Transformer

Content

- Introduction to Transformer
  - Transformer Model Architecture
  - Transformer’s Encoder Decoder
  - Attention Mechanism
  - Self-Attention
  - Multi-Head Attention & Masked Multi-Head Attention
- Transformer Essential Libraries: Hugging face
- BERT (*Self Read)

Introduction to Transformer

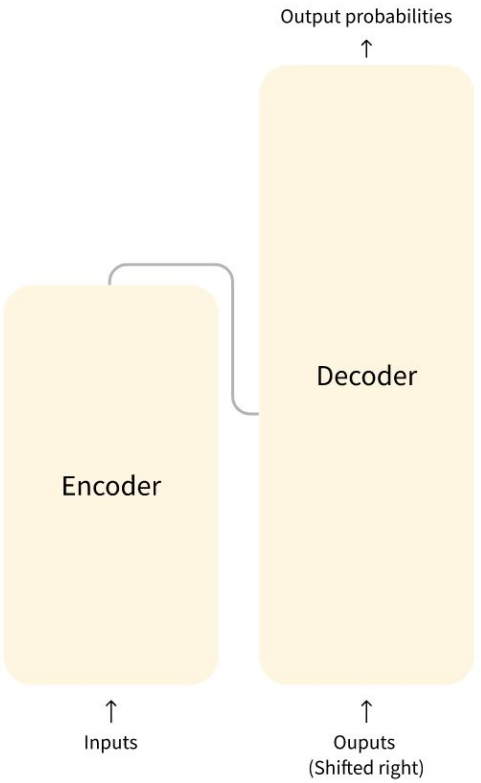
Transformer Background

- Introduced in 2017 paper titled “Attention is all you need”
- Designed to overcome the limitations of recurrent architectures such as RNN and LSTM
- The major limitation of RNN, LSTM being their inability to learn long term dependencies in a given sequence
- Transformer architecture introduced the concept of Self-Attention to represent and learn long term dependencies

Transformer Background

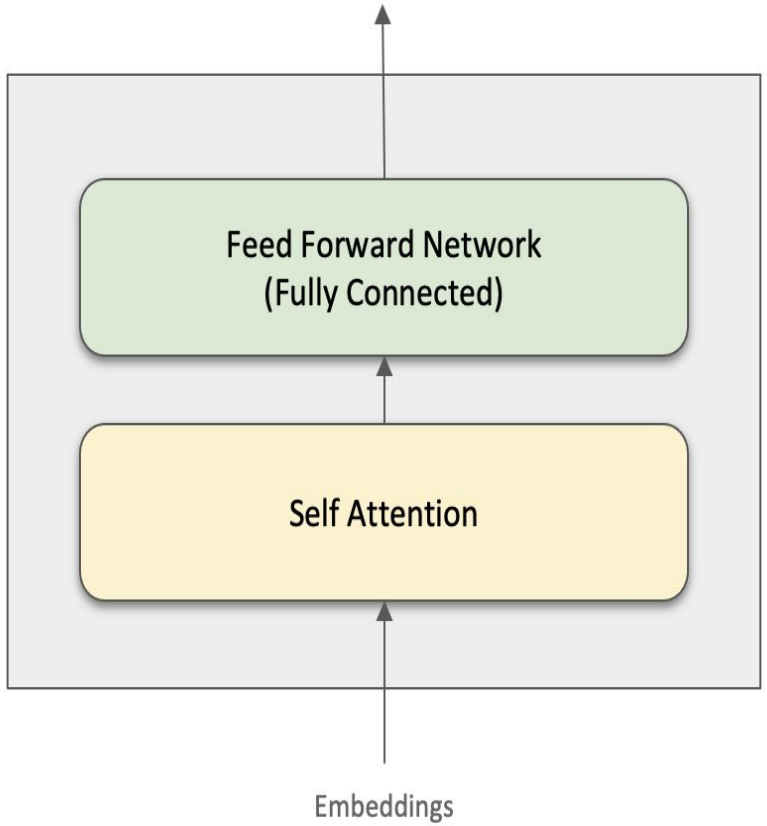
- Primarily designed to tackle language translation problem

