Introduction to Transformers

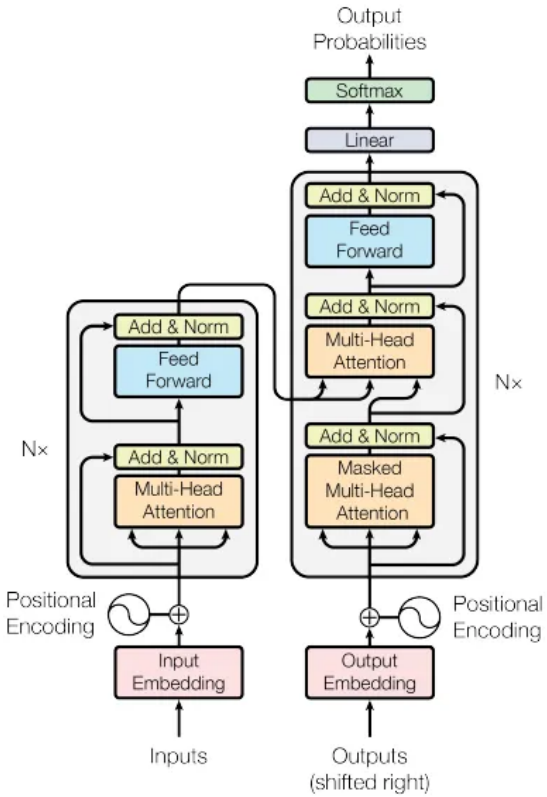
Yogesh Haribhau Kulkarni

YHK

Outline

Objective

To understand



From "Attention is all you need" paper by Vaswani, et al., 2017

Thats' it . . .

YHK

Assuming Familiarity with ...

- Linear Algebra: Vectors, Matrices
- Calculus: Gradient Descent
- Statistics: Distributions
- Machine Learning: Supervised, Softmax
- Deep Learning: Back-propagation,
- Natural Language Processing: Tokenization, Word Vectors, Seq2Seq (RNN, LSTM)

YHK

Sequence-to-Sequence (seq2seq)

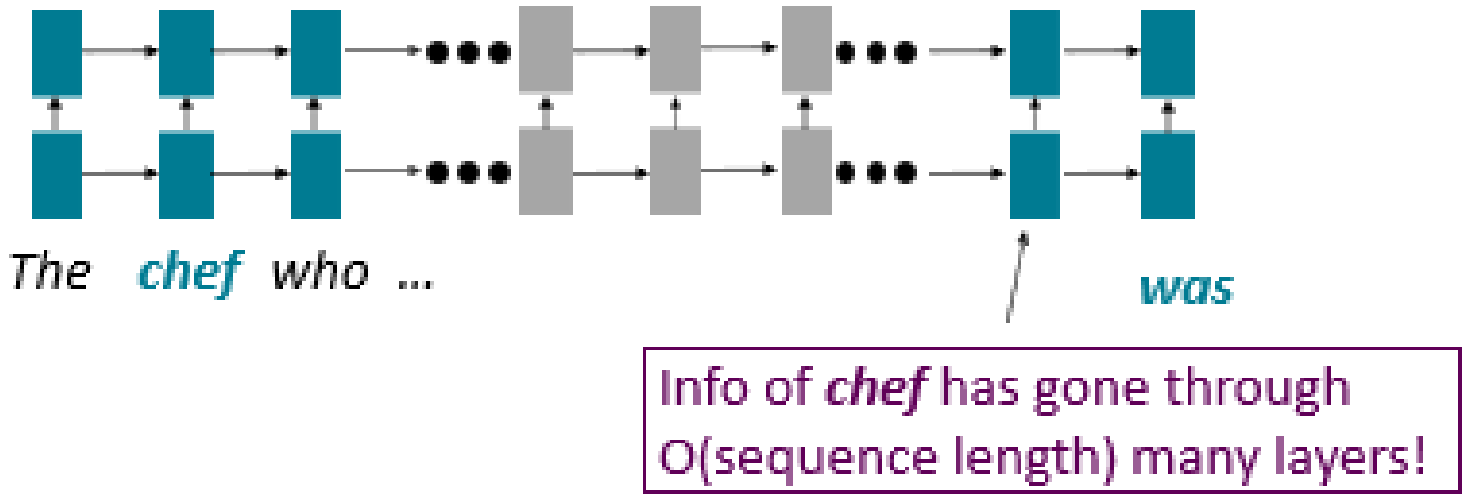
See any issues with this traditional seq2seq paradigm?

YHK

Issues with recurrent models

Linear interaction distance.

- Hard to learn long-distance dependencies (because of vanishing gradient problems!)
- Linear order of words is "baked in"; not necessarily the right way to think about sentences . . . Meaning sentence structure of one language may not be correspondingly same as order in the other language.

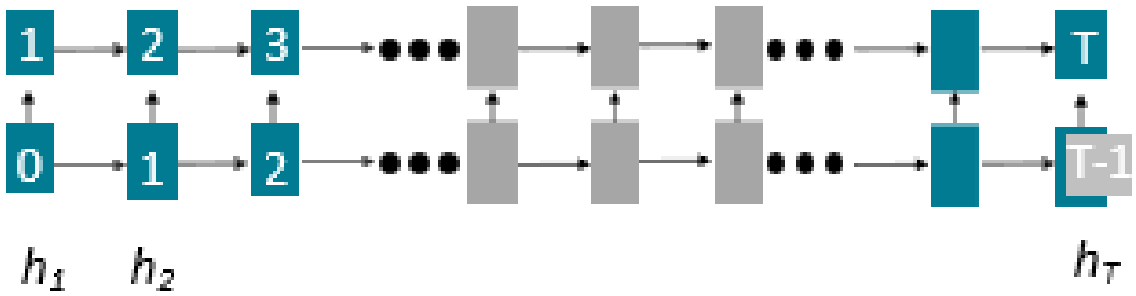


YHK

Issues with recurrent models

Lack of parallelizability.

- GPUs can perform a bunch of independent computations at once!
- But future RNN hidden states can't be computed in full before past RNN hidden states have been computed
- Inhibits training on very large datasets!



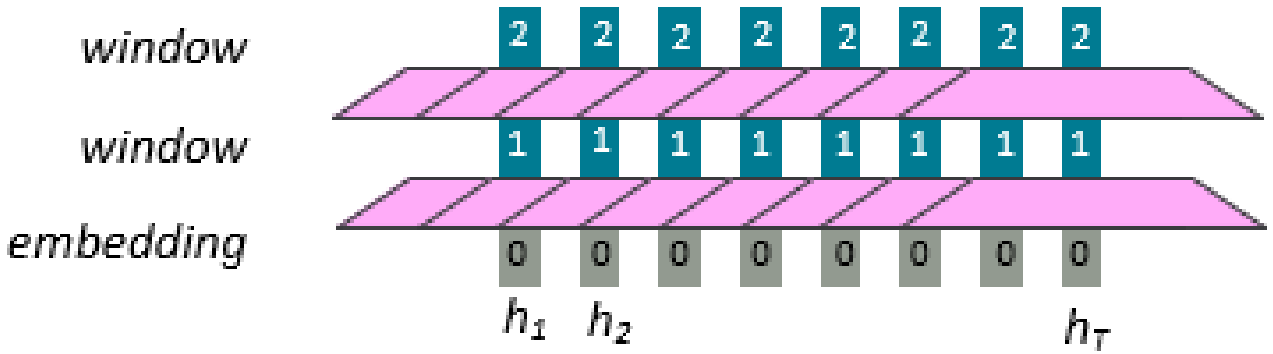
Numbers indicate min # of steps before a state can be computed

YHK

Then?

If not recurrence, then what? How about word windows? Word window models aggregate local contexts

- Also known as 1D convolution
- Number of unparallelizable operations not tied to sequence length!



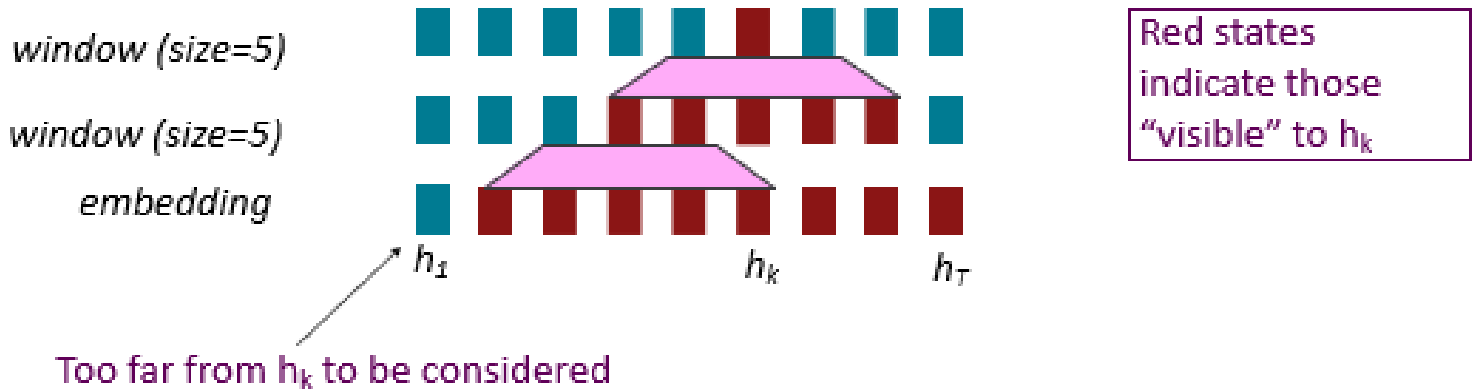
Numbers indicate min # of steps before a state can be computed

YHK

Then?

What about long-distance dependencies?

- Stacking word window layers allows interaction between farther words
- But if your sequences are too long, you'll just ignore long-distance context

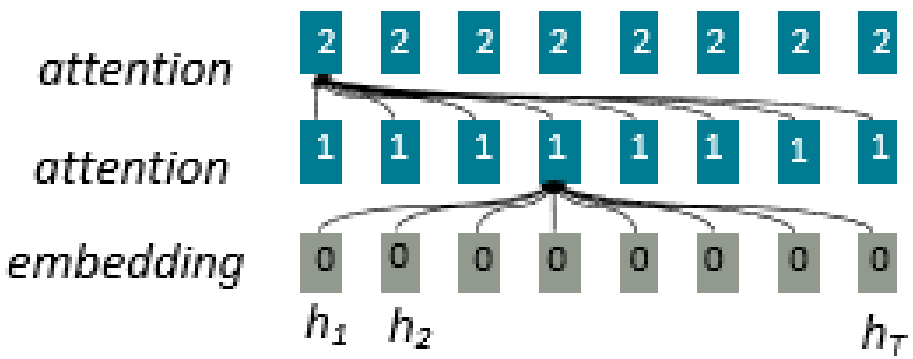


YHK

Attention

If not recurrence, then what? How about attention?

- Attention treats each word's representation as a query to access and incorporate information from a set of values.
- We saw attention from the decoder to the encoder; today we'll think about attention within a single sentence.
- If attention gives us access to any state. . . maybe we can just use attention and don't need the RNN?
- All words interact at every layer!



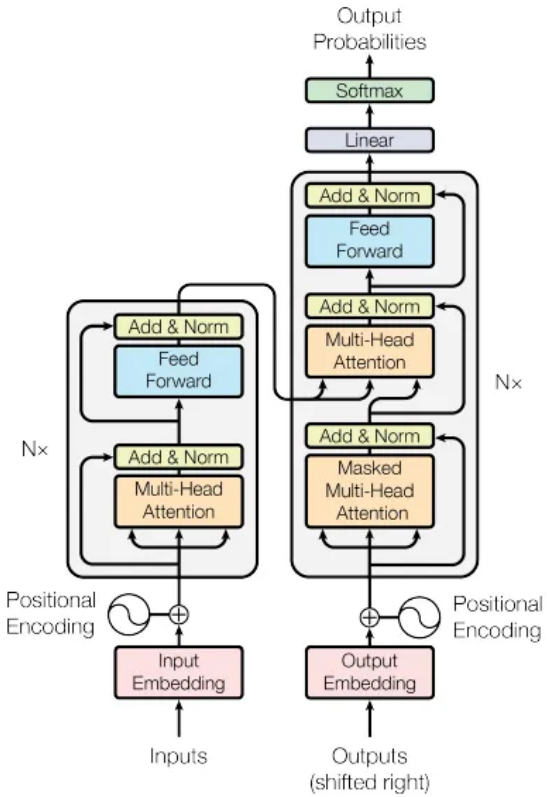
All words attend to all words in previous layer; most arrows here are omitted

YHK

Transformer

"Attention is all you need"

Goal: Study Transformer



(Ref: "Attention is all you need". 2017. Aswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser, Polosukhin

https://arxiv.org/pdf/1706.03762.pdf

YHK

A High-Level Look

Lets start with a very high-level view . . . and then Zoom in each sub-blocks. Here is the Transformer block and its i/o.

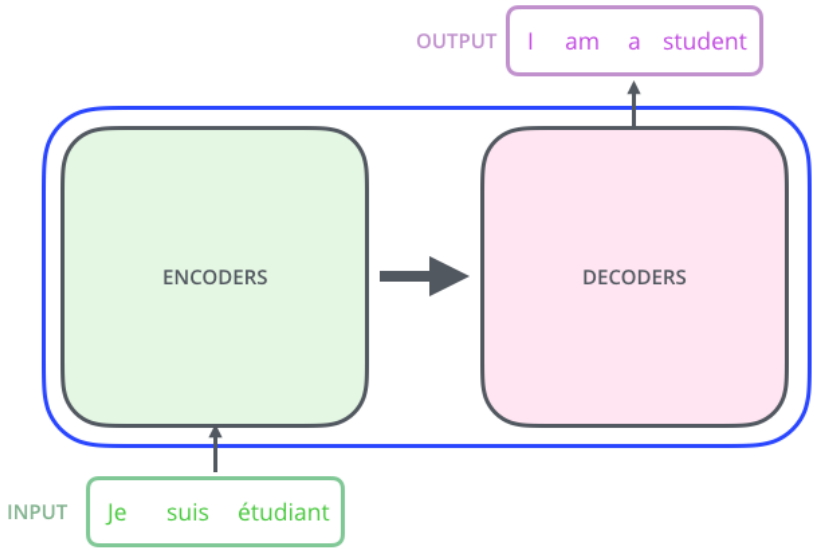
(Ref: "The Illustrated Transformer" - Jay Alammar)

- Main architecture: encoder-decoder ie sequence to sequence
- Task: machine translation with parallel corpus (encoder-decoder)
- Sub-tasks: Word Embedding (Encoder), Predict each translated word (Decoder)
- Can be done by RNNs, LSTMs, etc. But they had issues.
- Here we bring parallelization.
- Challenges? Different input-output sizes, different attention correspondence, word order matters, semantic preservation, compute cost, storage cost, etc.

YHK

One level down

Within Transformer block we have



(Ref: "The Illustrated Transformer" - Jay Alammar)

Inference time flow:

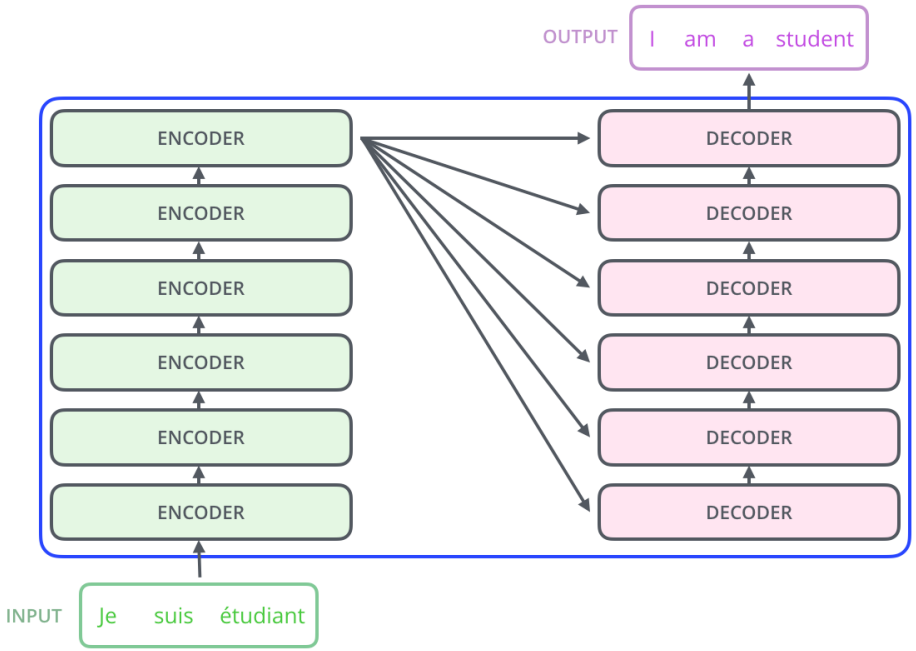
- Encoder takes input
- Does some processing, creates a 'latent' representation, sends it to decoder
- Decoder processes this input to generate the output

What would be ML training flow?

YHK

One level down

Each sub-block ie Encoder-Decoder blocks are actually stacks of encoders decoders.



