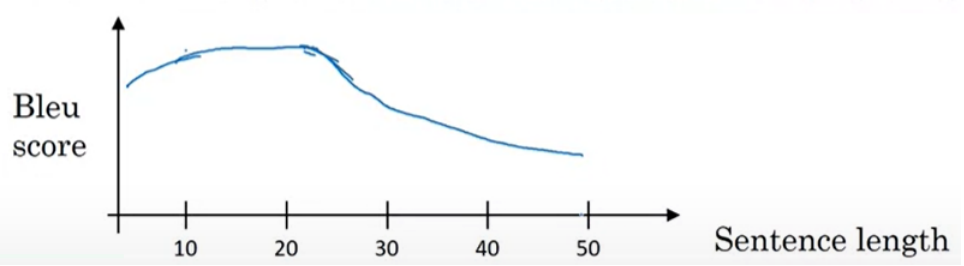
dl.ai_Attention_model

1)Simple attention

a)Why attention is needed?

Before attention models we used to use Encoder decoder recurrent neural networks for machine translation task. But they had an issue , the size of the representation of the sentence we get after passing it to an encoder is fixed. We have a tensor depicting a sentence of 10 words if we use a tensor of the same size to depict a 100 word sentence don't you think the quality of representation decreases. We see a decrease in blue score as sentence length increases.

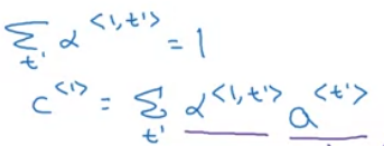


Humans don't read the entire sentence at once and then translate it. We tend to translate the sentence part by part. To accomplish this we use attention models which gives attention to words near a given token to find the a output token.

b)Architecture of Simple attention

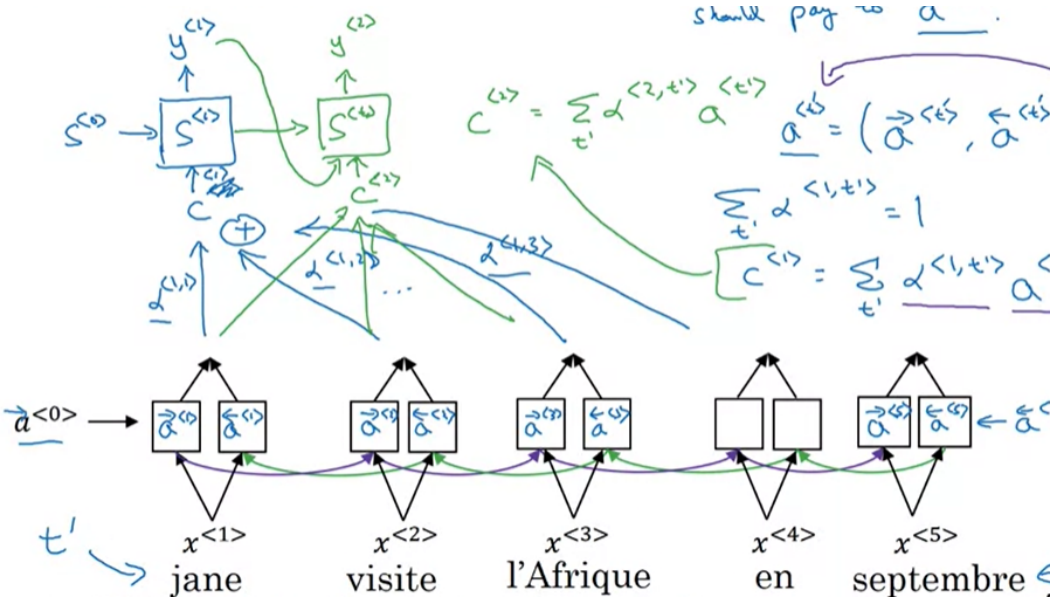
We first use a BRNN to produce the contexts of each input word given by A<t>.A<t> is a concatenation of a→<t> and a←<t>

We take in all the contexts(outputs of brnn) apply a weight $\alpha < i, t >$on them to create a single context vector for each i. $\alpha < i, t >$ denotes the amount of attention y<i> must pay to A<t>



When then pass it to another many to many Rnn with states S which use it as the input and produces the final translation.

