedding is done.*

3) Self Attention :How does it work ? For example,input is given as Pizza comes out from oven and it tastes good.It will detect that 'it' is associated with pizza,not oven. Let's do this with the example "let's go ".We have to multiply the position encoded values for the word Let's by a pair of weights and those products to get -1.0 .Then we do the same thing with different weights to get 3.7. These two new values are **QUERY VALUES.**Thus with some different weights , **KEY VALUES** are obtained for 'Let's' and also for 'go'.From sone calculations on the key values ,we can see that **Let's** is much more similar to itself than it is to the word **go**..**Let's** have much more influence in encoding than **go.**

* And we do this by running similarity scores by something called a **SOFTMAX** function.
* Output of **Softmax** function is used to determine what percentage of each input word we should use to determine what percentage of each input wird should we use to encode the word **Let's.** We use 100 percent of the word **Let's** and 0 percent of **go** to encode **Let's .**

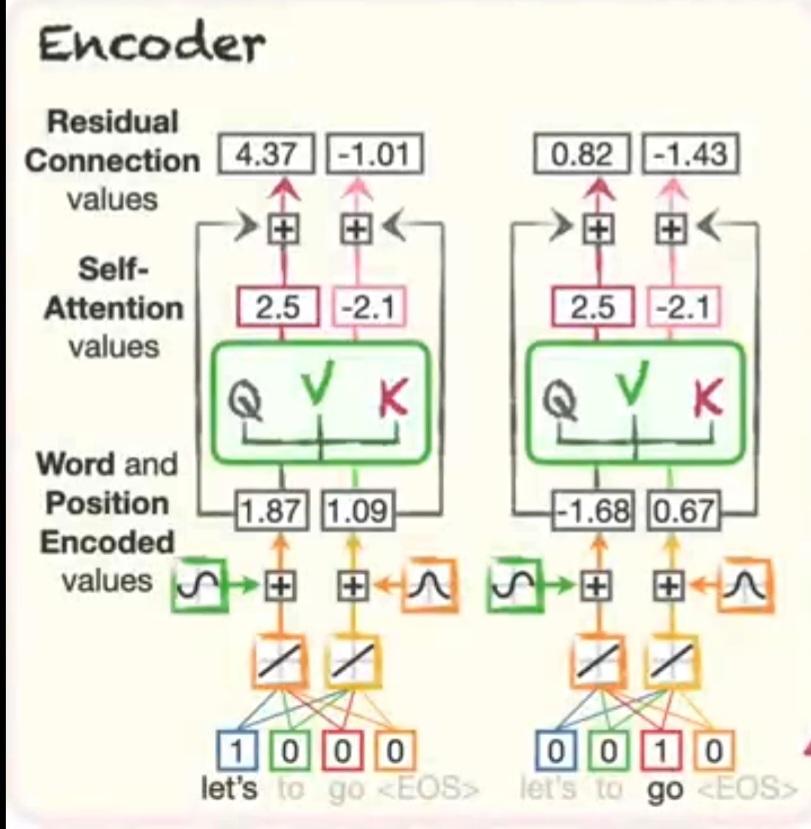
Lastly we get two more values which we call **VALUES.**Scale the values to represent **Let's** by **1.0** and we create two **VALUES** for **go** and scale those to reprent **go** by **0.0.**We add the scale values together. Thus from the sum,**Let's** are SELF - ATTENTION values for **Let's.**

Similary , we get SELF - ATTENTION values for **go.**

* Same sets of **weights** are used for **Key,Query,Values** for both **Let's** and **go.**
* Because we can do all of the computation at the same time,**Transformers** can take advantage of parallel computing and run fast.

4) Now, SELF - ATTENTION values are added with WORD and POSITION ENCODED values to get **Residual Connection** values.

Now , we have encoded the input for simple transformer in these 4 steps.



TRANSFORMER can do all of the computations for different input words at the same time,rather than doing them sequentially for each other,means we can process a lot of words relatively quickly on a chip with a lot of computing cores,like a **GPU** ( graphics processing unit), or on multiple chips on the cloud.

Now,we need to decode it.