(Ref: "The Illustrated Transformer" - Jay Alammar)

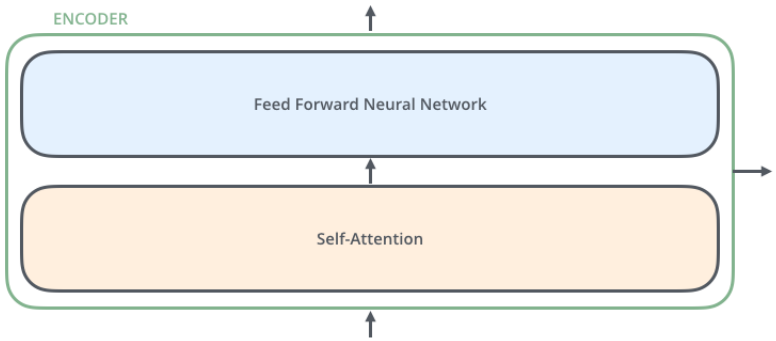
- Encoder has 6 blocks, so does Decoder.
- Why 6? Why not 7? They found it to be good. Like Hyper-parameter.

How many blocks GPT 3.5 has?

YHK

One level down in Encoder Cell

Each Encoder sub-block is actually stack of . . .



(Ref: "The Illustrated Transformer" - Jay Alammar)

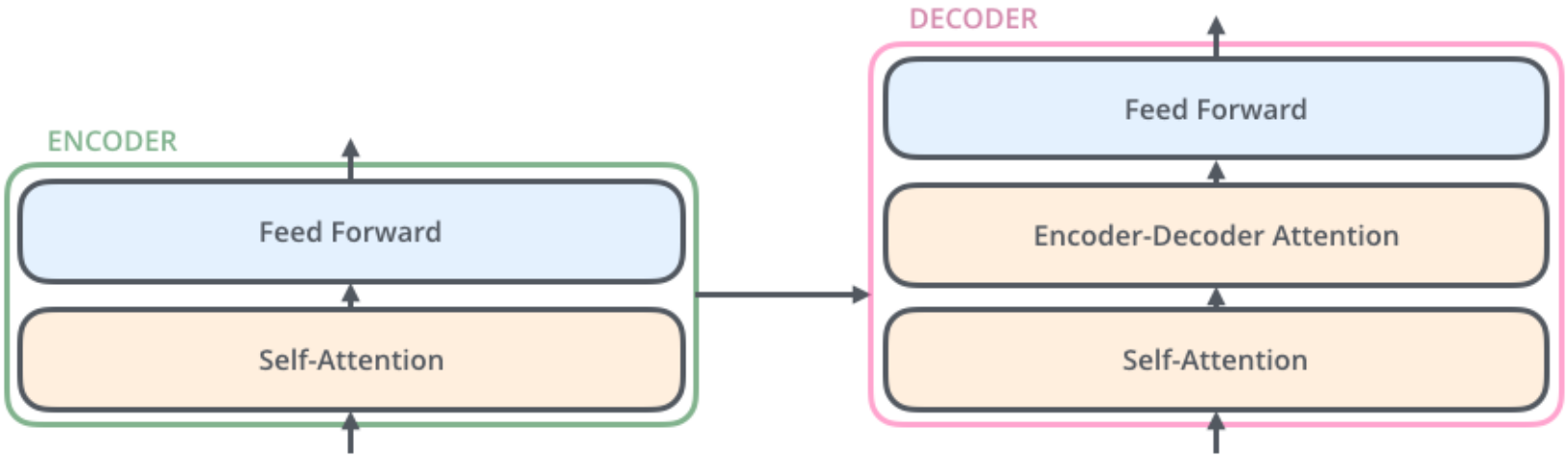
- Self Attention: a layer that helps the encoder look at other words in the input sentence as it encodes a specific word.
- Feed Forward: The outputs of the self-attention layer are fed to a feed-forward neural network. The exact same feed-forward network is independently applied to each position.

Input Embedding? Positional Embedding? Residual connections? batch normalization?

YHK

Similarly on Decoder side

Each Decoder sub-block is actually stack of . . .



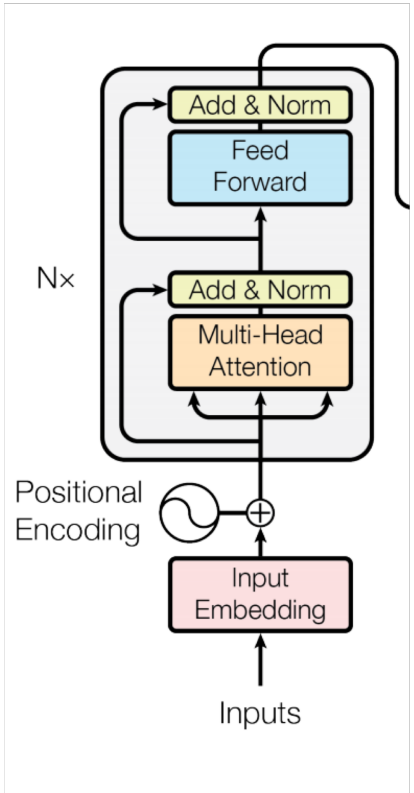
(Ref: "The Illustrated Transformer" - Jay Alammar)

- The decoder also has both those layers (Self but Masked Attention and Feed Forward), but between them is an (cross) attention layer that helps the decoder focus on relevant parts of the input sentence

YHK

Transformer Encoder

Actual architecture in the research paper.



- For encoder block, initial input, tokenization, embedding, positional encoding, happens at the start, then the first Encoder cell starts.
- In each cell, we have multi-head self attention, residual connections, batch normalization and then feed forward then again residual + batch normalization.
- Output of this cell is passed to the next encoder cell.
- Output of the last encoder cell in the encoder block is passed to the Decoder block.

- Mind well, that each word here gets processed parallelly, unlike RNN or LSTM.
- Cells are repeated 6 times (in vertical stack)

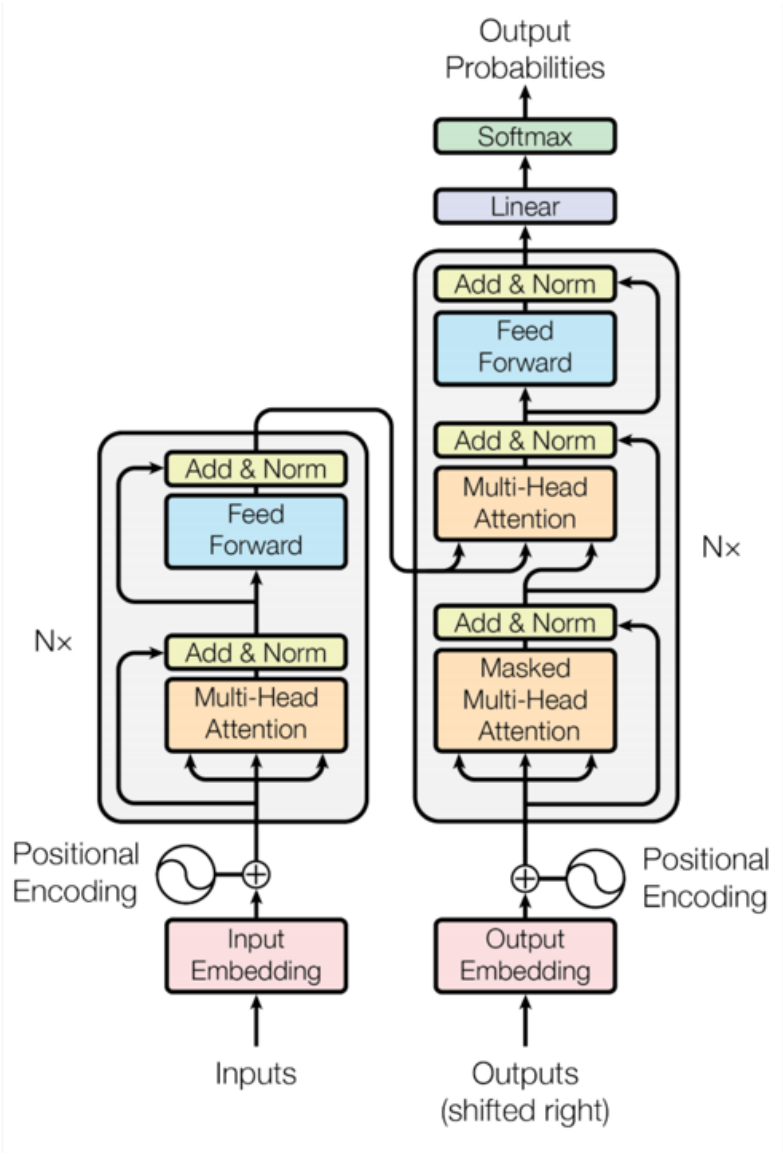
YHK

Transformer Decoder

- 2 sublayer changes in decoder
- Masked decoder self-attention on previously generated outputs:

- Encoder-Decoder Attention, where queries come from previous decoder layer and keys and values come from output of encoder

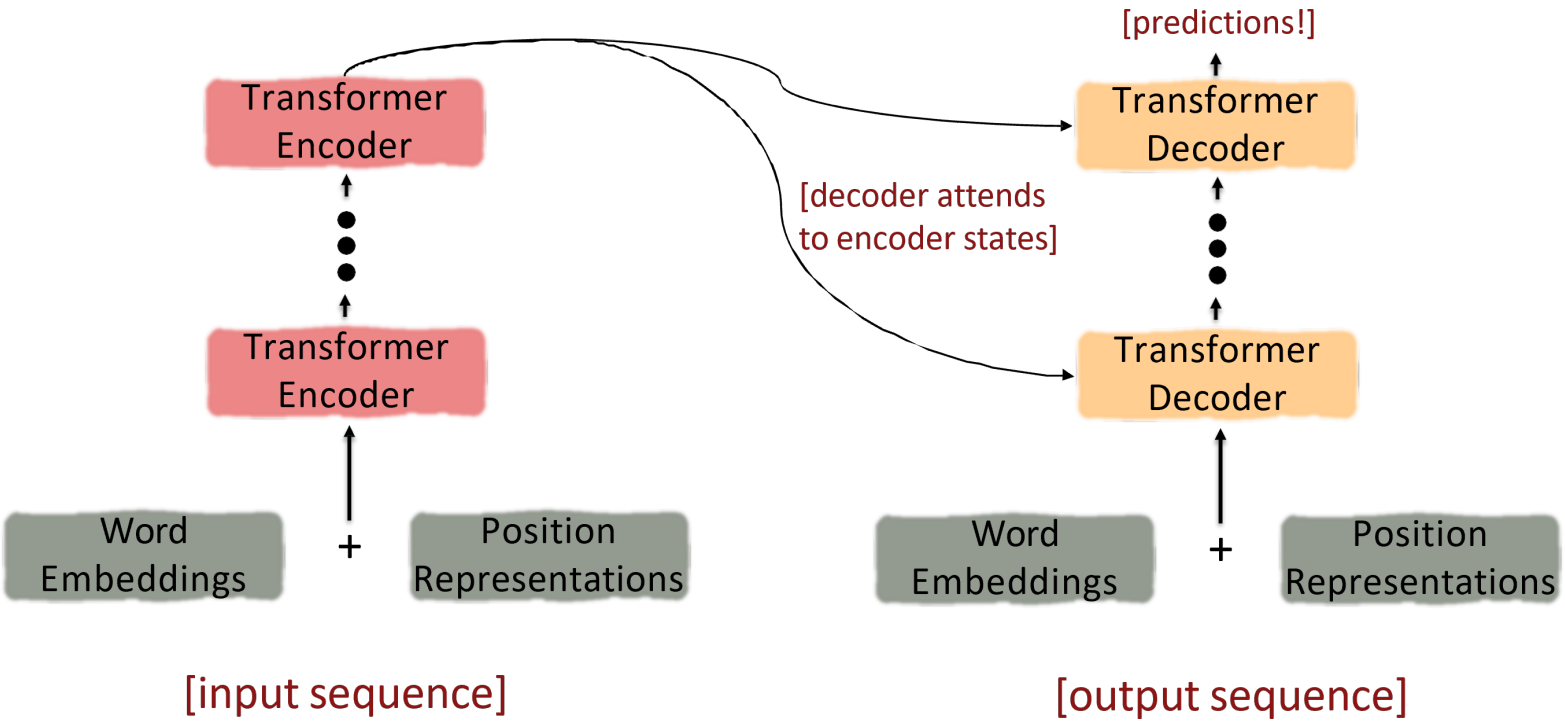
Blocks repeated 6 times also



YHK

Transformer Encoder-Decoder

Another look at the Transformer Encoder and Decoder Blocks at a high level



YHK

Transformer Encoder-Decoder

Next, let's look at the Transformer Encoder and Decoder Blocks in action

- Imagine Machine Translation: French to English
- What would be ML training flow?
- What would be ML inference flow?
- Can we use input or output text as is, ie in string format?
- Whats the NLP way of making text available for ML or DL algorithms?

YHK

Block: Input Embedding

YHK

Convert Sentence to Tokens

- Each word can be one or more tokens
- Subword tokenization, prefixes, suffixes, etc
- Highly Language dependent, imagine Sanskrit tokenization
- Each model will have its corresponding tokenizer, thusr forming vocabulary array, word-to-id
- "I love cats" could be tokenised to $[816, 8129, 312]$

For now, we can use 'Toekn' and 'Word' synonymously.

YHK

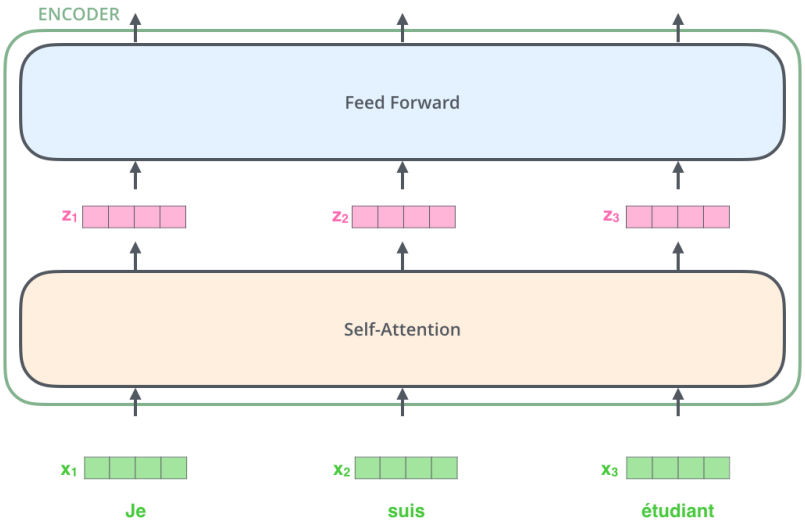
Convert Tokens to Vectors

(Ref: "The Illustrated Transformer" - Jay Alammar)

- Either by Frequency based ie one-hot or tf-idf or Neural way ie Word2Vec (in paper)
- Or put embedding layer (like in TensorFlow or Keras NLP, which also gets trained)
- Size: 512, a hyper-parameter we can set – basically it would be the length of the longest sentence in our training dataset.
- After embedding the words in our input sequence, each of them flows through each of the two layers of the encoder.

YHK

Flow of Vectors in Encoder Cell

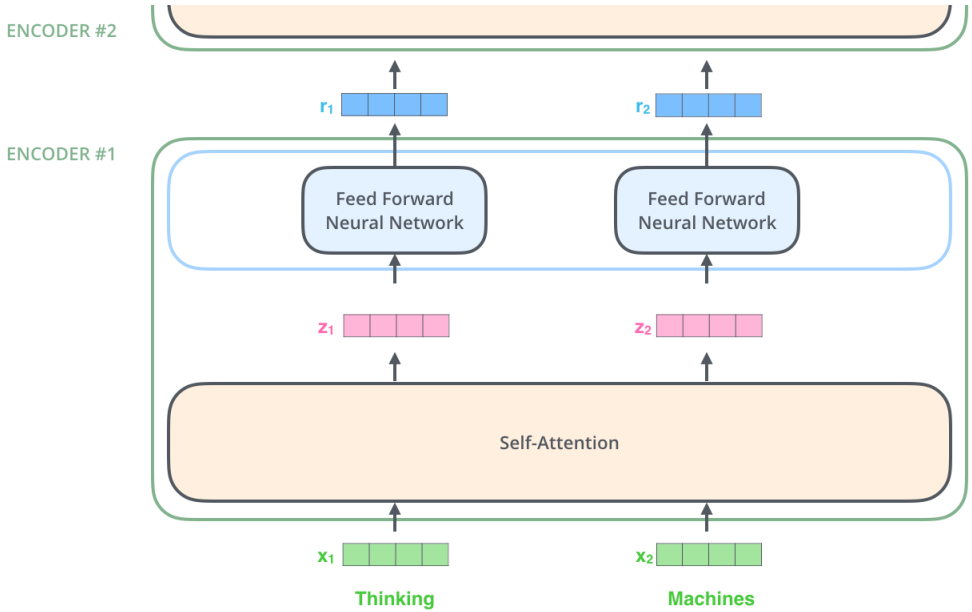


(Ref: "The Illustrated Transformer" - Jay Alammar)

- Key property : the word in each position flows through its own path in the encoder.
- Calculations in the self-attention layer do need other words.
- But later, in the feed-forward layer does not have those dependencies,
- thus the various paths can be executed in parallel while flowing through the feed-forward layer.