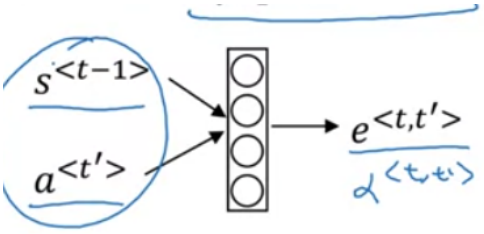
However unlike the weights inside brnn or rnn. The weights $\alpha < i, t >$ changes with each S<i> for each A<t>.We end the recurrence of S once we encounter a <eos> token.



How do we compute $\alpha < i, t >$?We have to apply a softmax on e<i,t> to ensure the sum of the weights is one.

$$\alpha^{<t,t'>} = \frac{\exp(e^{<t,t'>})}{\sum_{t'=1}^{T_x} \exp(e^{<t,t'>})}$$

We know e<i,t> will depend on S<i-1,t> and A<t> as the weights must differ based on the previous word. We have small neural network to get relation between S<i-1,t> and A<t>.



2) Attention is all you need

As we move from RNN to GRU to LSTM the complexity increases but the training time doesn't decrease as all of them ultimately are sequential models. The paper "attention is all you need" proposed a method where we just use attention for text to text transformation without using any sequence to sequence models. This allows the model to be parallelized just like CNNs. We achieve this through two methods called self attention and multi-head attention.

a)Self Attention

For self attention we use different inputs and equation to find attention compared to simple attention. For simple attention equation the input is a mapping e<i,t> of (S<i-1,t> and A<t>.For self attention equation input are three states Query(Q),Key(K) and Value(V) which are a mapping of x<t>.

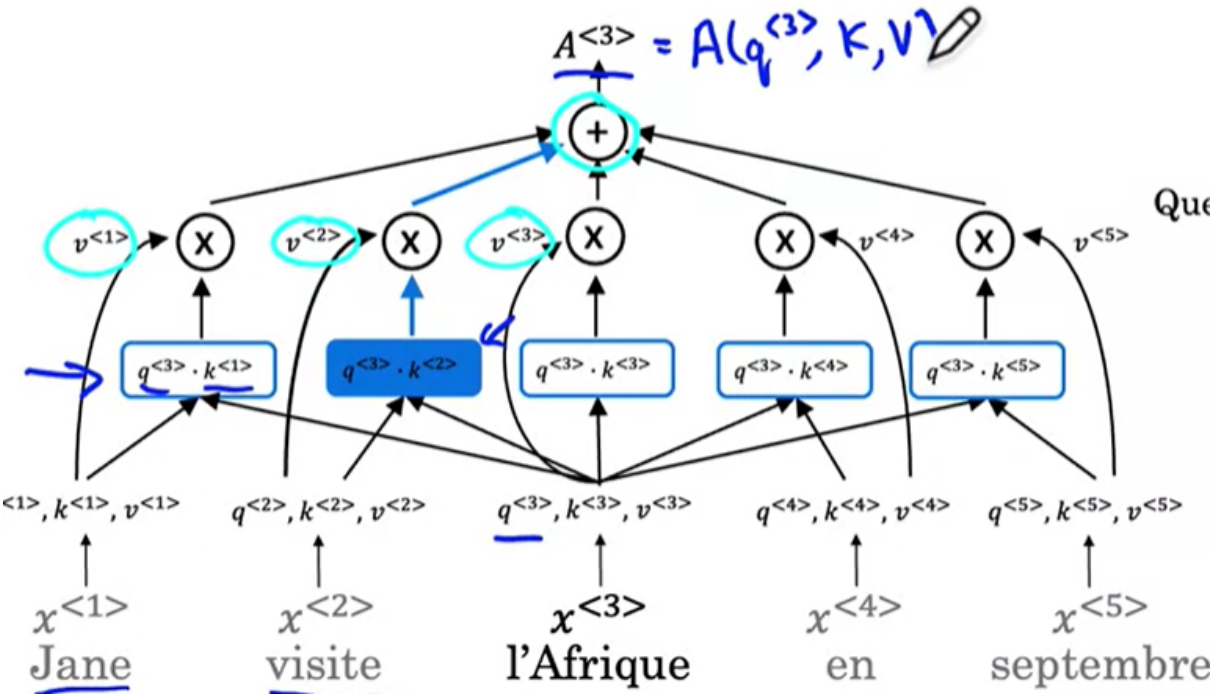
$$A(q,K,V)=\sum_i \frac{\exp(q\cdot k^{<i>})}{\sum_j \exp(q\cdot k^{<j>})} v^{<i>}$$

For each x<t> we have a q, K and V. Query is a like a question which corresponds to x<t> example for the word africa "What is happening in africa",Key is like the answer to that question.