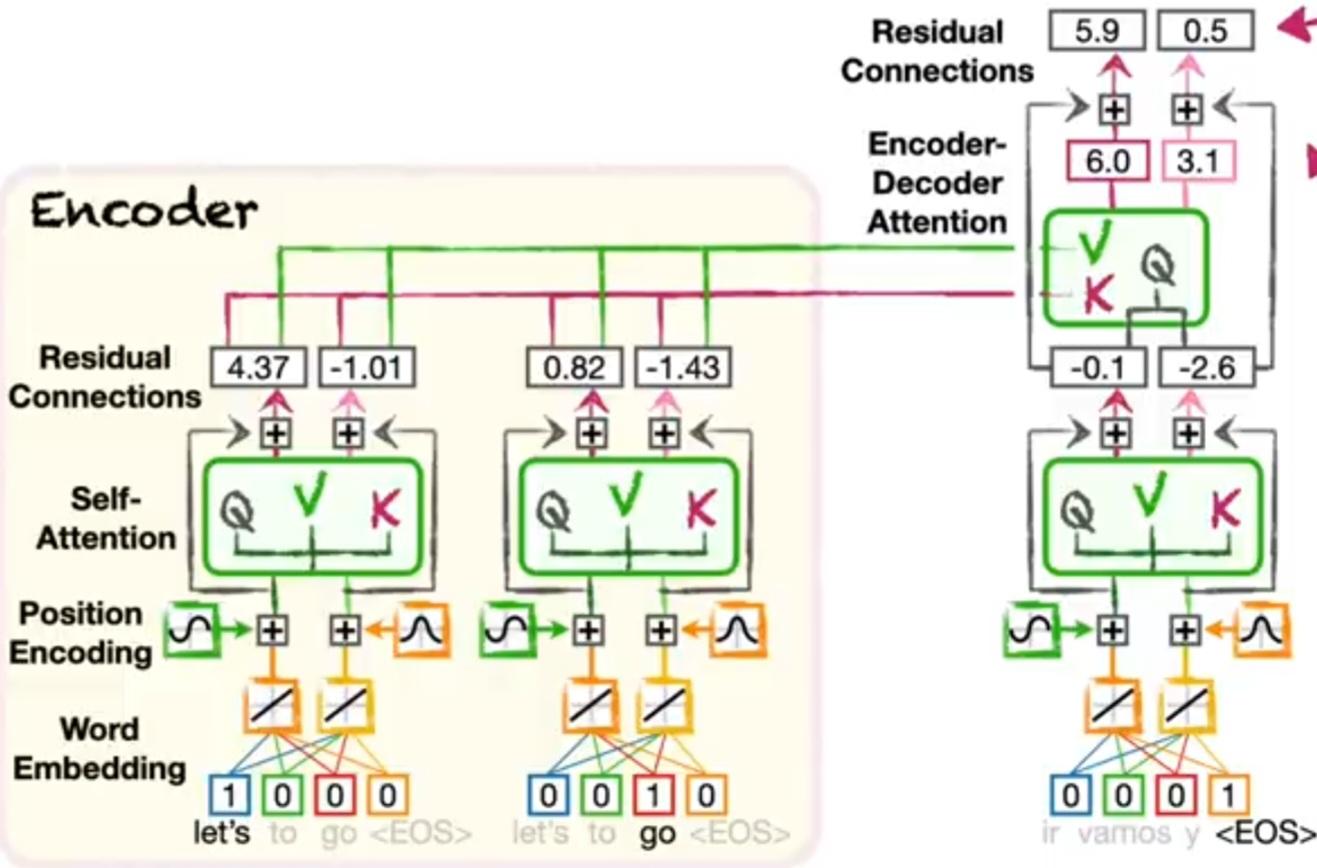
1)WORD EMBEDDING :