YHK

Flow of Vectors in Encoder Block



(Ref: "The Illustrated Transformer" - Jay Alammar)

- The word at each position passes through a self-attention process.
- Then, they each pass through a feed-forward neural network – the exact same network with each vector flowing through it separately.
- Meaning ALL weights in these cells, block are same for ALL words going through in each epoch, of course they get adjusted during back propagation.

YHK

Block: Positional Encoding

YHK

Why?

- In languages like English (unlike Sanskrit) word order matters, meaning 'Suresh ate chicken' and 'chicken ate Suresh' are two different things.
- As input is all word vectors together, parallel, (unlike RNN or LSTM where they are sequential ie one after another, there the order is preserved), the sense of position is lost.
- Need to add that information by unique position value/signature.
- There are many reasons why a single number, such as the index value, is not used to represent an item's position in transformer models.
- For long sequences, the indices can grow large in magnitude.
- If you normalize the index value to lie between 0 and 1, it can create problems for variable length sequences as they would be normalized differently.
- A cool mechanism (similar to Fourier Transform) was employed.
- This mechanism is such that uniqueness/signature of a position is achieved and also the distance requirement is fulfilled, meaning neighboring words have less distance and vice versa.

YHK

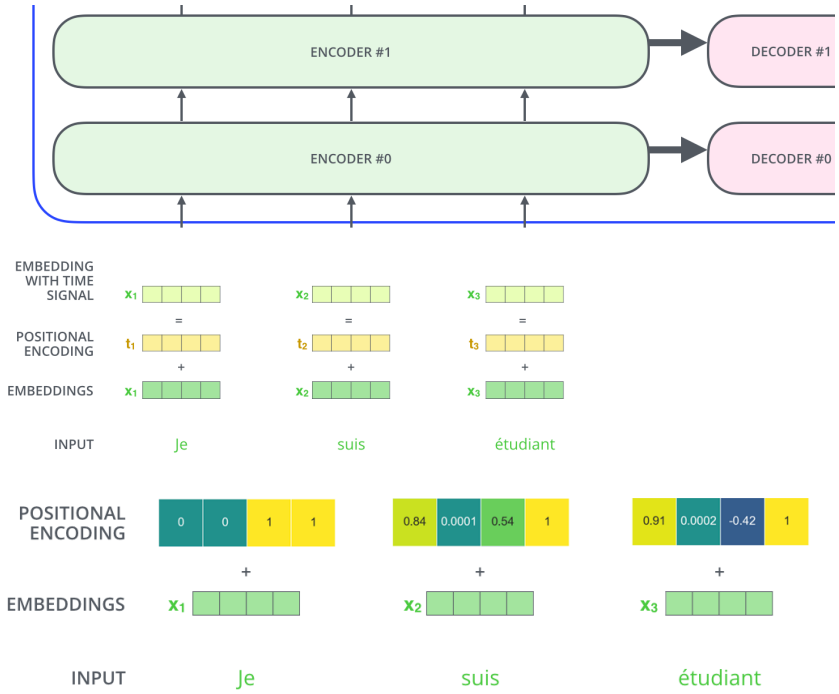
What?

Note: Confirm that uniqueness/signature of a position is achieved and also the distance requirement is fulfilled, meaning neighboring words have less distance and vice versa.

```
Position 1: [0.1, 0.2, 0.3]
Position 2: [0.4, 0.5, 0.6]
Position 3: [0.7, 0.8, 0.9]

Combine Embeddings and Positional Encodings

I (position 1) = [0.34 + 0.1, -0.21 + 0.2, 1.21 + 0.3] = [0.44, -0.01, 1.51]
love (position 2) = [-1.36 + 0.4, 0.98 + 0.5, 0.23 + 0.6] = [-0.96, 1.48, 0.83]
cats (position 3) = [0.54 + 0.7, 0.68 + 0.8, 0.49 + 0.9] = [1.24, 1.48, 1.39]
```

YHK

Representing The Order of The Sequence Using Positional Encoding



(Ref: "The Illustrated Transformer" - Jay Alammar)

- Special pattern based vectors encode position or the distance between different words in the sequence.
- Using sine waves of different phase lags and amplitude to create unique set of basis functions, equal to number of words in the sentence.
- They are added to input word embedding.

YHK

Positional Patterns

- The Position Encoding is computed independently of the input sequence.
- These are fixed values that depend only on the max length of the sequence.

$PE_{(pos,2i)} = sin(pos/n^{2i/d_{model}})$
$PE_{(pos,2i+1)} = cos(pos/n^{2i/d_{model}})$

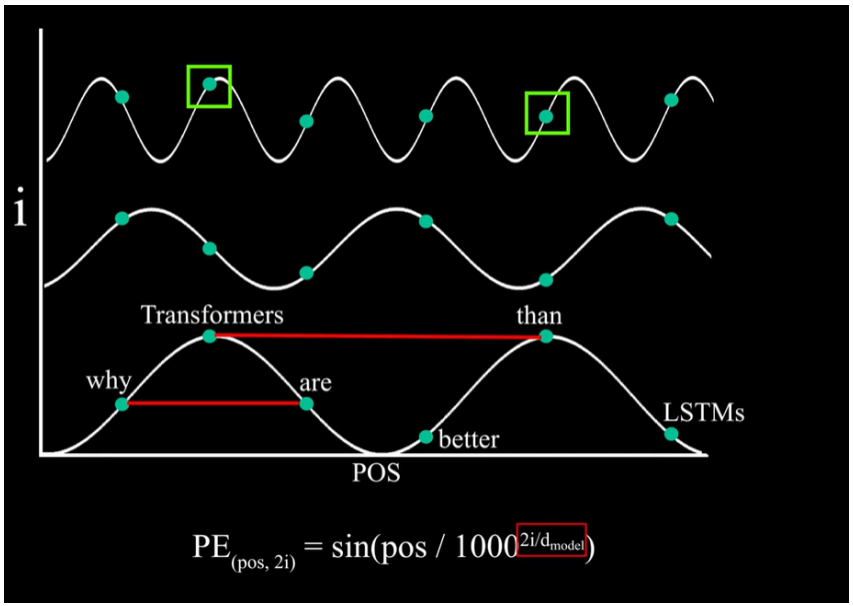
- pos is the position of the word in the sequence
- n is User-defined scalar, set to 10,000 by the authors of Attention Is All You Need.
- d_{model} is the length of the encoding vector (same as the embedding vector) and
- i is the index value into this vector.Used for mapping to column indices $0 \leq i < d/2$, with a single value of i maps to both sine and cosine functions

(Ref: Transformers Explained Visually - How it works, step-by-step - Ketan Doshi)

YHK

What Formula Means?

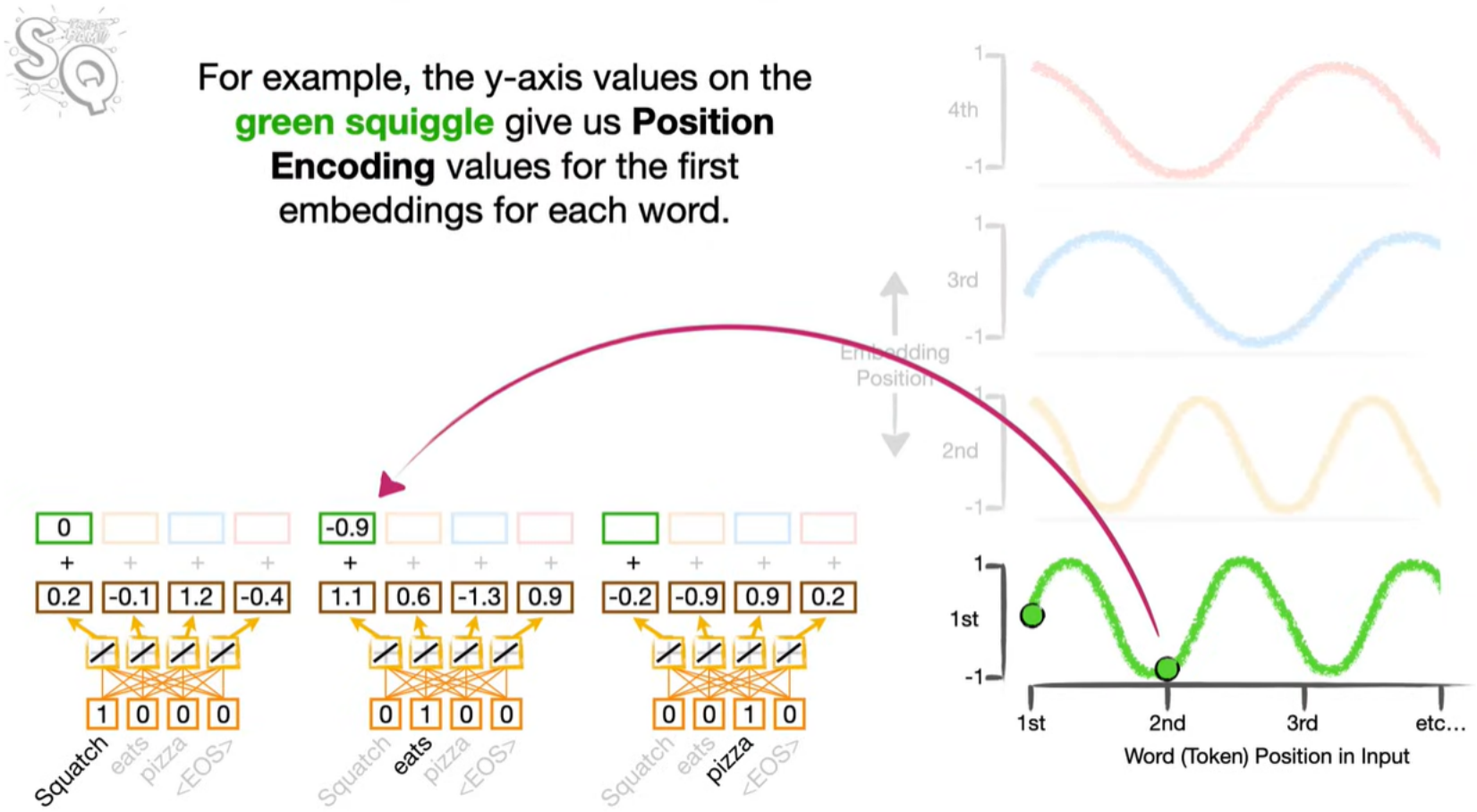
- *pos* is on x axis and encoded value is on y axis.
- If we had used just one wave, two words, say 'Why' and 'are' would have got same y value, so no use!!
- Have more waves, with phase lag and different frequencies (due to $2i/d_{model}$ part of the equation
- So, for these words, their y values for diff waves would be different, making them unique.



(Ref: "Attention is all you need —— Transformers Explained —— Quick Explained" - Developers Hutt)

YHK

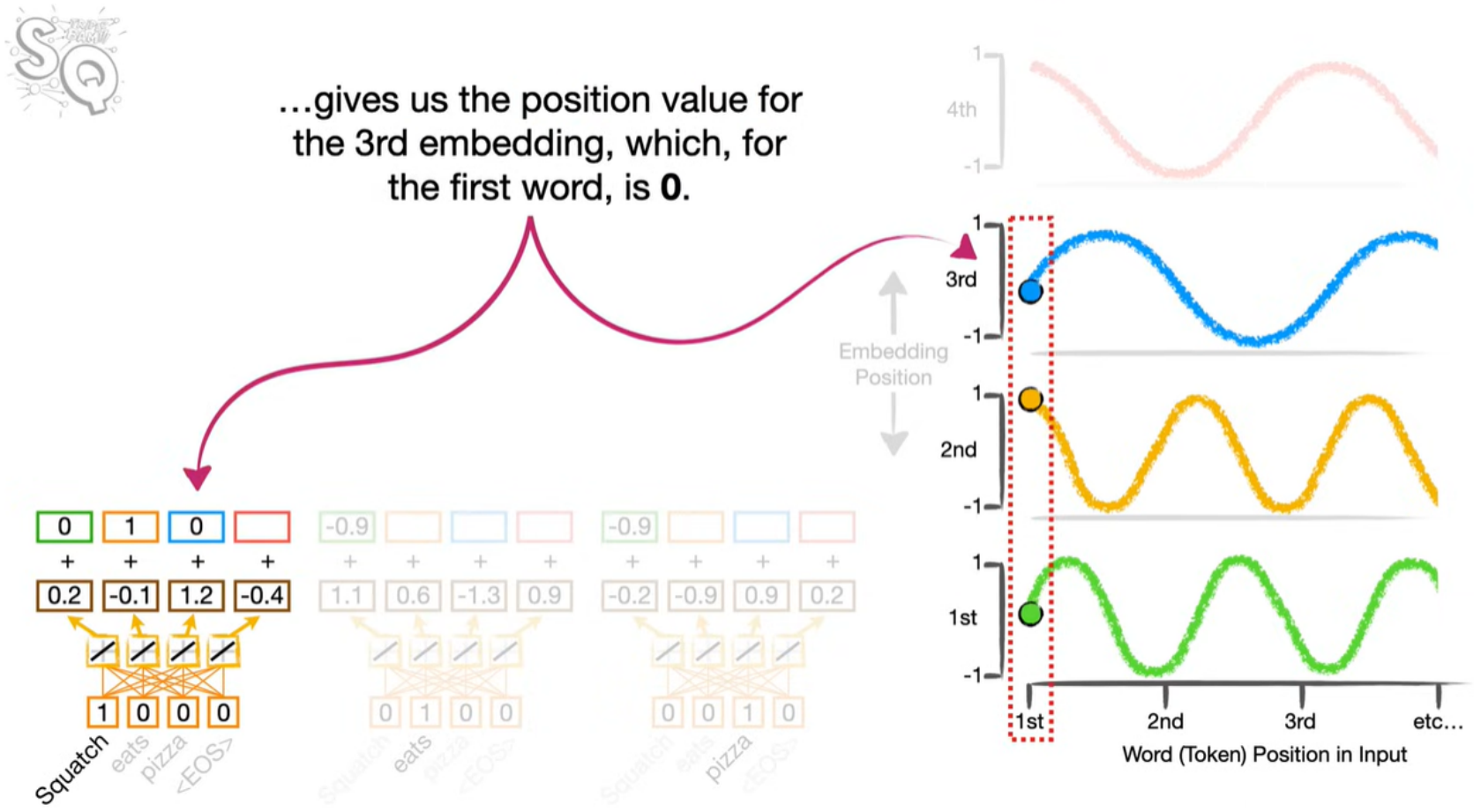
Mechanism



Transformer Neural Networks, ChatGPT's foundation, Clearly Explained!!!

(Ref: StatQuest with Josh Starmer)

Mechanism



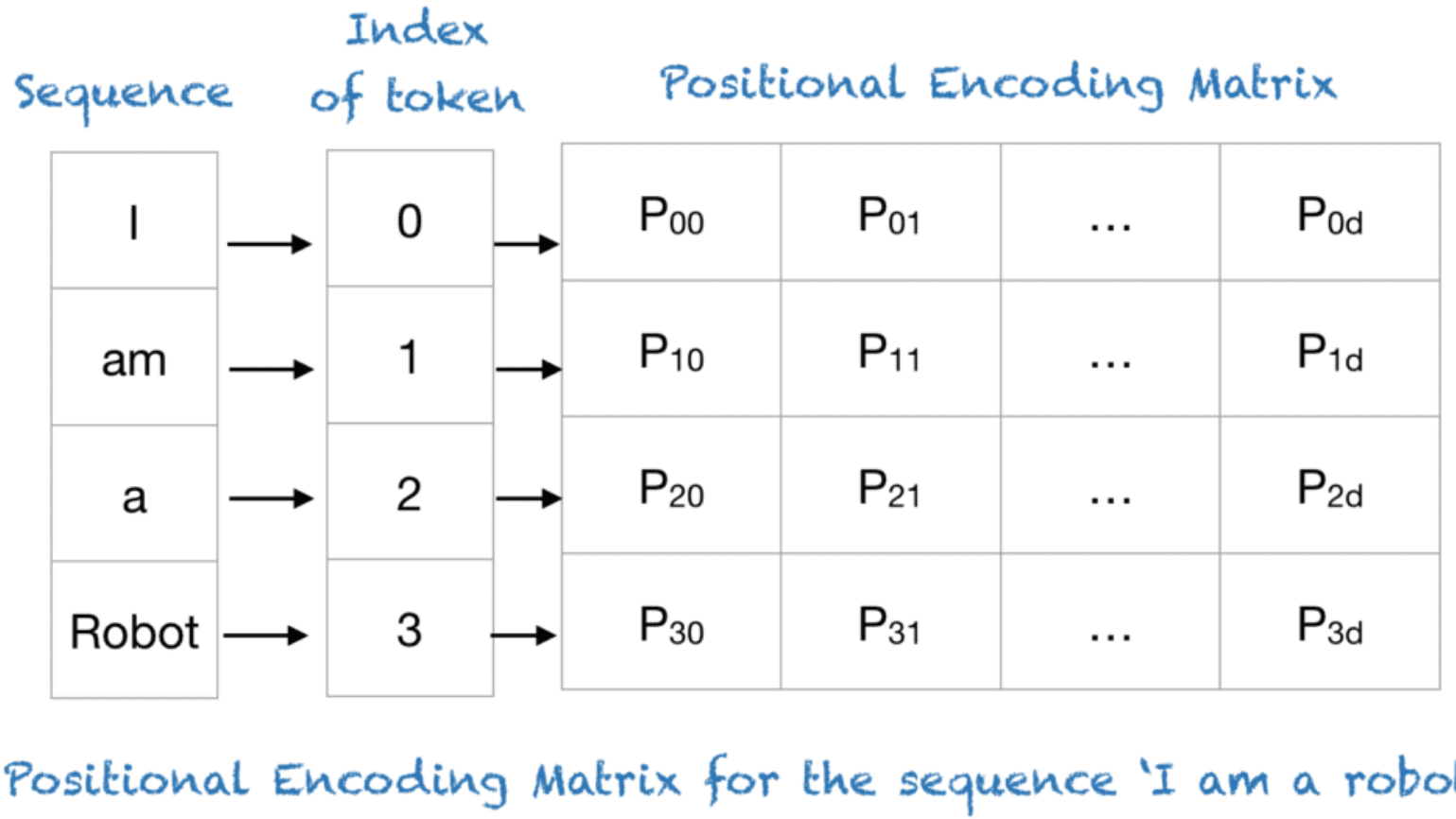
Transformer Neural Networks, ChatGPT's foundation, Clearly Explained!!!