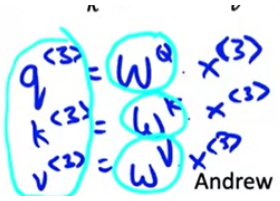
For the first word we take its query q<1> we take the dot product of q<1> with keys of each x<t>.If value of the dot product is high it means that the key answers the query. This way the machine is able to learn the relation between words.

We take the soft-max of the dot products which will give us i different probabilities(valued between 0 and 1).If the value of dot product was high we would get a probability closer to 1.If the dot product is low the probability would be closer to 0.This way the query and key act together like a gate with Value being the candidate.

Finally we multiply these probabilities with the corresponding value V<t>.The final tensor that we get is called as self attention of x<1> it is specific for x<1> as we only used the query q<1> but it also includes the information of the other words through the keys of each word.

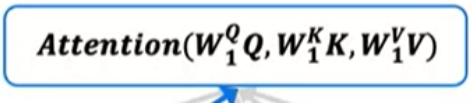


We get Query Key and Value from x<t> by multiplying it with a weight each. The weights are shared across each transformer unit and each unit acts independently hence we can parallelize it.

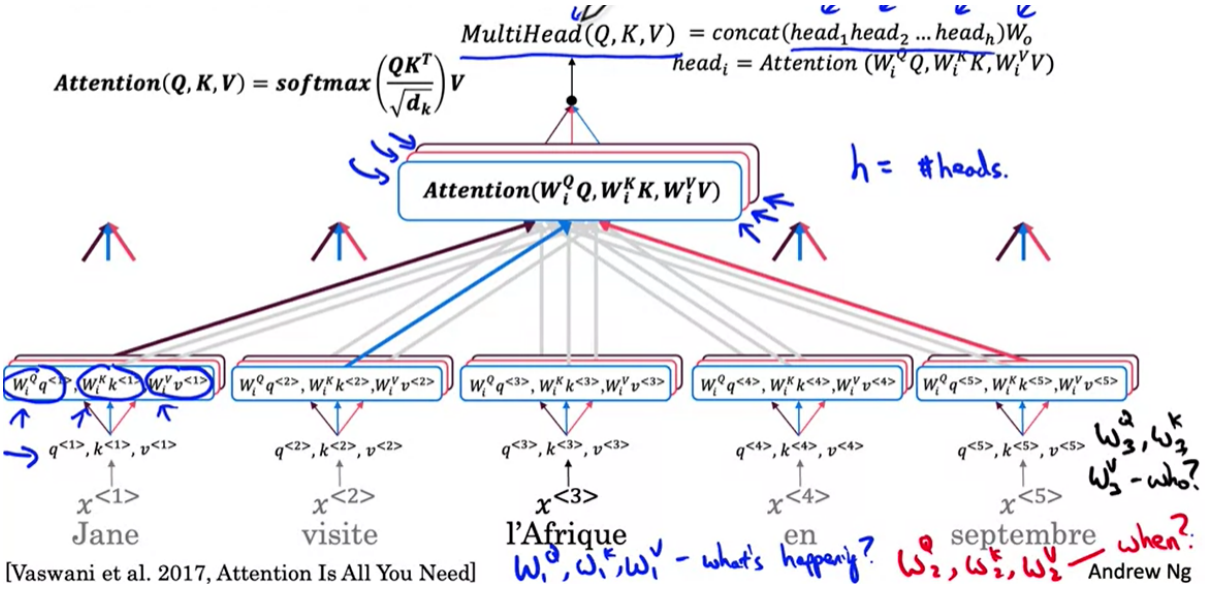


b)Multi-head Attention

Since we are reusing the same Wq we will be asking the same query throughout but what if we want to extract more features we would need to ask more questions. In multi-head attention instead of passing the same q to attention each time we pass different value of q by multiplying x<t> with a different weight W_i^q for each head. We similarly change K and V. We pass these three new values to the attention equation.