- **The architecture is primarily composed of two blocks: Encoder & Decoder**



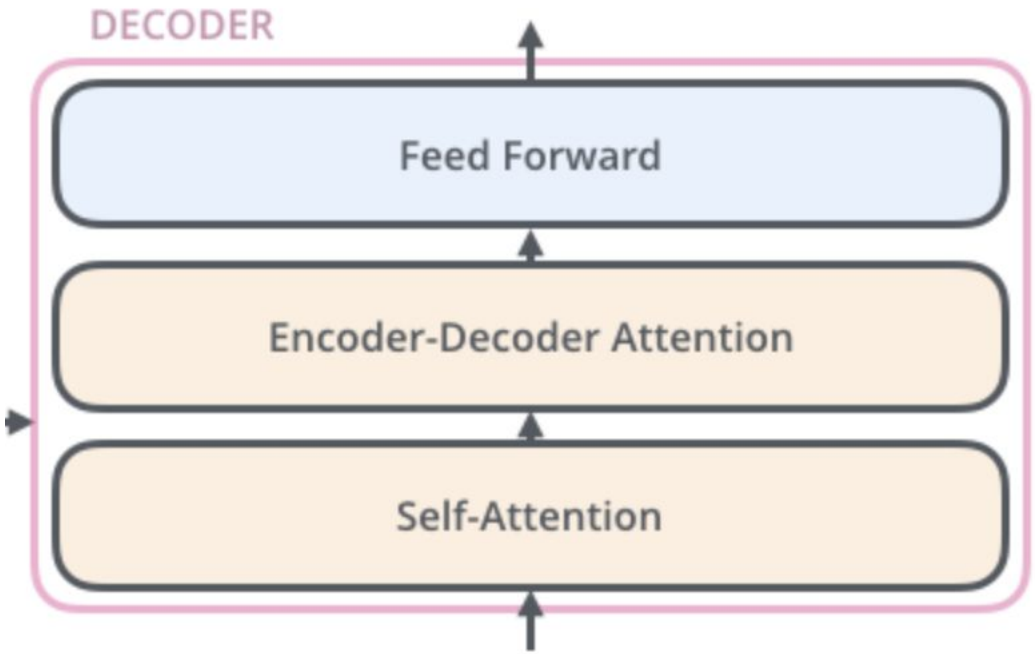
Encoder Block

- Encoder has two subblocks ‘Self attention’ and ‘Feed forward neural network’.
  - Self attention - helps the encoder incorporate the influence other words in the input sentence as it encodes a specific word.
  - Feed forward neural network - A fully connected neural network
- Multiple identical sequential encoder block shall exist. (original transformer paper proposed sequence of 6 encoders)
- All the words of sentence passed together.(no timestamp-based sequence of words like RNN)



Decoder Block

- Encoder input passes to Decoder.
- Decoder has independent input also, that is translated meaning of word in output language.
- This independent input passed in sequence (first word, then first two words, then first three words and son on) why ?
- Decoder architecture components are almost same as encoder, it has one additional layer attention layer (“Encoder-decoder attention”)
- Decoder final output is probability of o/p words