(Ref: StatQuest with Josh Starmer)

YHK

Mechanism

Each position/index is mapped to a vector

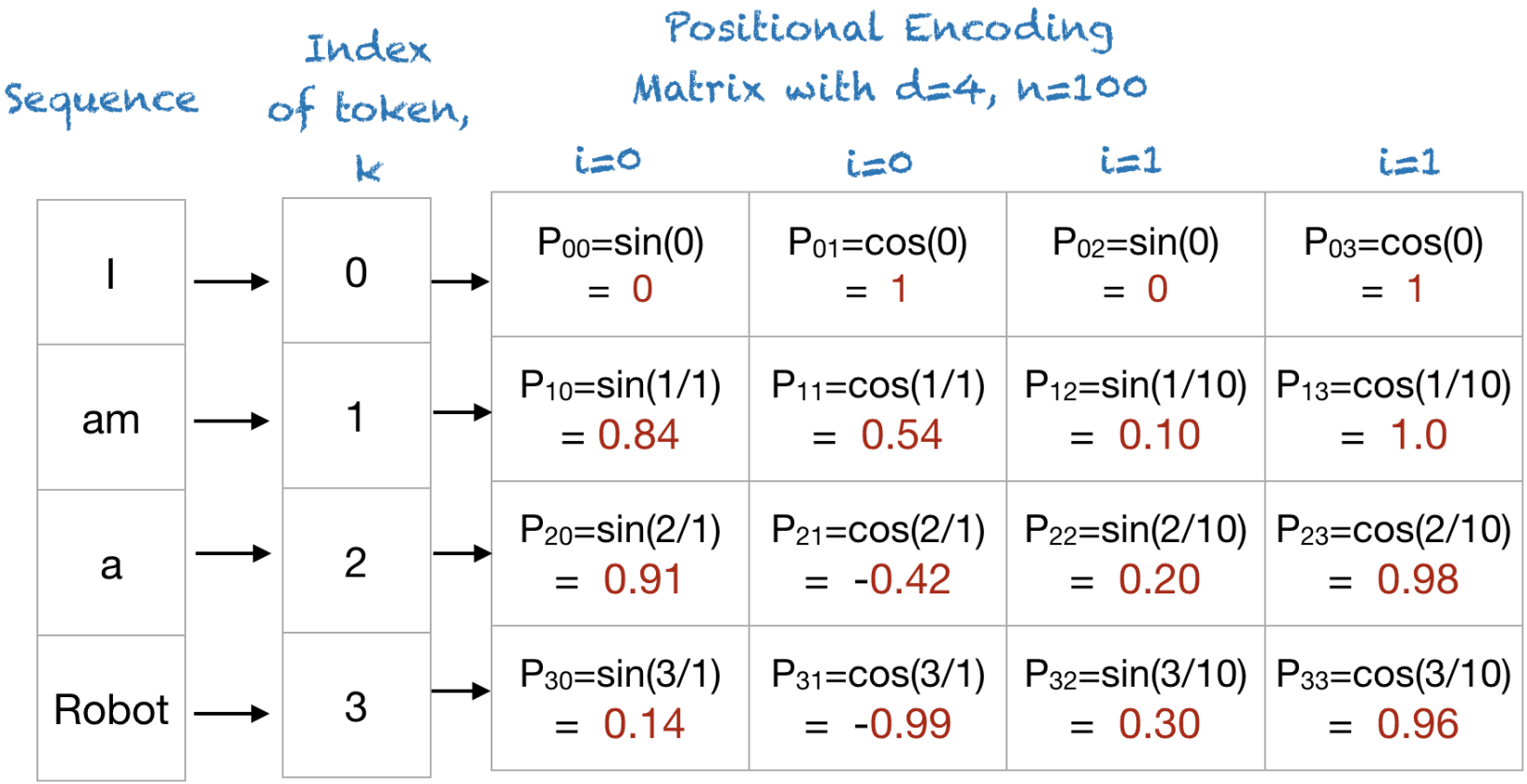


(Ref: "A Gentle Introduction to Positional Encoding in Transformer Models, Part 1" - Mehreen Saeed)

YHK

Example

Each position/index is mapped to a vector



Positional Encoding Matrix for the sequence 'I am a robot'

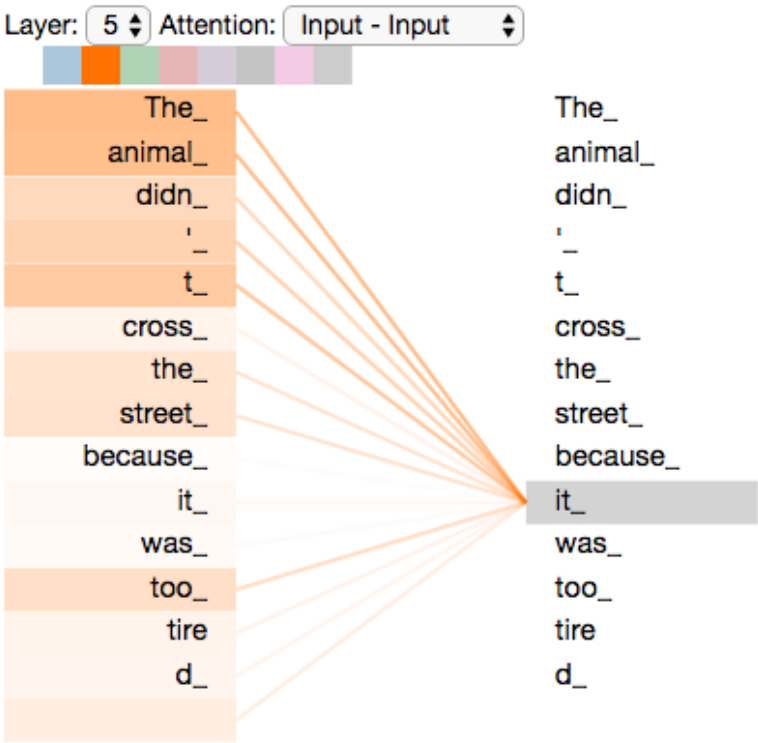
(Ref: "A Gentle Introduction to Positional Encoding in Transformer Models, Part 1" - Mehreen Saeed)

YHK

Self Attention

YHK

Self-Attention at a High Level



(Ref: "The Illustrated Transformer" - Jay Alammar)

- "The animal didn't cross the street because it was too tired", what "it" refers to?
- the street or to the animal?
- For "it", self-attention mechanism (some how, or should) allows it to associate "it" with "animal".
- ie for each word (each position in the input sequence), self attention allows it to look at other positions in the input sequence for clues that can help lead to a better encoding for this word.
- Lets see how that can be achieved.

YHK

Self-Attention in Principle

- Slight digression: lets say you have database of key(k)-value(v) pairs, then you have a query (q), which needs to go and search in the database and get corresponding or closest value.
- So, q is matched (similarity) with all ks, whichever matches most, its v is returned.
- The Query word can be interpreted as the word for which we are calculating Attention. The Key and Value word is the word to which we are paying attention ie. how relevant is that word to the Query word.
- For example, for the sentence, "The ball is blue", the row for the word "blue" will contain the attention scores for "blue" with every other word. Here, "blue" is the Query word, and the other words are the "Key/Value".

YHK

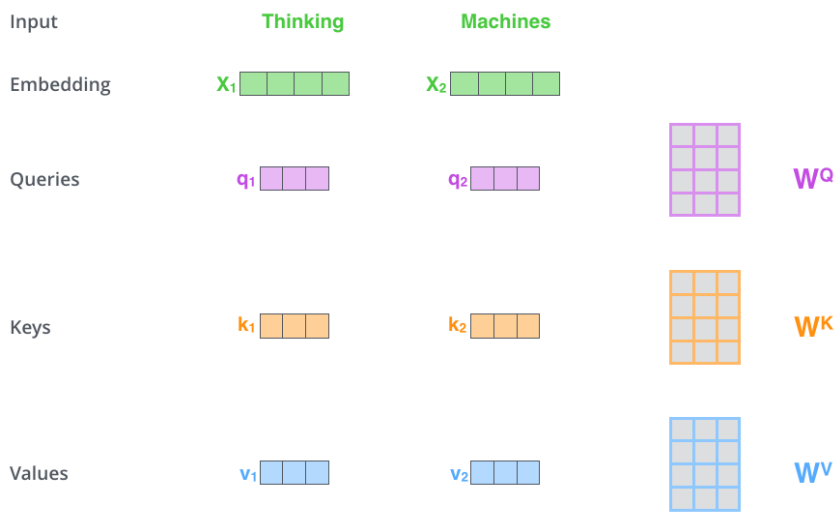
Self-Attention in Principle

W_q, W_k, and W_v are learned during training and are shared between all of the embedding vectors.

```
I = ‘[0.44, -0.01, 1.51]‘
love = ‘[-0.96, 1.48, 0.83]‘
cats  = ‘[1.24, 1.48, 1.39]‘

‘‘I’’ =
‘[0.44, -0.01, 1.51] * W_q = Q_1‘
‘[0.44, -0.01, 1.51] * W_k = K_1‘
‘[0.44, -0.01, 1.51] * W_v = V_1‘

‘‘love’’ =
‘[-0.96, 1.48, 0.83] * W_q = Q_2‘
‘[-0.96, 1.48, 0.83] * W_k = K_2‘
‘[-0.96, 1.48, 0.83] * W_v = V_2‘

‘‘cats’’ =
‘[1.24, 1.48, 1.39] * W_q = Q_3‘
‘[1.24, 1.48, 1.39] * W_k = K_3‘
‘[1.24, 1.48, 1.39] * W_v = V_3‘
```

YHK

Self-Attention in Detail



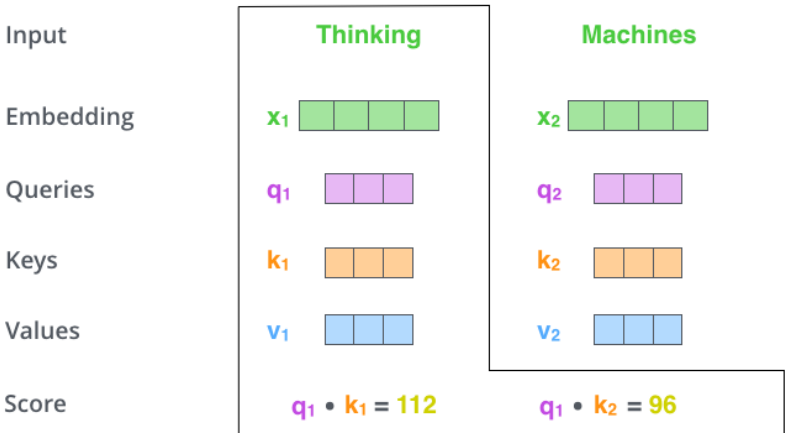
(Ref: "The Illustrated Transformer" - Jay Alammar)

- Create three vectors from each of the encoder's input vectors (in this case, the embedding of each word).
- So for each word, we create a Query vector, a Key vector, and a Value vector.
- These vectors are created by multiplying the embedding by three matrices, which are trained during the training process.

Multiplying x_1 by the W^Q weight matrix produces q_1, the "query" vector associated with that word. We end up creating a "query", a "key", and a "value" projection of each word in the input sentence.
Note: new vectors are smaller, 64, while the embedding and encoder input/output vectors have dimensionality of 512. They don't HAVE to be smaller, just a choice.

YHK

Self-Attention Scores



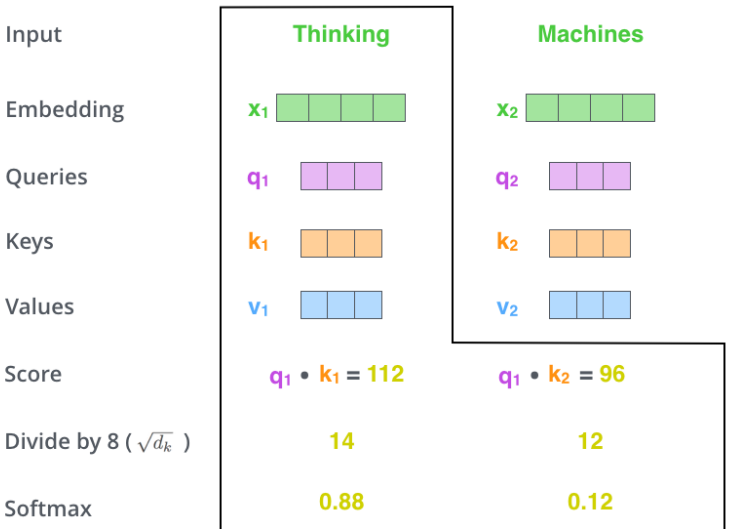
(Ref: "The Illustrated Transformer" - Jay Alammar)

- Calculating the self-attention for the first word in this example, "Thinking".
- Need to score each word of the input sentence against this word, thats its importance wrt that word
- score = dot product of words query vector with key vector of the running words. Each is a single scalar value. So, score for first word $w_1, s_1 = [q_1 k_1, q_1 k_2, \ldots]$

* Please note that in the diagram the Embedding vectors also include positional encoding incorporated inside.

YHK

Self-Attention Scores



(Ref: "The Illustrated Transformer" - Jay Alammar)

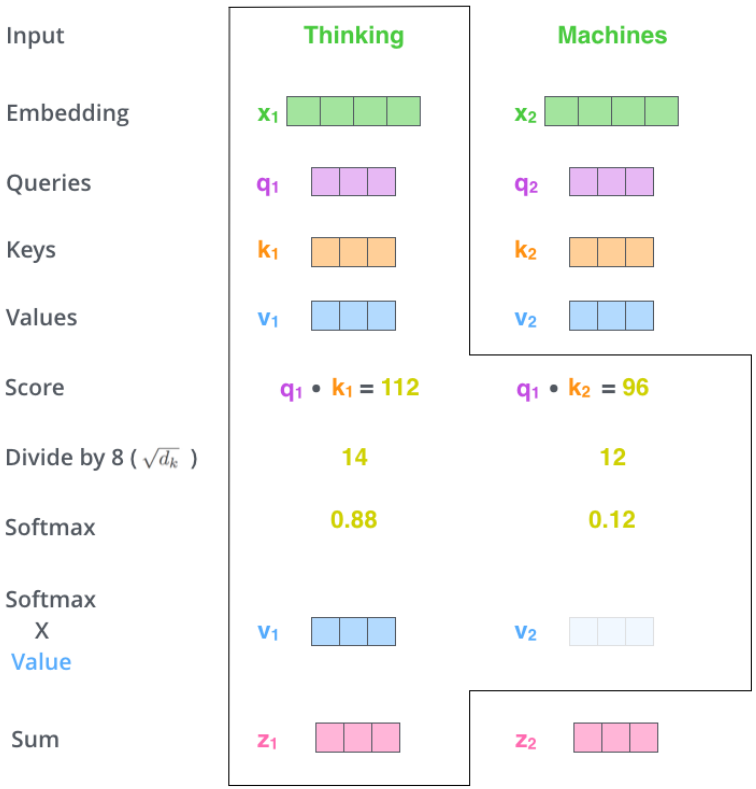
- Divide the scores by 8 (the square root of the dimension of the key vectors used in the paper – 64.
- This leads to having more stable gradients. There could be other possible values here, but this is the default),
- then pass the result through a softmax that normalizes from 0 to 1

Note: Clearly word with itself will have the highest softmax score, thats ok but then next matching-high scores have importance

* Please note that in the diagram the Embedding vectors also include positional encoding incorporated inside.