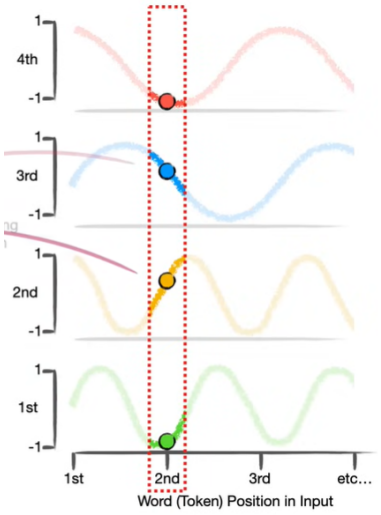
Heads are similar to layers in CNN. We can parallelize the operations for each head> after we get the attention for each head we simply concatenate the attention values and multiply it with one more weight W_o to get our multi head attention.

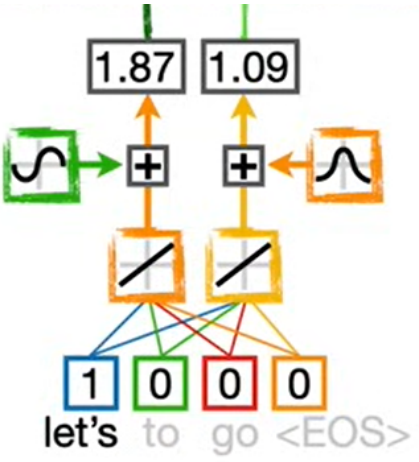


The diagram is wrong we don't multiply q<1> twice before passing it to attention equation

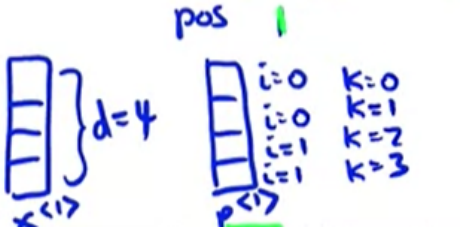
c)Basic Transformer

Transformer pipeline for encoder consists of Word embedding → Positional encoding → Self/Multi-head Attention → Residual connection. If you pay "attention" to our previous self attention model the position of the word may not captured properly by the attention values. Hence we need a way to incorporate the position of the word into its self attention values.





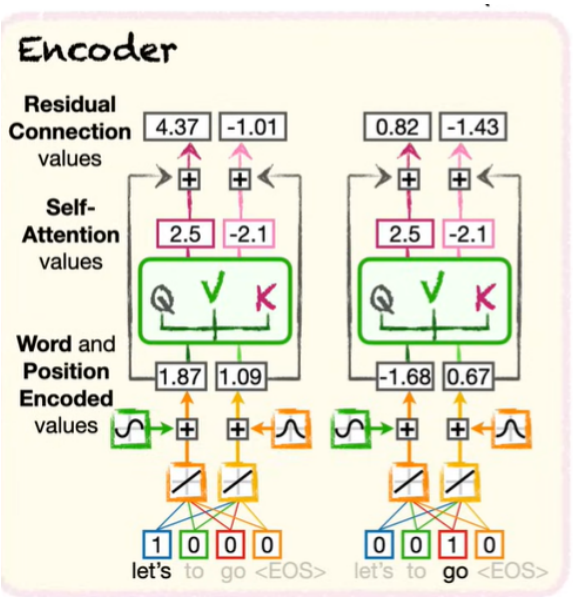
Let say our word embedding has a length of 4 then we will use 4 different sin cos functions to add a positional encoding to our embedding. We would simply take the value of the 4 functions for corresponding value in x axis and add these values to the word embedding. Since each of these function is systematically different that is the omega of the function gradually decreases the positional encoding is unique. For a actual embedding having 300 dimension the omega is very low hence two adjacent values can't be equal.