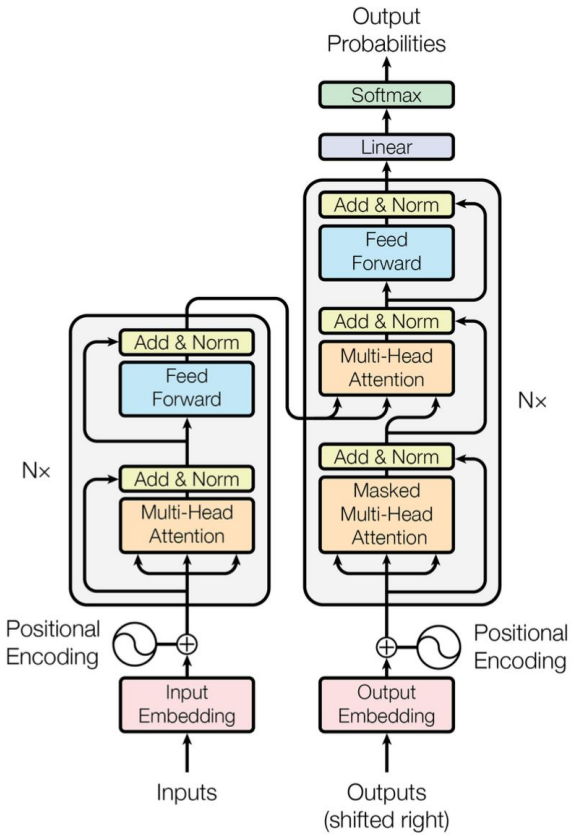
Depending on the task, Transformers blocks can be used independently or together

- **Encoder-only models :** Suitable for tasks that require understanding of the input(full sentence), such as, sentence classification, named entity recognition

- **Decoder-only models**: Good for generative tasks such as text generation.

- **Encoder-decoder models** or **sequence-to-sequence models**: Good for generative tasks that require an input, such as translation or summarization.

The Transformer **original** model Architecture

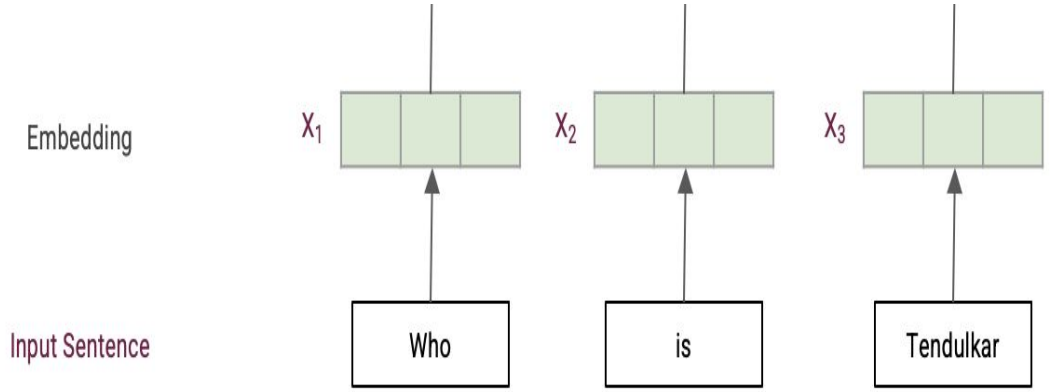


1. Word/Input embedding
2. Positional Encoding
3. Attention (Multi Head /Masked Multi head)
4. Skip connections
5. Add & Normalization layer
6. Feed forward network for encoding
7. Encoder Decoder Attention

* Diagram From Attention is all you need paper

Input Embedding

- Convert each input word into a vector using an embedding algorithm.
- Original paper uses Word2Vec embedding (a Google's product)
- All words of sentence are embedded and forwarded to next layer together (no recurrence and and no convolution)
- Original paper uses 512 as dimensionality of vector



Positional Encoding

- Enable each word, carry information about its position
- Provides/add the model with information about the positions of words in the input sequence.
- A positional vector is added to the embedded vector
- Original paper uses shown functions/formula for positional encoding (however other functions are also possible)
- Here, pos is the position and i is the dimension
- Positional vector are fixed and not learned

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

Attention Mechanism/Layer

- Transformer models introduces special layer “ Attention” (Multi head attention & Masked Multi head attention)

- On a granular level, attention layer instructs the model to focus on certain words in a sentence you passed (as a word by itself has a meaning, but that meaning is strongly influenced by the context of surrounding words)