YHK

Self-Attention Scores with Value Vectors



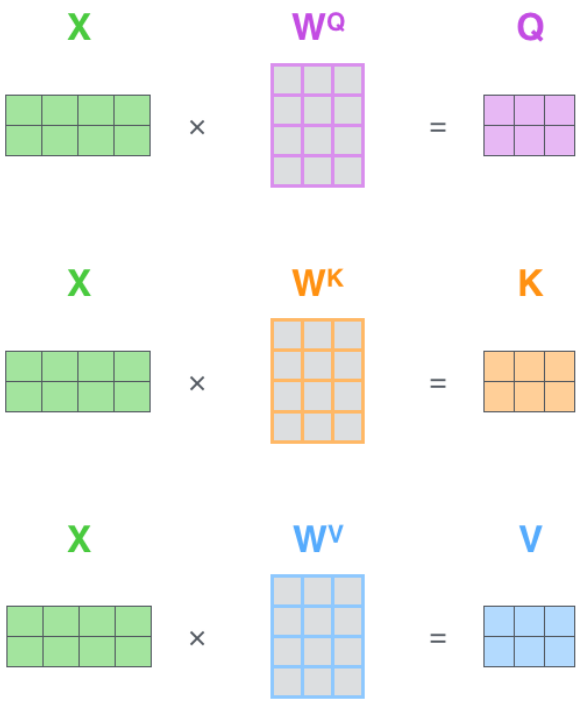
(Ref: "The Illustrated Transformer" - Jay Alammar)

- Multiply each value vector by the softmax scores respectively, like $s_1 \times v_1$ and $s_2 \times v_2$, here ss are scalars scores or weights, where as vs are value vectors.
- The intuition here is to keep intact the values of the word(s) we want to focus on, and drown-out irrelevant words, basically weighted per position.
- The last step is to sum up the weighted value vectors.
- This produces the output of the self-attention layer at this position (for the first word).

* Please note that in the diagram the Embedding vectors also include positional encoding incorporated inside.

YHK

Self-Attention Scores in a Matrix Way



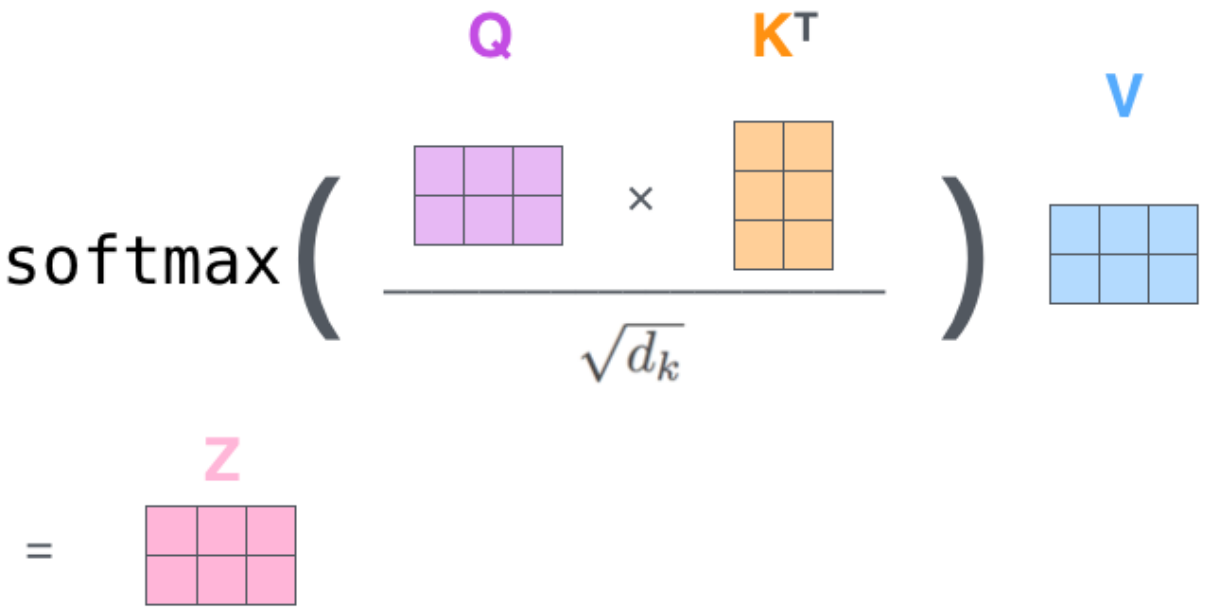
- Assuming two words in the sentence, so number of rows are two, num columns are 512 (but shown 4 just for brevity) and the q/k/v vectors (64, or 3 boxes in the figure)
- Calculate the Query, Key, and Value matrices.
- Input sentence vector is matrix X, and multiplying it by the weight matrices to be trained (W^Q, W^K, W^V).

(Ref: "The Illustrated Transformer" - Jay Alammar)

* Please note that in the diagram the Embedding vectors X also include positional encoding incorporated inside.

YHK

Self-Attention Scores in a single formula



(Ref: "The Illustrated Transformer" - Jay Alammar)

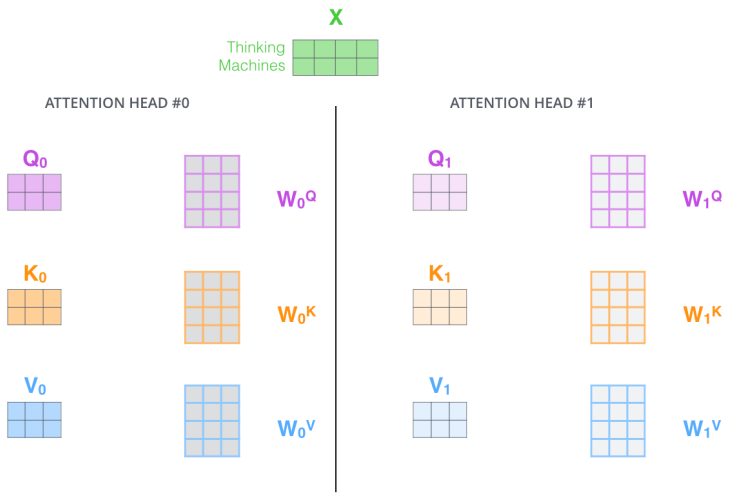
YHK

Multi-head Attention

YHK

The Beast With Many Heads

Having multiple self attention helps:



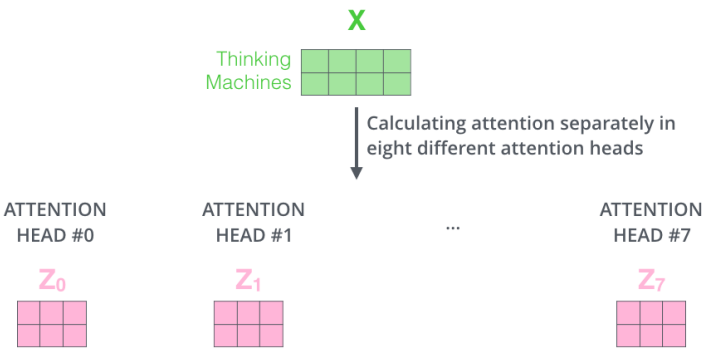
(Ref: "The Illustrated Transformer" - Jay Alammar)

- Expands the model's ability to focus on different positions. Because each weight matrices of Q K V are initialized randomly, they can get settled differently, and also differently for different sentences.
- Paper uses 8 heads in a single multi-head self attention cell.
- ie Each of these sets is randomly initialized. Then, after training, each set is used to project the input embeddings (or vectors from lower encoders/decoders) into a different representation subspace.

YHK

The Beast With Many Heads

Having multiple self attention helps:



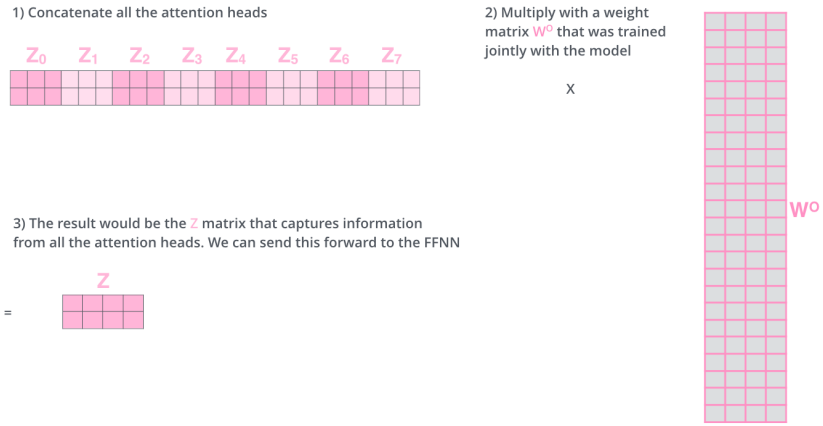
(Ref: "The Illustrated Transformer" - Jay Alammar)

- One problem though.
- The next feed-forward layer is not expecting eight matrices – it's expecting a single matrix (a vector for each word). So we need a way to condense these eight down into a single matrix.
- How do we do that?

YHK

The Beast With Many Heads

Having multiple self attention helps:

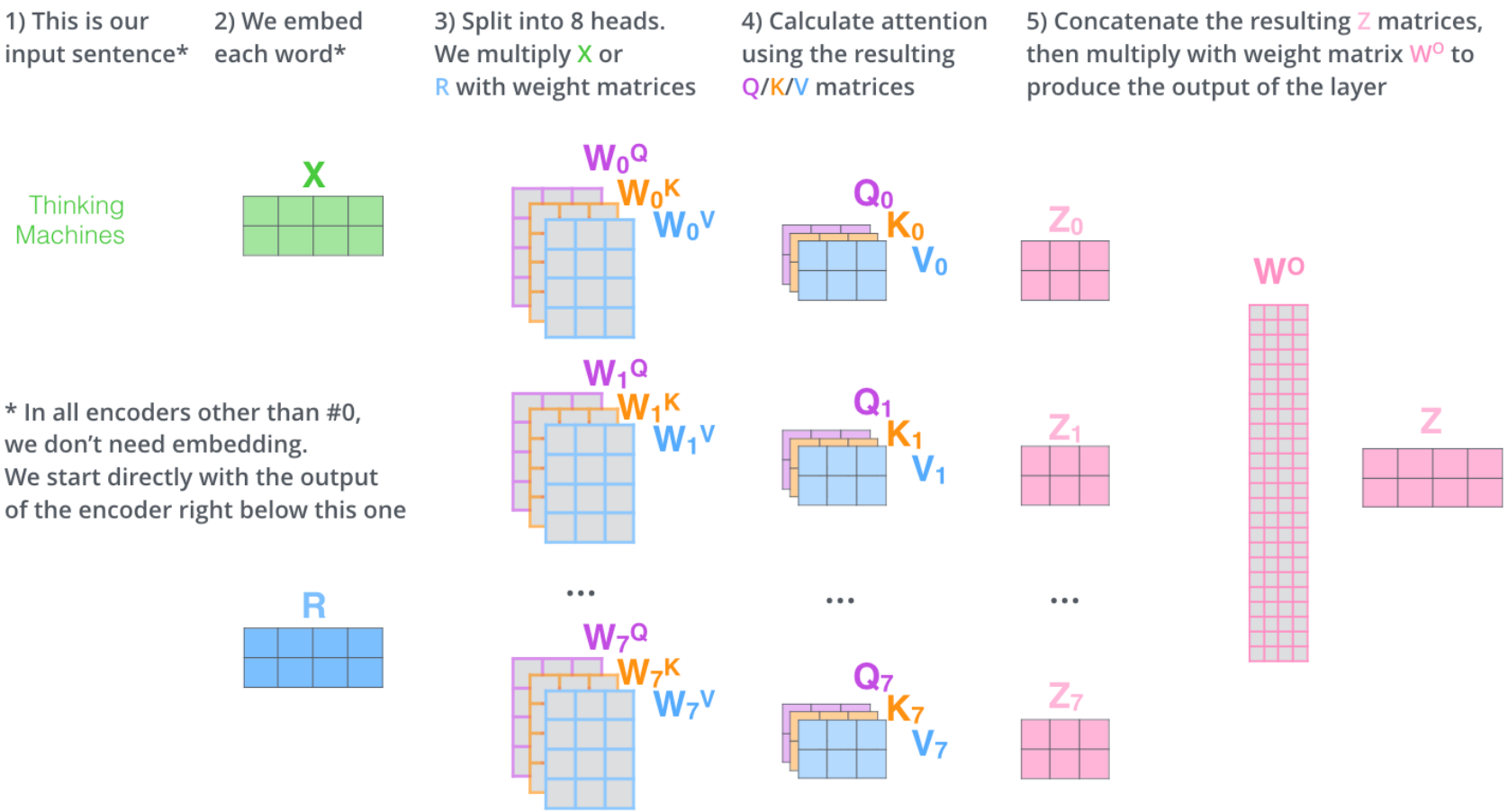


(Ref: "The Illustrated Transformer" - Jay Alammar)

- We concat the matrices then multiply them by an additional weights matrix W^O.

YHK

Self Attention in Summary

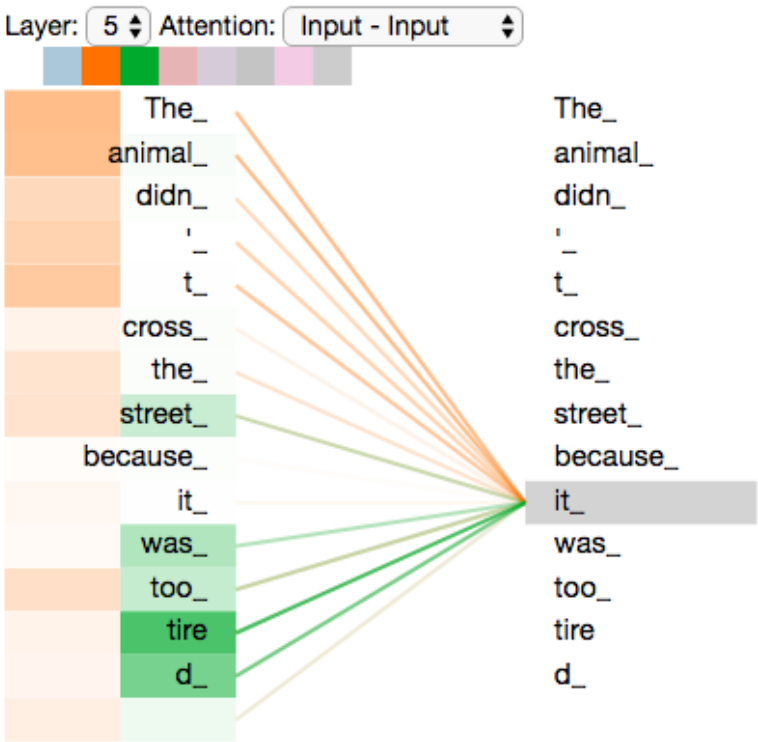


(Ref: "The Illustrated Transformer" - Jay Alammar)

* Please note that in the diagram the Embedding vectors X also include positional encoding incorporated inside. R shown in the diagram refers to

YHK

Self Attention in Summary Example

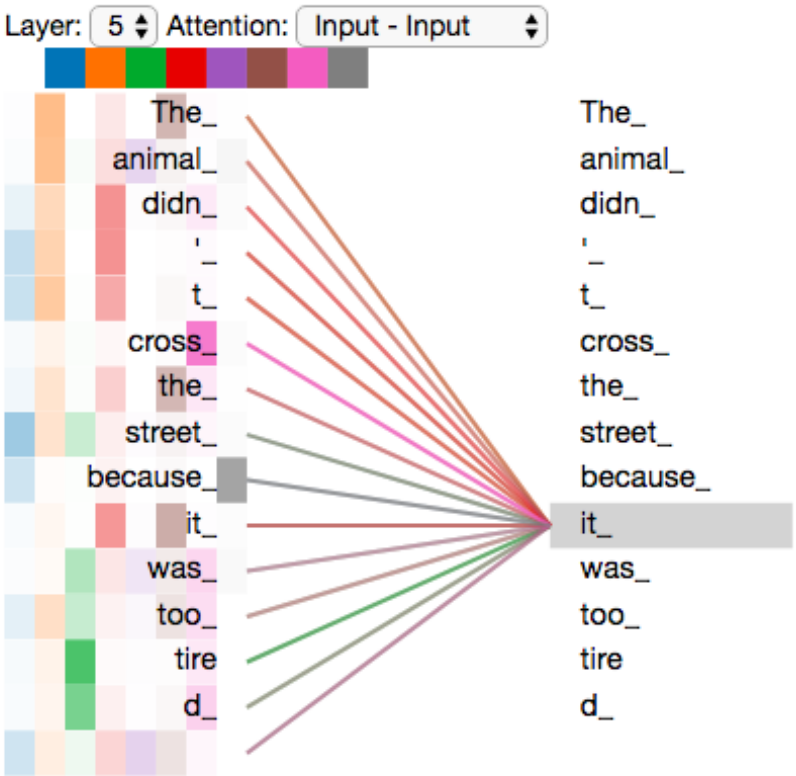


(Ref: "The Illustrated Transformer" - Jay Alammar)

As we encode the word "it", one attention head is focusing most on "the animal", while another is focusing on "tired" – in a sense, the model's representation of the word "it" bakes in some of the representation of both "animal" and "tired".

YHK

Self Attention in Summary Example



(Ref: "The Illustrated Transformer" - Jay Alammar)

If we add all the attention heads to the picture, however, things can be harder to interpret, but thats what happens. This is how Self-Attention block generates, "attended" vectors.

YHK

Feed Forward

YHK

Feed Forward

- The feed-forward layer operates on the output of the multi-headed self-attention layer.
- It treats each position of the global representation independently.
- It does not take the relationship between elements into consideration.
- The feed-forward layers contain weights that are tuned during training.
- There are two linear transformations/layers (W_1, W_2) with a non-linear activation function (usually ReLU) in between.
- This non-linearity allows the model to capture complex relationships in data, relationships which aren't directly related.
- The result of the feed-forward layer is added back to the global_representation matrix through a residual connection
- Note that that this operation is addition, rather than concatenation. This means that residual_output is the global_representation matrix enhanced by the knowledge gleaned from the feed-forward neural network.
- residual_output is normalised. This step is necessary to reduce internal covariate shift and stabilise training. To put it simply, if internal parameters change too rapidly between each layer the model cannot learn properly.