$$PE_{(pos,2i)} = sin(\frac{pos}{10000^{\frac{2i}{d}}})$$

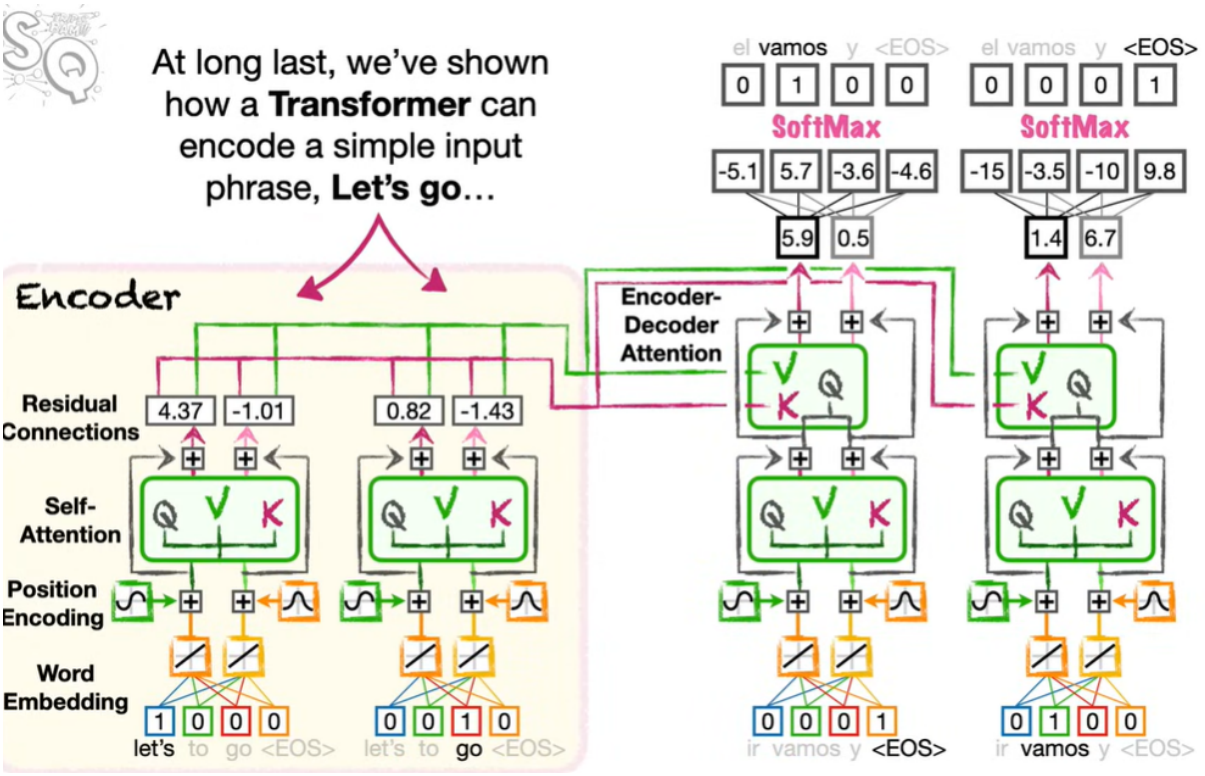
$$PE_{(pos,2i+1)} = cos(\frac{pos}{10000^{\frac{2i}{d}}})$$

After our self attention or multi-head attention we can even add a residual connection or normalization



According to Andrew ng we even pass the attention values through a feed forward neural network and repat the process WE,PE,SA,RC FFNN for 6 times.

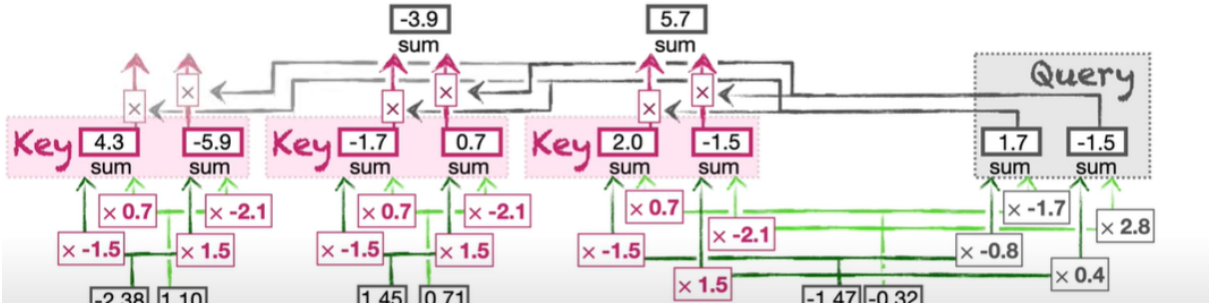
For the decoder we start with <eos> token as input into the WE,PE,SA and RC to get decoder attention values(mine). We find our current query using this decoder attention values. We find key and value for each word in the encoder using the Residual connection values. The attention values we get using the above three inputs is called the encoder decoder attention. The queries are asked by the decoder, the key and the values are given by the encoder this allows our model to learn relation between words in the prompt and response. We finally pass this EDA values to a fully connected layer and soft-max to predict the next word. We pass the prediction we get as the input for the next unit(decoding is sequential)unless our prediction is a <eos> token.