Self Attention

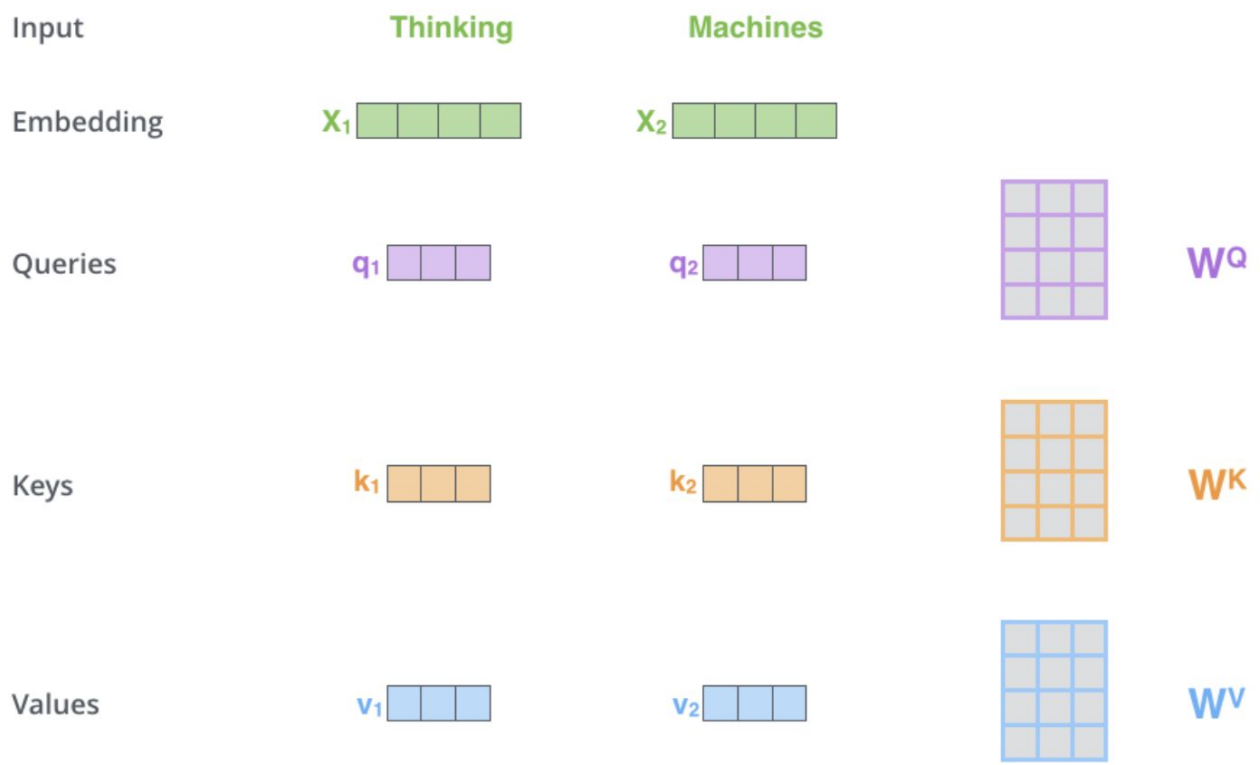
- Understanding how important a word is from its own frame of reference in the context of other words in the sentence, is “Self Attention”.

- It is a method of evaluating the degree of dependency of each word on other words in a sentence.

Computing Self Attention: Step 1 - Q,K, V generation

- Three weights - Wq, Wk, Wv are defined. For first step their values are initialize randomly and for successive iteration its value are adjusted using back propagation i.e. through training process
- Original paper uses weight matrix size as 512 * 64
- Input each word vector is multiplied with these weights that results into three vector Q,K and V called as Query, Key and value respectively.
- For each word of the input sentence three vectors -Qi, Ki, Vi is generated .Here I denotes the position of word
- In the original paper dimension of Q,K,V vector is 64 (512* (521*64)) . Note that these new vectors (Q,K,V) are smaller in dimension than the embedding vector.
- Purpose is to make the computation of multiheaded attention (mostly) constant.