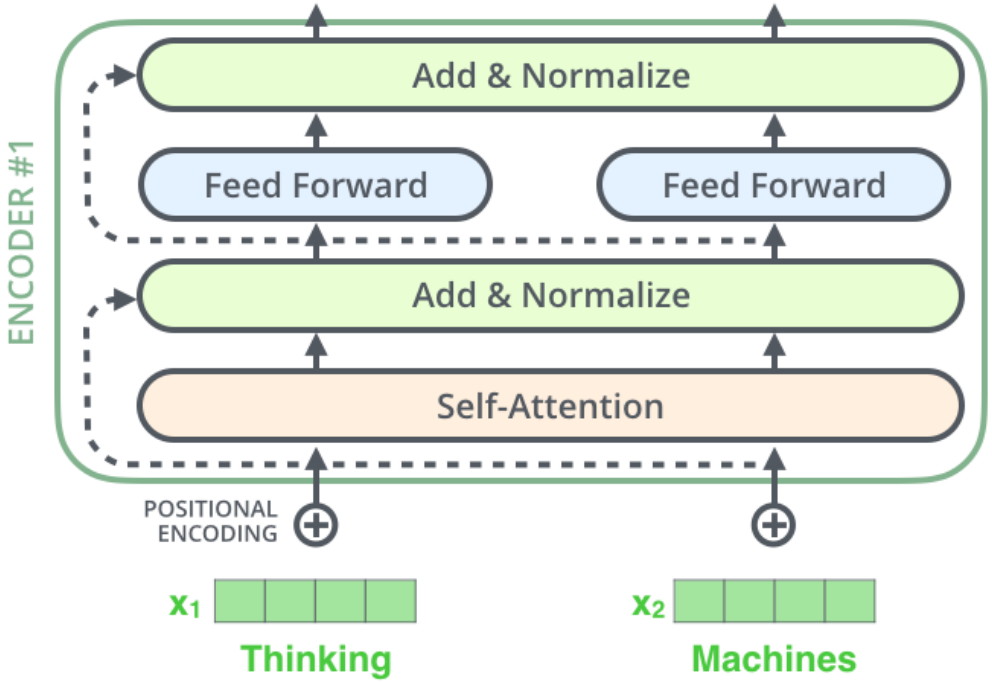
(Ref: Unlocking the Transformer: Advancing Technology From RNNs and LSTMs - Anthony Gavriel)

YHK

Add & Norm

YHK

The Residuals



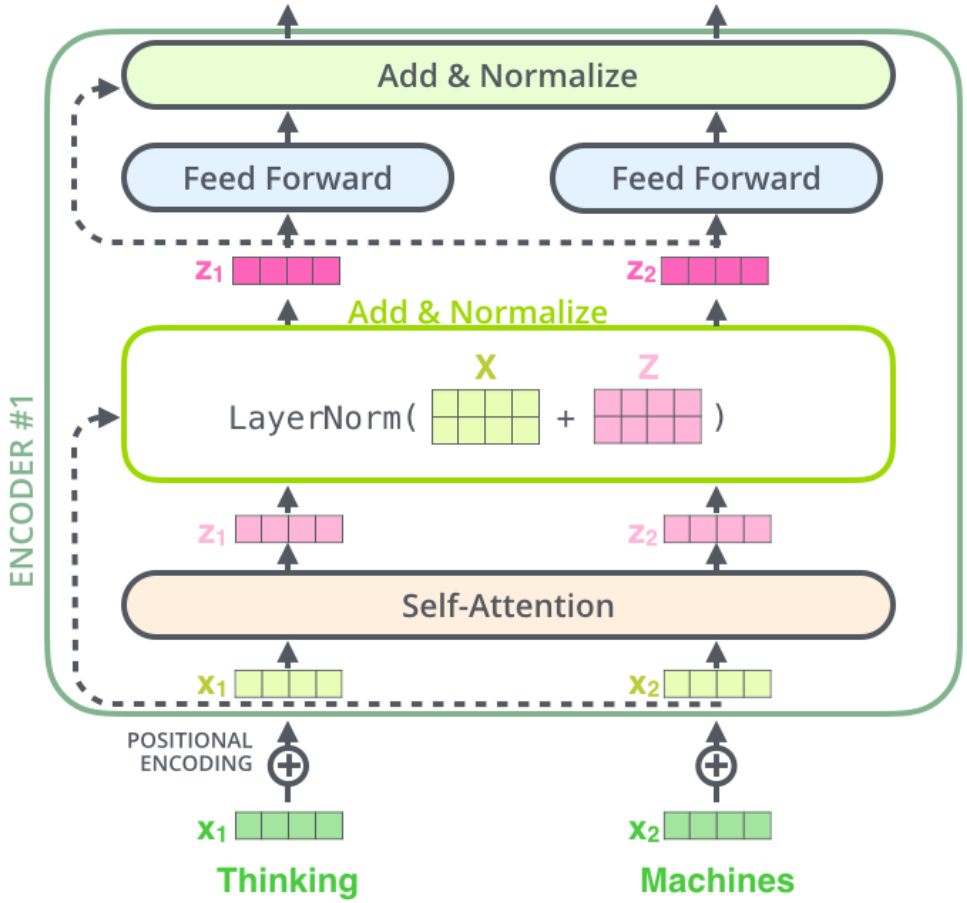
(Ref: "The Illustrated Transformer" - Jay Alammar)

Each sub-layer (self-attention, ffnn) in each encoder has a residual connection around it, and is followed by a layer-normalization step.
The dotted line means, sometimes, the self-attention block wont be considered/skipped and directed next Add/Normalize block is taken up.

YHK

Visualize The Residuals

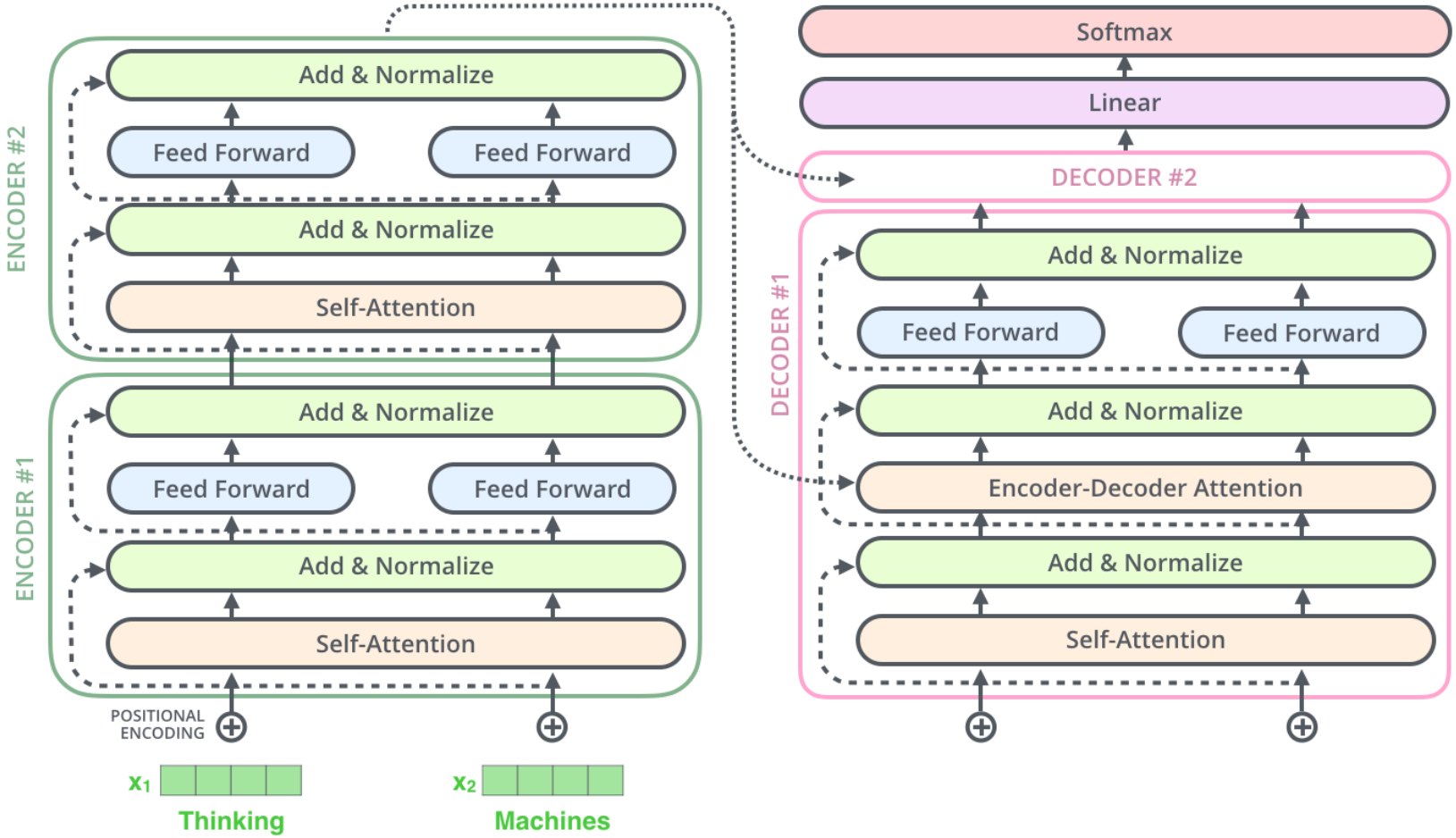
Visualize the vectors and the layer-norm operation associated with self attention, it would look like this:



(Ref: "The Illustrated Transformer" - Jay Alammar)

YHK

Also for the Decoder



(Ref: "The Illustrated Transformer" - Jay Alammar)

Output of Encoders (K and V) are passed to 'Cross Attention' layers of each decoder cell.

YHK

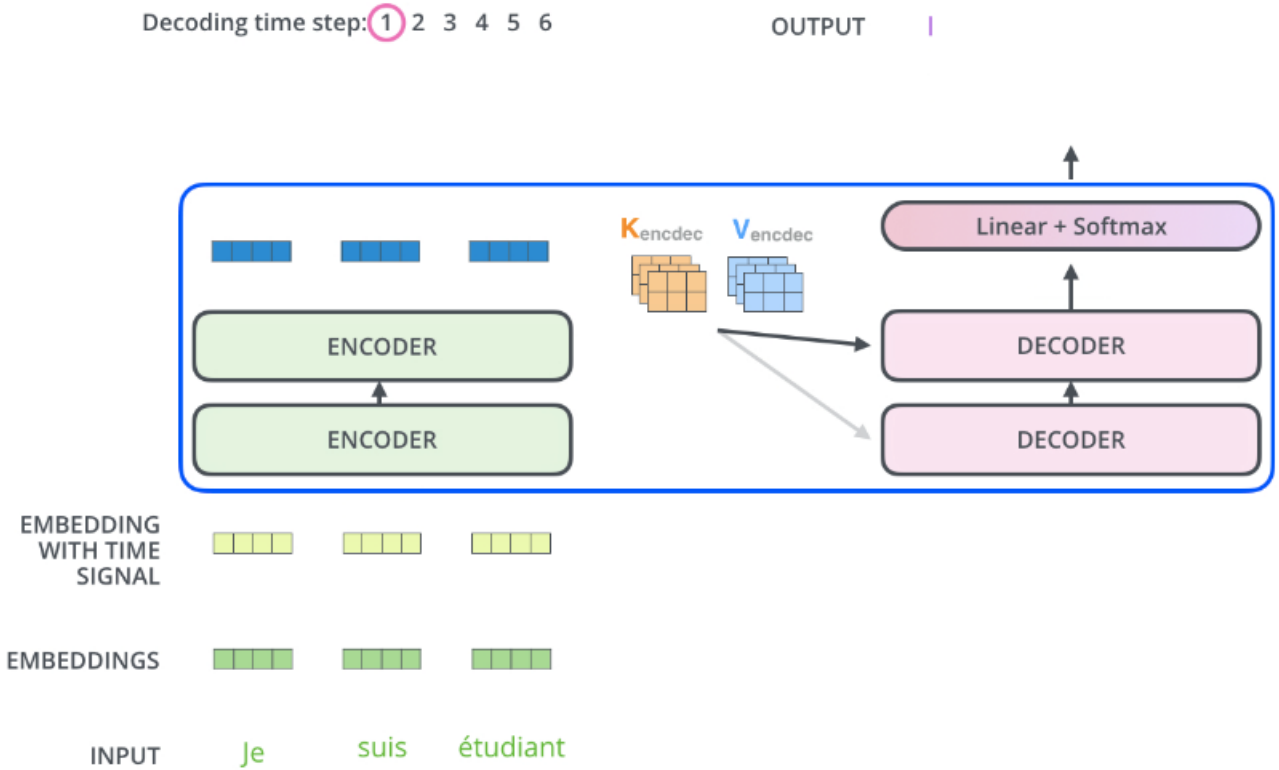
Encoder side is done

- The encoder blocks then produce a final output, encoding. This vector is a rich, condensed representation of the most important information from the input.
- So, ready with latent space, parallel word embedding of the input sentence ie BERT
- The decoder uses self-attention and feed-forward layers, just like the encoder, however it has an additional component, the encoder-decoder attention layer. This layer allows the decoder to focus on significant places from the input sequence.

YHK

Decoder Side

The Decoder Side - Overview

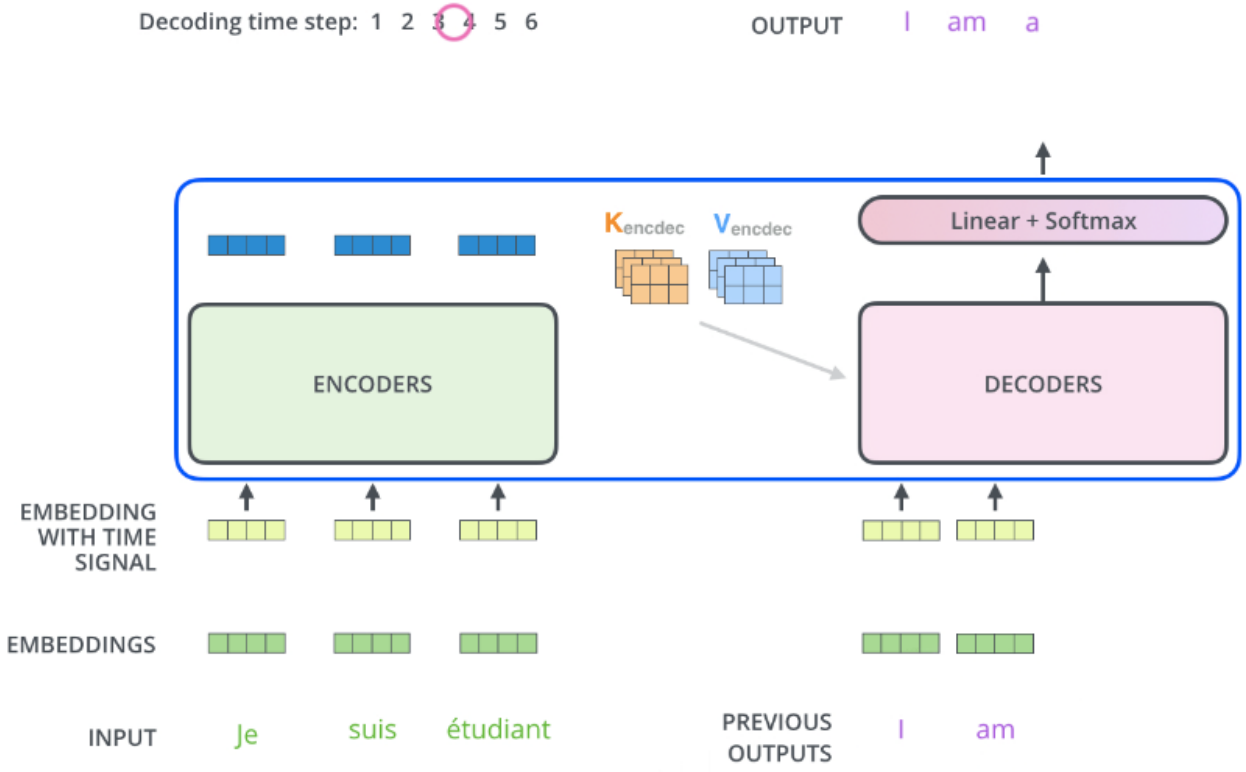


(Ref: "The Illustrated Transformer" - Jay Alammar)

The output of the top encoder is then transformed into a set of attention vectors K and V. These are to be used by each decoder in its "encoder-decoder attention" layer which helps the decoder focus on appropriate places in the input sequence.

YHK

The Decoder Side - Overview



(Ref: "The Illustrated Transformer" - Jay Alammar)

The following steps repeat the process until a special symbol is reached indicating the transformer decoder has completed its output. The output of each step is fed to the bottom decoder in the next time step, and the decoders bubble up their decoding results just like the encoders did.

YHK

The Decoder Side - Overview

- In the decoder, self-attention layers differ from those in the encoder.
  - The self-attention layer in the decoder can only attend to earlier positions in the output sequence.
  - This is achieved by masking future positions (setting them to $-\infty$) before the softmax step in the self-attention calculation.
- The "Encoder-Decoder Attention" layer functions similarly to multiheaded self-attention.
  - It generates its Queries matrix from the layer below it.
  - The Keys and Values matrix are taken from the output of the encoder stack.