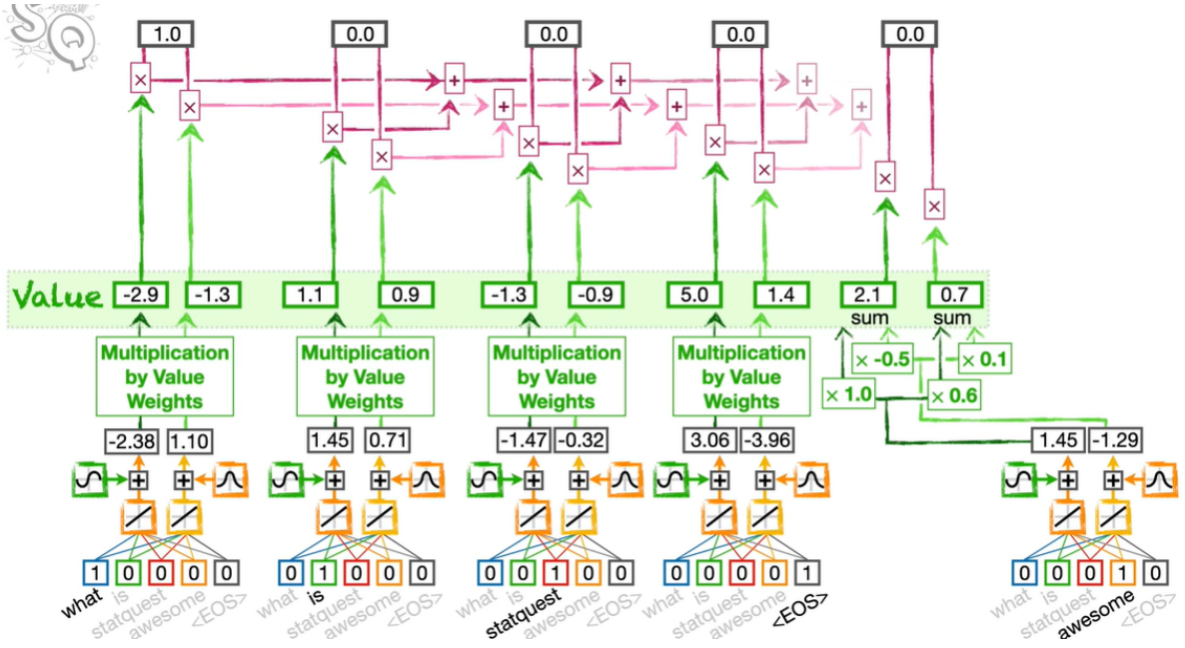


d)Decoder only Transformer

We can see that in basic transformers we need to use two different units for encoder and decoder. Chatgpt uses a decoder only transformer with each unit being alike. Chatgpt further uses a Masked self attention instead of regular self attention.

In masked attention when we take keys to consider for soft-max we only take the keys from the preceding words.




1.0
0.0
0.0
0.0
0.0
Value
-2.9
-1.3
1.1
0.9
-1.3
-0.9
5.0
1.4
2.1
0.7
sum
sum
Multiplication by Value Weights
Multiplication by Value Weights
Multiplication by Value Weights
Multiplication by Value Weights
× -0.5
× 0.1
× 1.0
× 0.6
-2.38
1.10
1.45
0.71
-1.47
-0.32
3.06
-3.96
1.45
-1.29
what
is
statquest
awesome
<EOS>