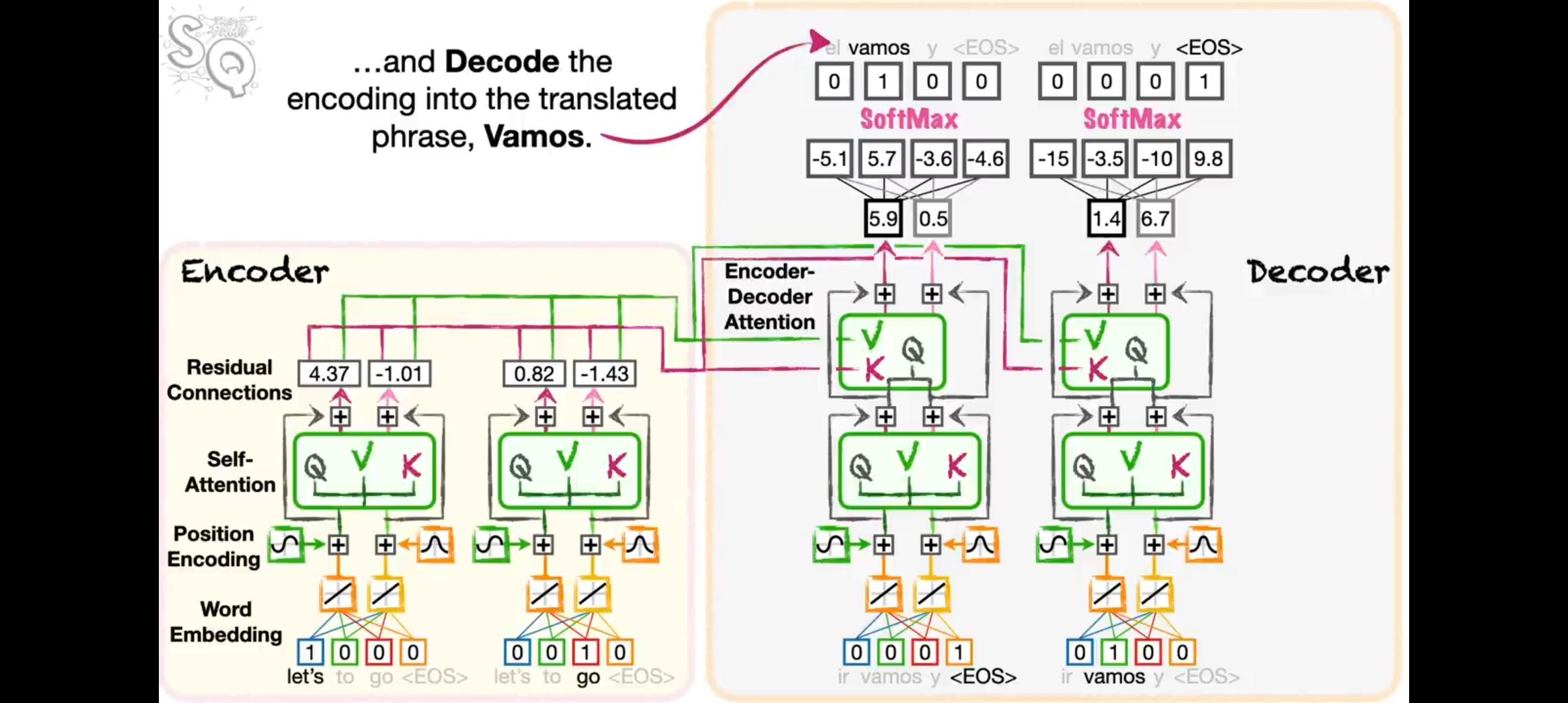
2)POSITIONAL EMBEDDING : The translated answer will be **ir Vamos.**So here we are taking input words as **ir,Vamos,y,<EOS>.**Only <EOS> is tajen as 1 and other inputs as 0.We get word embedding values as 2.70 and -1.34.Here the graphs are exactly same as the graphs we took while encoding.Thus by the same way ,we get the positional embedding values.

3) Self Attention: Query ,Key,Values are obtained . But , the sets of **weights** we used to calculate the **Decoder's Self - Attention Query,Key and Value** are different from the sets we used in **Encoder.**

The main idea of Encoder - Decoder Attention is to allow the **Decoder** To keep the track if **significant words** in the input.

Just like Self - Attention , we create 2 new values to represent the **Query** for the <EOS> token in the Decoder.Then we create **Keys** for each word in the Encoder.And we calculate the similarities by calculating the **Dot Product.**By **SoftMax ,**we calculate **Values** and scale them and then add the pairs of scaled values together to get the **Encoder - Decoder Attention** values.Sets of **weights** are different in **Encoder - Decoder Attention** than **Self - Attention .** But we can stack to keep track of words in complicated phrases .

Now we add another set of **Residual Connections** that allow the **Encoder - Decoder Attention** to focus on relationships between the output words and the input without having to preserve the **Self - Attention** or **Word** and **Position Encoding** that happened earlier.

Lastly, we run the values that represent **vamos** through the same **Fully Connected Layer** and **Softmax** that we used for the **vamos** token and **<EOS>** token.

Here are some key aspects of the Transformer model:

Attention Mechanism: The Transformer relies on a self-attention mechanism that allows it to weigh the importance of different parts of the input data when making predictions. This is crucial for understanding the context in sequences, such as language.

Multi-Head Attention: Transformers use multiple attention heads to capture different aspects of the input data simultaneously, enhancing their ability to learn complex relationships in the data.

Positional Encoding: Transformers don't have built-in sequential understanding, so positional encodings are added to the input data to provide information about the positions of tokens in the sequence.

Stacked Layers: Transformers consist of multiple layers (e.g., encoder and decoder layers) stacked on top of each other. Each layer refines the representation of the data.

Feedforward Neural Networks: Within each layer, there are feedforward neural networks that process the output of the attention mechanism.

Layer Normalization: Layer normalization is used to stabilize the activations within each layer.

Pretraining and Fine-Tuning: Transformers are typically pretrained on large datasets (unsupervised learning) and then fine-tuned on specific tasks (e.g., language translation, text generation) with smaller, task-specific datasets.

Applications: Transformers have been widely adopted in natural language processing (NLP) tasks, including machine translation, text summarization, sentiment analysis, and, of course, chatbot models like ChatGPT.

Some more information —-

* *Original* ***Transformer*** *has* ***37000*** *tokens,and longer input and output phrases.*
* *In reality ,* ***Similarity*** *= Dot product / ( # Embedding values)½*

**Conclusion:** I learned a lot about how a simple Transformer Model works. The references I followed to write this article are –

ChatGPT,YouTube Channel – StatQuest.