YHK

Output Embedding plus Positional Encoding

YHK

Masked Attention

YHK

Masked Attention Need

- We need masking to make the training parallel. And the parallelization is good as it allows the model to train faster.
- Example: to translate “I love you” to German
- The encoder works in parallel mode, takes the input sequence (”I love you”) and produces the numbers 11, 12, 13 as the vector representations of the input sequence.
- During the training we know that the translation should be “Ich liebe dich” with corresponding expected vector representations as 21, 22, 23.

(Ref https://stackoverflow.com/questions/58127059/how-to-understand-masked-multi-head-attention-in-transformer)

YHK

Masked Attention Need

As we actually know the expected outputs we can adjust the process and make it parallel. There's no need to wait for the previous step output.

- Parallel operation #A. Inputs: 11, 12, 13. Trying to predict 21.
- Parallel operation #B. Inputs: 11, 12, 13, and also 21. Trying to predict 22.
- Parallel operation #C. Inputs: 11, 12, 13, and also 21, 22. Trying to predict 23.

All input-output pairs are ready.

(Ref https://stackoverflow.com/questions/58127059/how-to-understand-masked-multi-head-attention-in-transformer)

YHK

Masked Attention Need

Now we can clearly see why masking is done.

- Say for Parallel operation #A. Inputs: 11, 12, 13. As it is trying to predict 21, it should not see any of 21, 22, 23, right? so all are masked.
- For Parallel operation #B. Inputs: 11, 12, 13, and also 21. Trying to predict 22. Here we need 21, so thats unmasked, but 22, 23 are masked, as they to be predicted one by one. So here it hides 2nd and 3rd outputs.
- For Parallel operation #C. Inputs: 11, 12, 13, and also 21, 22. Trying to predict 23. It hides 3rd output.

(Ref https://stackoverflow.com/questions/58127059/how-to-understand-masked-multi-head-attention-in-transformer)

YHK

Masked Attention Need

- Implementation is inside of scaled dot-product attention by masking out (setting to ∞) all values in the input of the softmax which correspond to illegal connections.
- Note: during the inference (not training) the decoder works in the sequential (not parallel) mode as it doesn't know the output sequence initially. But it's different from RNN approach as Transformer inference still uses self-attention and looks at all previous outputs

(Ref https://stackoverflow.com/questions/58127059/how-to-understand-masked-multi-head-attention-in-transformer)

YHK

Masked Multi-head Attention

Linear plus Softmax

The Final Linear and Softmax Layer

- The decoder stack produces a vector of floats as its output.
- How do we turn that into a word? That's the job of the final Linear layer which is followed by a Softmax Layer.
- The Linear layer is a fully connected neural network projecting the decoder stack's output vector into a much larger vector known as a logits vector.
  - Assuming a model vocabulary of 10,000 unique English words, the logits vector has 10,000 cells, each corresponding to the score of a unique word.
- The softmax layer transforms these scores into probabilities, ensuring they are all positive and sum up to 1.0.
  - The cell with the highest probability determines the chosen word, which serves as the output for the current time step.
- Given a final vector from the neural network,
  - This vector is expanded to a vocabulary-long vecto via $m \times n$ neural network.
  - Output is through a softmax function, causing the values to be compressed to the range of 0 to 1.
  - Each value can be interpreted as a probability.
- The index in the vocabulary vector with the highest probability determines the output word.

YHK

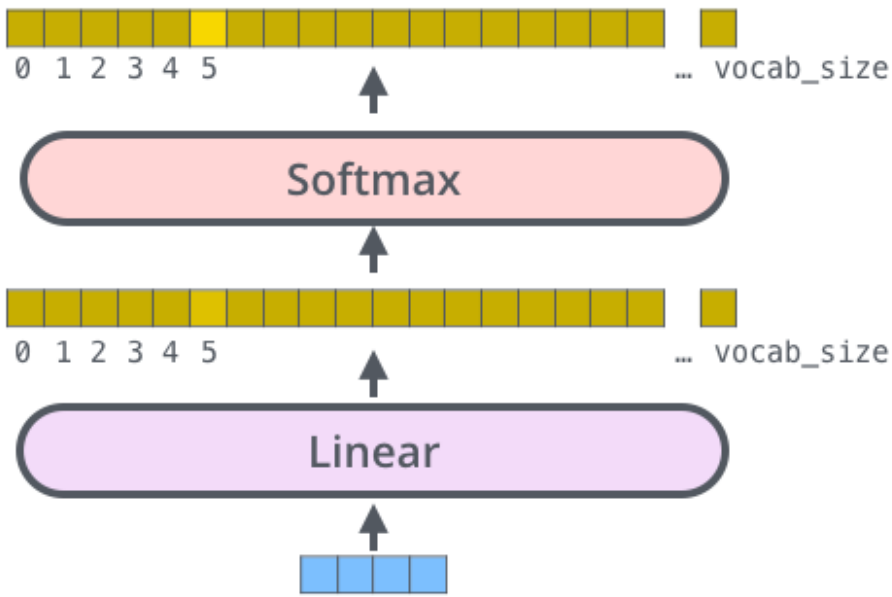
The Final Linear and Softmax Layer

Which word in our vocabulary is associated with this index?

am

Get the index of the cell with the highest value (**argmax**)

5



(Ref: "The Illustrated Transformer" - Jay Alammar)

Backpropogation during training

Pre-training workflow

Output Vocabulary

WORD	a	am	I	thanks	student	<eos>
INDEX	0	1	2	3	4	5

One-hot encoding of the word “am”

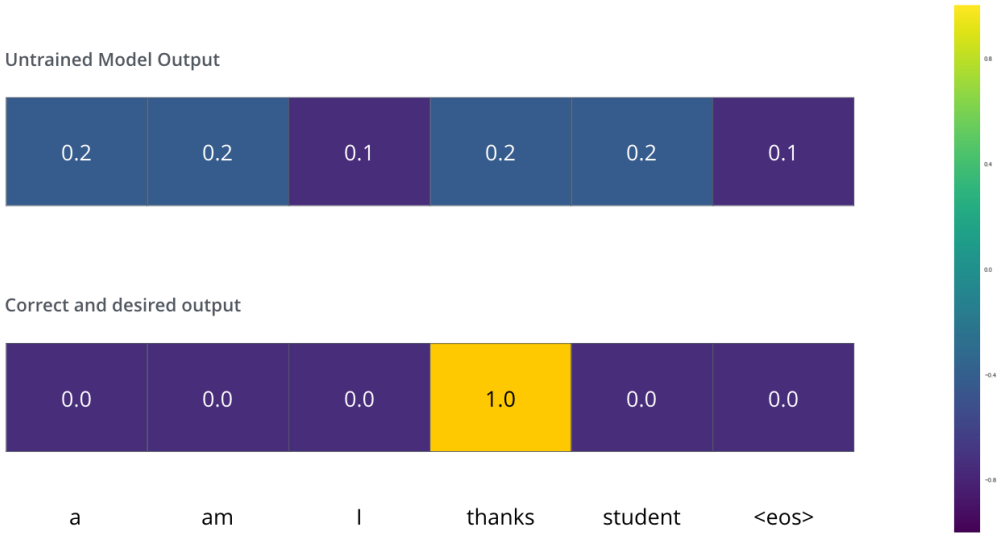
0.0	1.0	0.0	0.0	0.0	0.0

(Ref: “The Illustrated Transformer” - Jay Alammar)

- During training, an untrained model would go through the exact same forward pass.
- But we have labeled dataset, ie inputs as well as output sentences.
- Say, vocab is of 6 words and using one hot encoding.
- Assume our training sentences have only one pair, ’merci’ and ’thanks’

YHK

The Loss Function



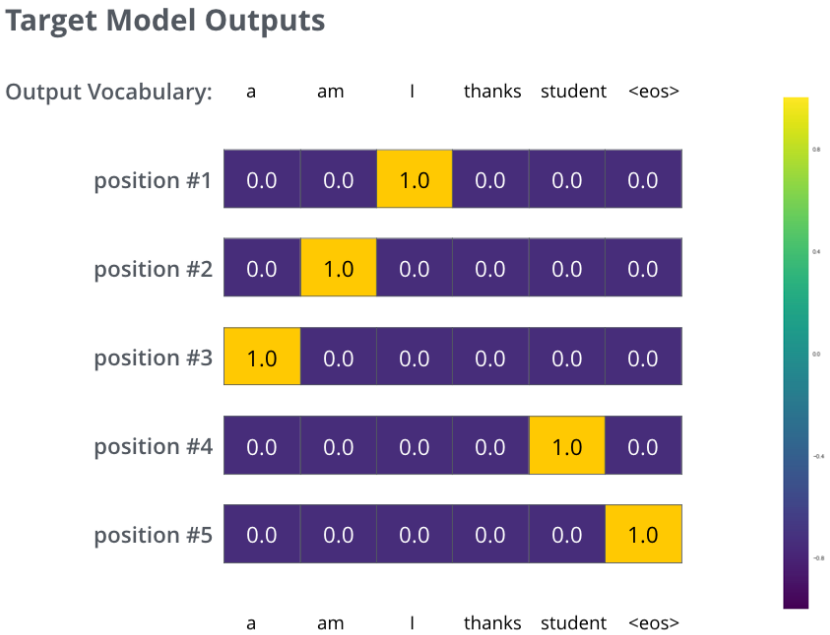
(Ref: "The Illustrated Transformer" - Jay Alammar)

- Goal: For "merci", we want the output to be a probability distribution indicating the word "thanks".
- But since this model is not yet trained (meaning all weight matrices have random values), that's unlikely to happen just yet.
- How do you compare two probability distributions?
- Say, we simply subtract one from the other. (For more details, look at cross-entropy and Kullback–Leibler divergence.)

YHK

The Loss Function - More complex example

For example – input: "je suis etudiant" and expected output: "i am a student". We want our model to successively output probability distributions where:

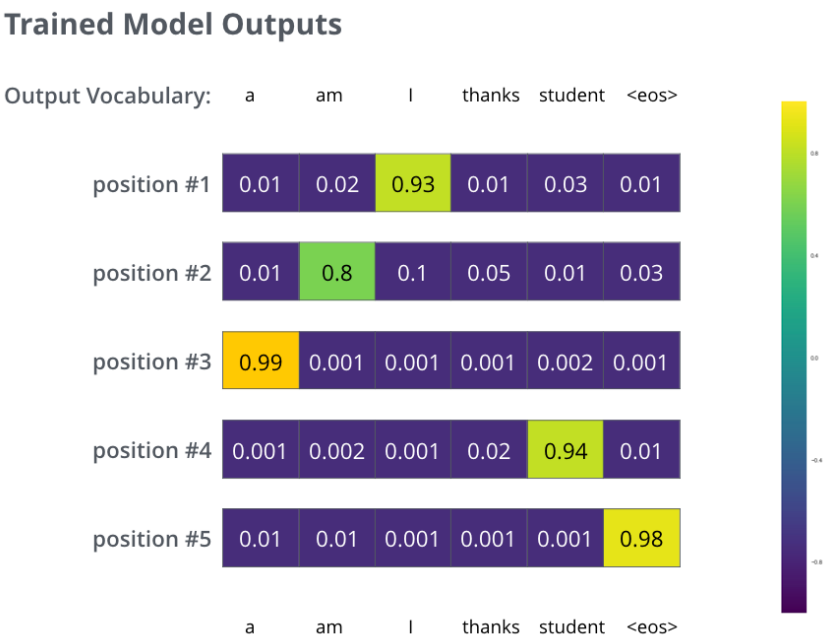


(Ref: "The Illustrated Transformer" - Jay Alammar)

- Each probability distribution is represented by a vector of width vocab_size (6 in our toy example, but more realistically a number like 30,000 or 50,000)
- The first probability distribution has the highest probability at the cell associated with the word "i"
- The second probability distribution has the highest probability at the cell associated with the word "am"
- And so on, until the fifth output distribution indicates $< EOS >$ symbol, which also has a cell associated with it from the 10,000 element vocabulary.

YHK

The Loss Function - More complex example



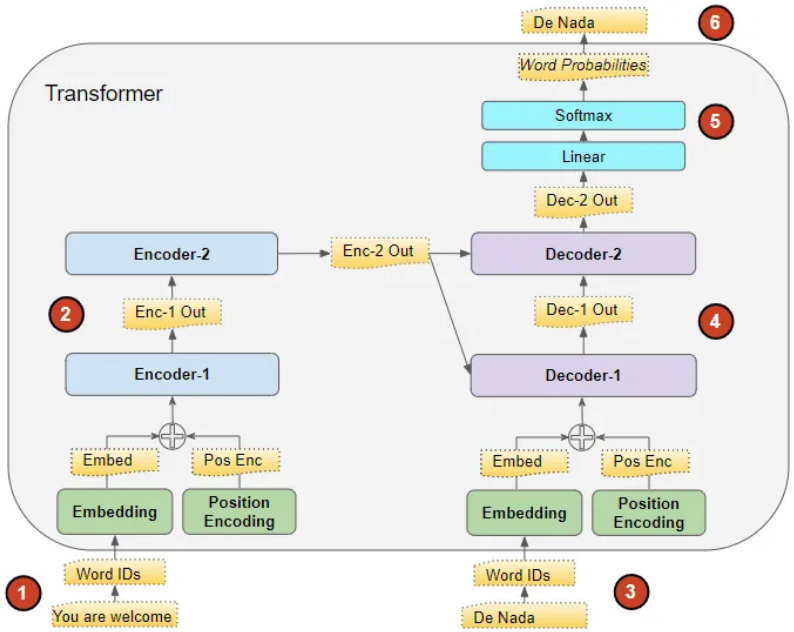
(Ref: "The Illustrated Transformer" - Jay Alammar)

We repeat this for positions #2 and #3. . . etc. This method is called "beam search"

- Now, because the model produces the outputs one at a time, we can assume that the model is selecting the word with the highest probability from that probability distribution and throwing away the rest (greedy decoding)
- Another way, take top 2, (say, 'I' and 'a' for example), then in the next step, run the model twice: once assuming the first output position was the word 'I', and another time assuming the first output position was the word 'a', and whichever version produced less error considering both positions #1 and #2 is kept.

YHK

Pre-training workflow Summary

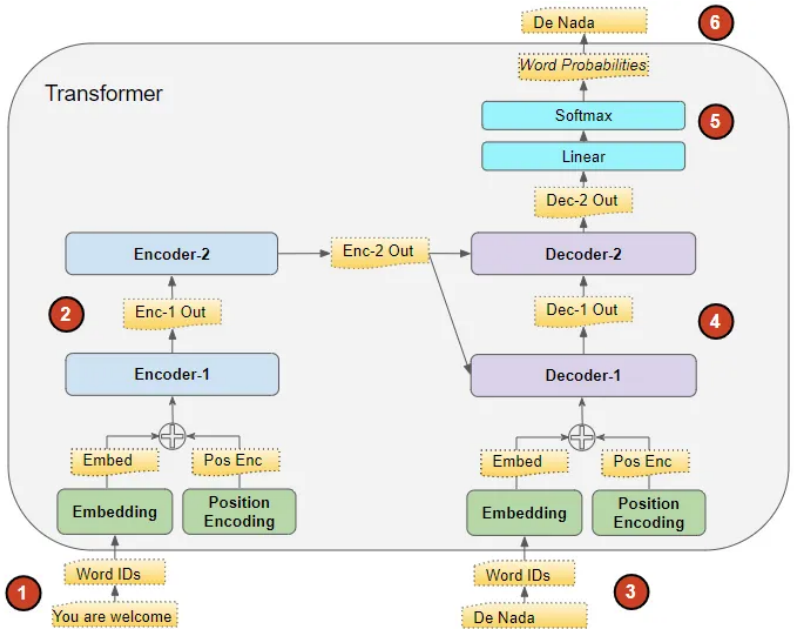


(Ref: "Transformers Explained Visually (Part 1): Overview of Functionality" - Ketan Doshi)

- The input sequence is converted into Embeddings (with Position Encoding) and fed to the Encoder.
- The stack of Encoders processes this and produces an encoded representation of the input sequence.
- The target sequence is prepended with a start-of-sentence token, converted into Embeddings (with Position Encoding), and fed to the Decoder.