Computing Self Attention: Step 1 - Illustration

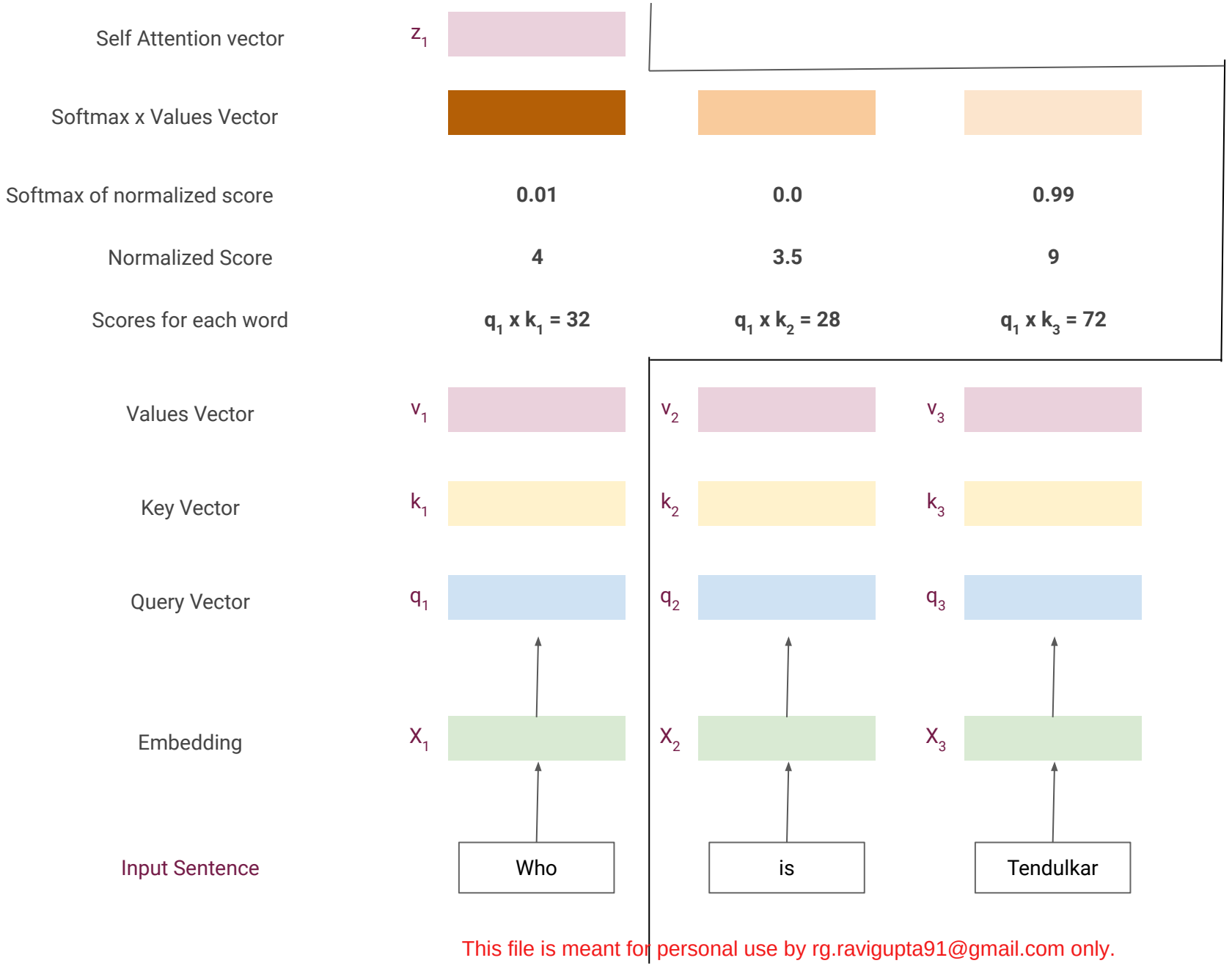


Multiplying x1 by the WQ weight matrix produces q1, the "query" vector associated with that word. We end up creating a "query", a "key", and a "value" projection of each word in the input sentence.