YHK

Pre-training workflow Summary

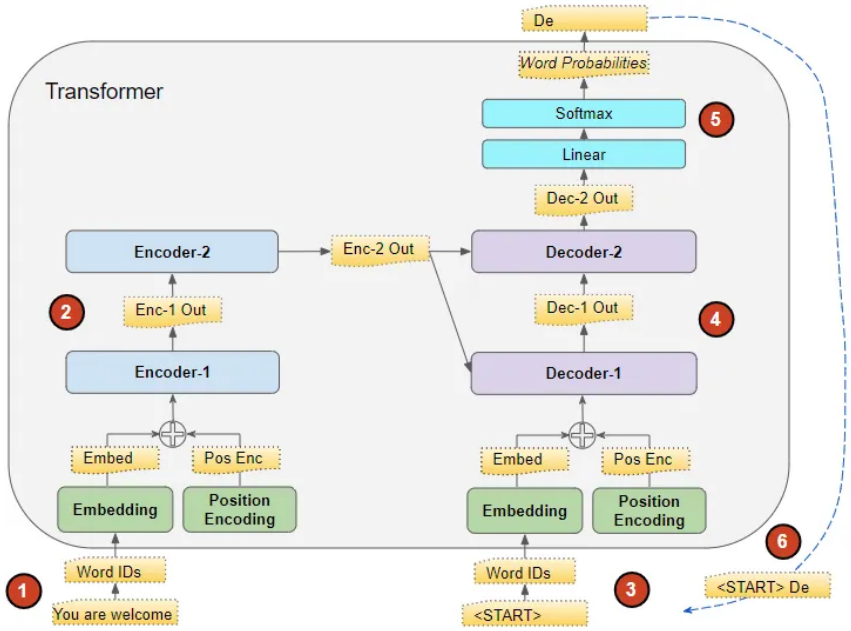


(Ref: "Transformers Explained Visually (Part 1): Overview of Functionality" - Ketan Doshi)

- The stack of Decoders processes this along with the Encoder stack's encoded representation to produce an encoded representation of the target sequence.
- The Output layer converts it into word probabilities and the final output sequence.
- The Transformer's Loss function compares this output sequence with the target sequence from the training data. This loss is used to generate gradients to train the Transformer during back-propagation

YHK

Inference workflow Summary

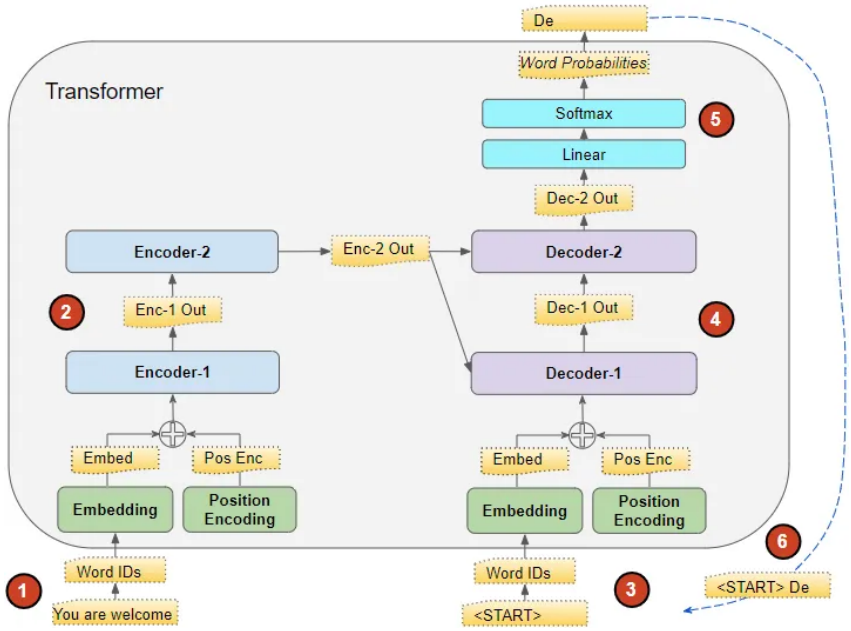


(Ref: "Transformers Explained Visually (Part 1): Overview of Functionality" - Ketan Doshi)

- The input sequence is converted into Embeddings (with Position Encoding) and fed to the Encoder.
- The stack of Encoders processes this and produces an encoded representation of the input sequence.
- Instead of the target sequence, we use an empty sequence with only a start-of-sentence token. This is converted into Embeddings (with Position Encoding) and fed to the Decoder.
- The stack of Decoders processes this along with the Encoder stack's encoded representation to produce an encoded representation of the target sequence.

YHK

Inference workflow Summary



(Ref: "Transformers Explained Visually (Part 1): Overview of Functionality" - Ketan Doshi)

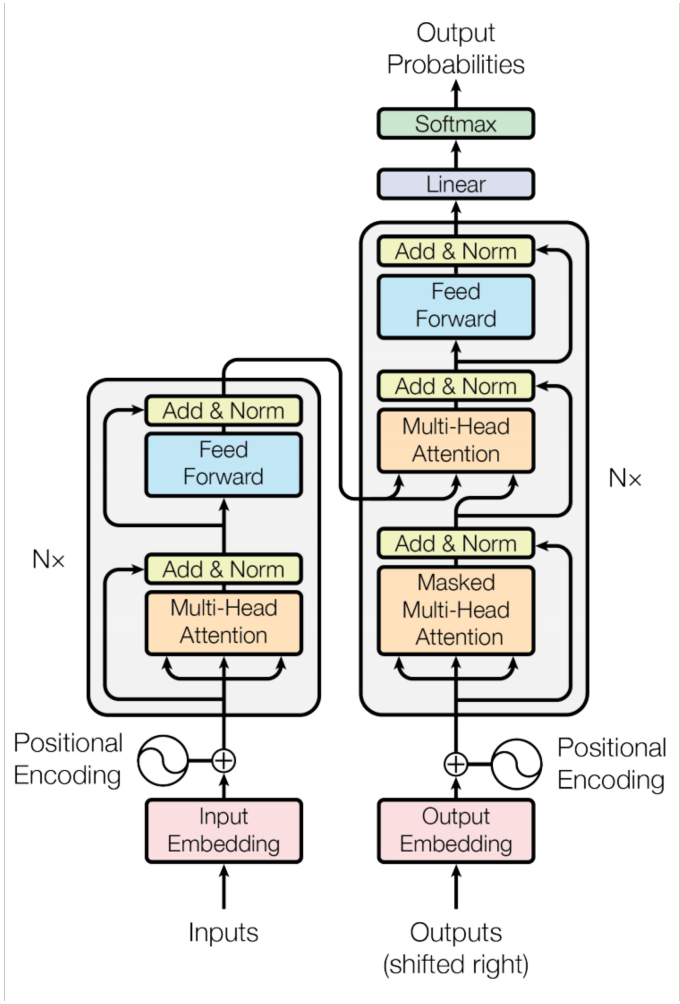
Note that since the Encoder sequence does not change for each iteration, we do not have to repeat steps #1 and #2 each time.

- The Output layer converts it into word probabilities and produces an output sequence.
- We take the last word of the output sequence as the predicted word. That word is now filled into the second position of our Decoder input sequence, which now contains a start-of-sentence token and the first word.
- Go back to step #3. As before, feed the new Decoder sequence into the model. Then take the second word of the output and append it to the Decoder sequence. Repeat this until it predicts an end-of-sentence token.

YHK

Conclusion

Transformer Overview

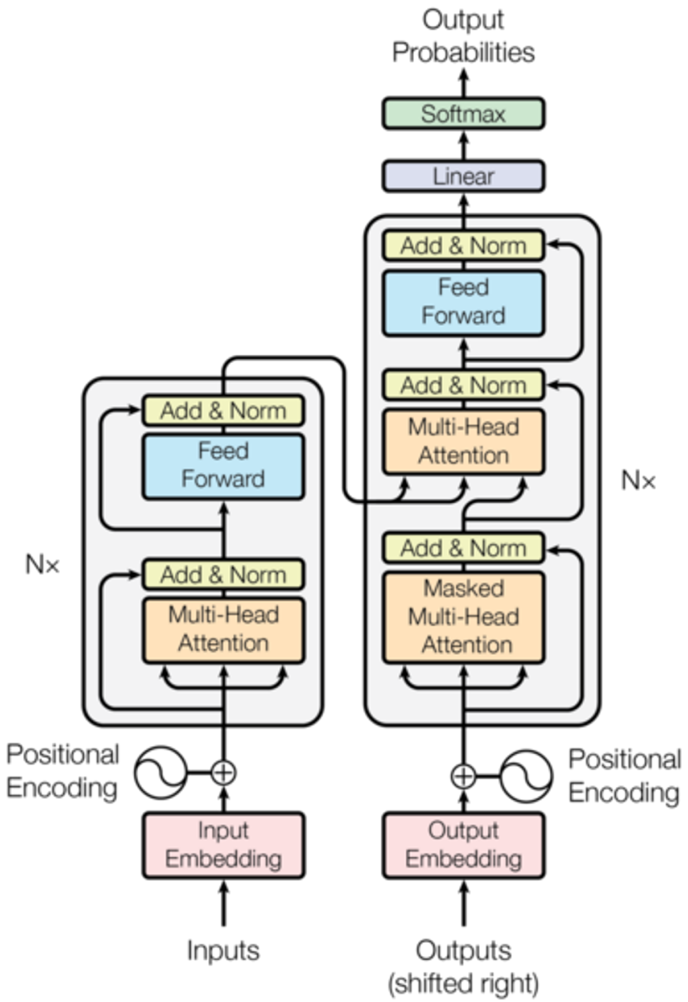


Attention is all you need. 2017.
Aswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser, Polosukhin
https://arxiv.org/pdf/1706.03762.pdf

- Non-recurrent sequence-to-sequence encoder-decoder model
- Task: machine translation with parallel corpus
- Predict each translated word
- Final cost/error function is standard cross-entropy error on top of a softmax classifier

YHK

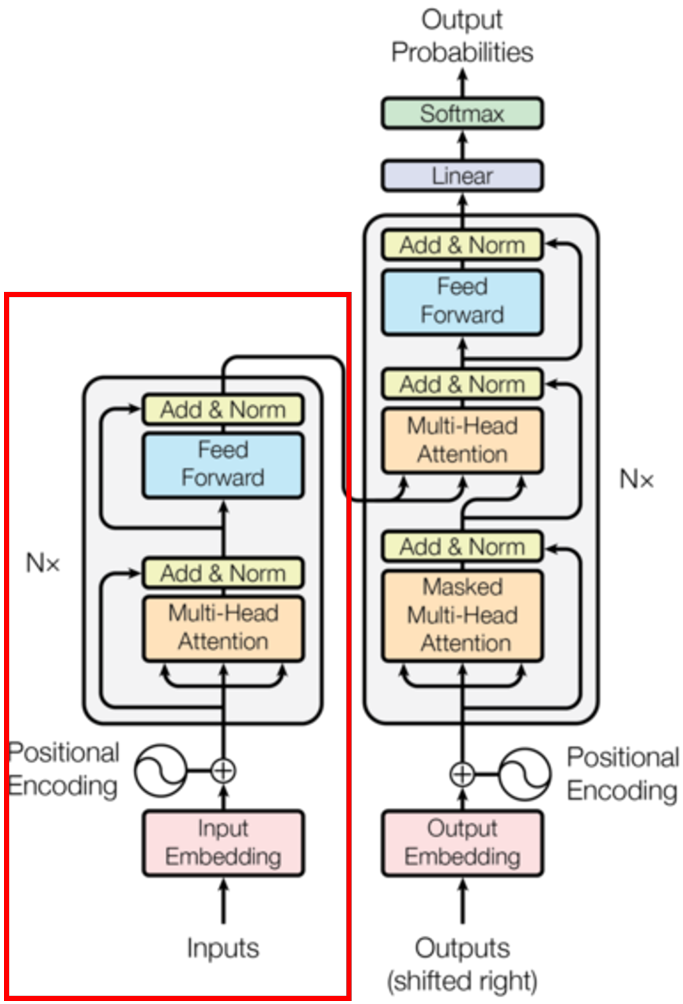
Architecture



- Encoder-Decoder was there
- New?: Multi head attention
- Fine tuned with Custom Data
- Flavors:
  - Encoder only
  - Decoder only
  - Encoder-Decoder

YHK

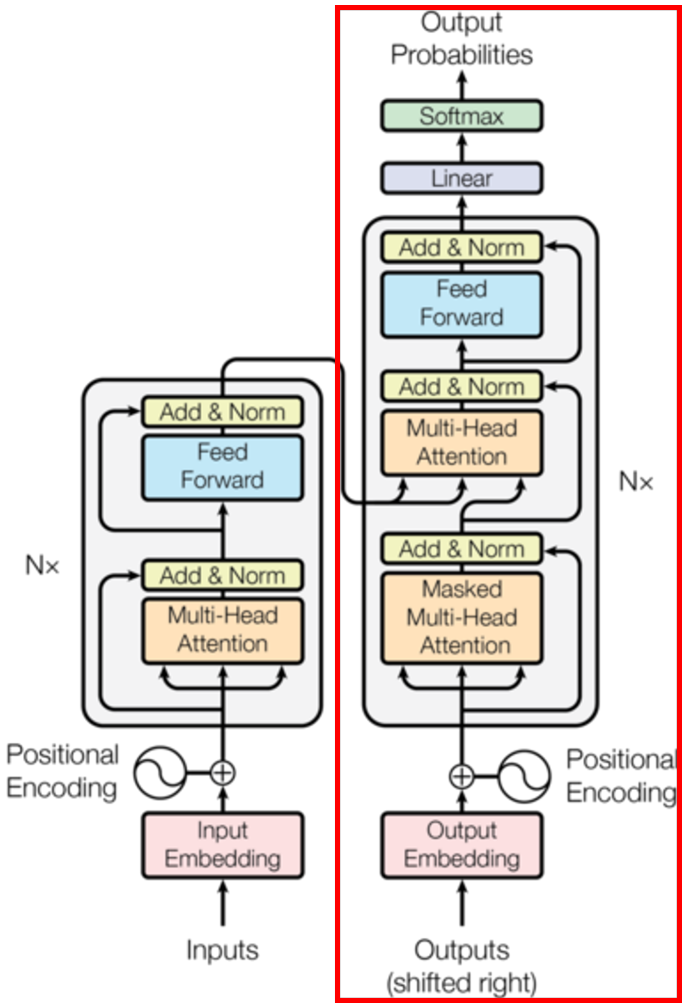
Encoder only



- If we are only interested in training a language model for the input for some other tasks, then we do not need the decoder of the transformer.
- Pre-trained by predicting masked word
- BERT, XLNet, DistillBERT, RoBERTA
- Usage:
  - Text Classification
  - Named Entity Recognition
  - Extractive Question Answering

YHK

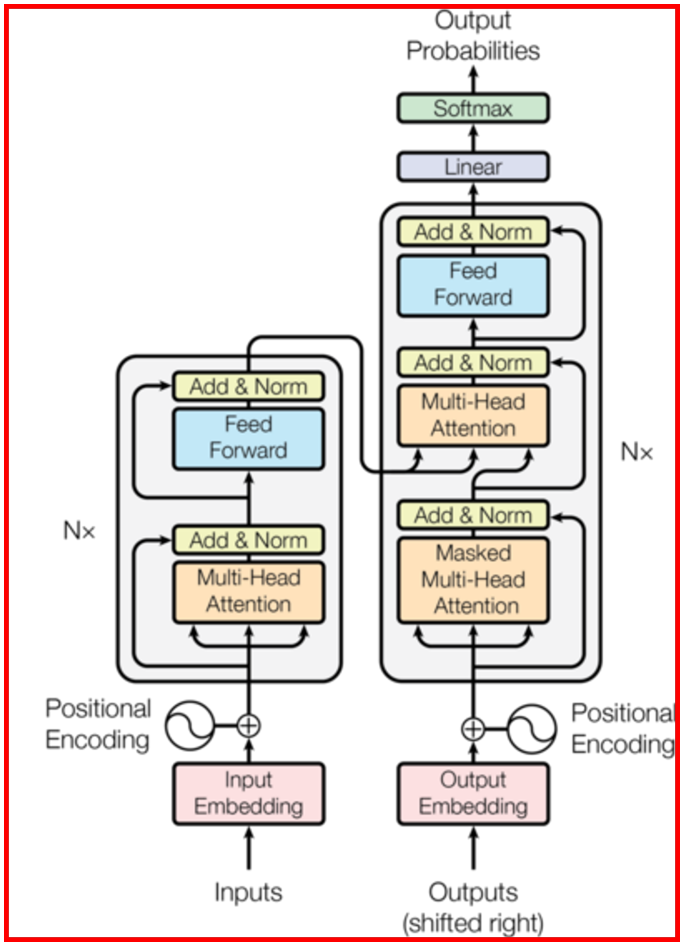
Decoder only



- If we do not have input, we just want to model the "next word", we can get rid of the encoder side of a transformer and output "next word" one by one.
- Pre-trained by predicting next word
- GPT-*, Transformer XL
- Usage:
  - Text Generation

YHK

Encoder-Decoder, both



- Pre-trained by Seq2Seq manner
- T5, BART, PEGASUS
- Usage:
  - Text summarization
  - Machine Translation
  - SQL generation

YHK

Conclusion

Main innovations in Transformers:

- Positional Encoding: instead of architecture relying on sequence order, like RNNs, the Transformers store the position, making them order independent architecturally as the position is part of th data itself so that you can then have parallel processing
- Self-attention: Weights vector, how much each word contributes, learnt by seeing many pairs, during training. Self, meaning within itself. Internal structure understanding, Encoder only.

YHK

References

References

- The Illustrated Transformer – Jay Alammar
- “I never knew Sentence Transformers could be so useful!” - Pradip Nichite
- The Annotated Transformer – Sasha Rush
- The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning) – Jay Alammar
- Attention is all you need: Discovering the Transformer paper - Eduardo Munoz, has implementation
- Attention is all you need (Transformer) - Model explanation (including math), Inference and Training - Umar Jamil

YHK

Thanks ...

- Search **"Yogesh Haribhau Kulkarni"** on Google and follow me on LinkedIn and Medium
- Office Hours: Saturdays, 2 to 5pm (IST); Free-Open to all; email for appointment.
- Email: yogeshkulkarni at yahoo dot com

(Generated by Hugging Face QR-code-AI-art-generator, with prompt as "Follow me")

YHK