*Image: http://jalammar.github.io/illustrated-transformer/

Computing Self Attention: Step 2 : calculate score

- Score each word of the input sentence against all the words of the sentence.
- For example, for 1st position word - Q1*K1,Q1*K2, Q1* K3,….,Q1*Kn.
- First, dot product of Q and K is calculated
- Second, the obtained value is divided by square root of the dimension of the key vectors. (in original paper its 8)
- The output is then pass through a SoftMax operation.
- SoftMax score determines how much each word will be expressed at this position
- Next **step** is to multiply each value vector by the softmax score, to keep intact the values of the word(s) we want to focus on, and drown-out irrelevant words
- Next step is **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)



Self Attention vector for 1st word is sum of all the vectors in the previous step

Values for each word are multiplied with respective softmax score

Softmax provides relative importance of each word for 1st word

Scores are divided by square root of length of key vector e.g 8 (square root of 64)

Dot product of query vector of 1st word and key vector of each word in the sentence.

Calculating Self Attention for 1st word

Computing Self Attention: Step 2 :Illustration

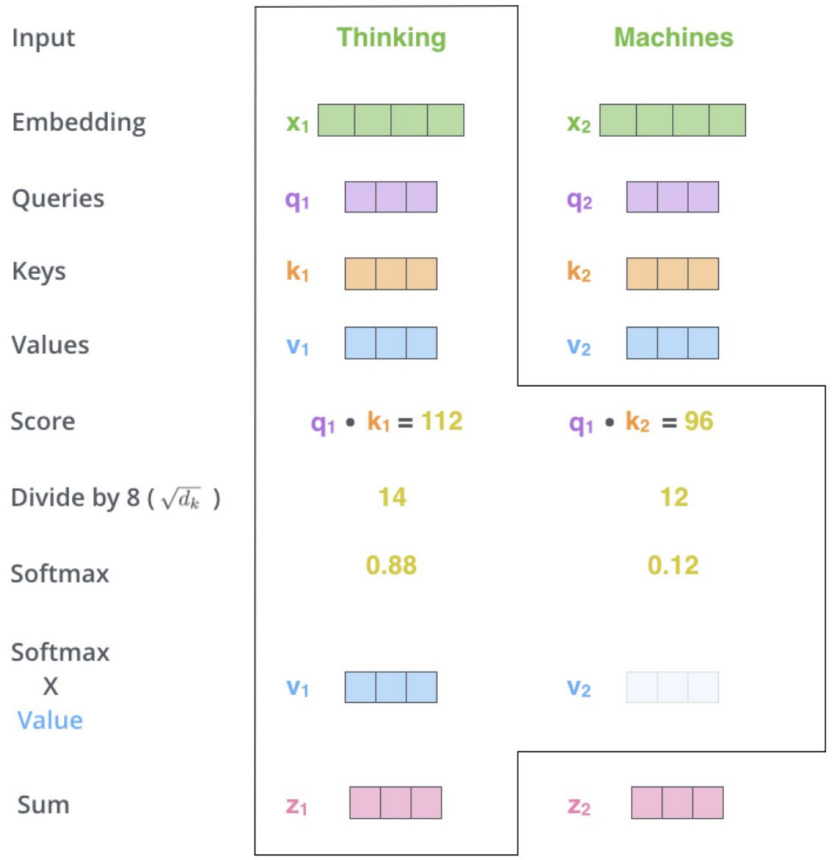


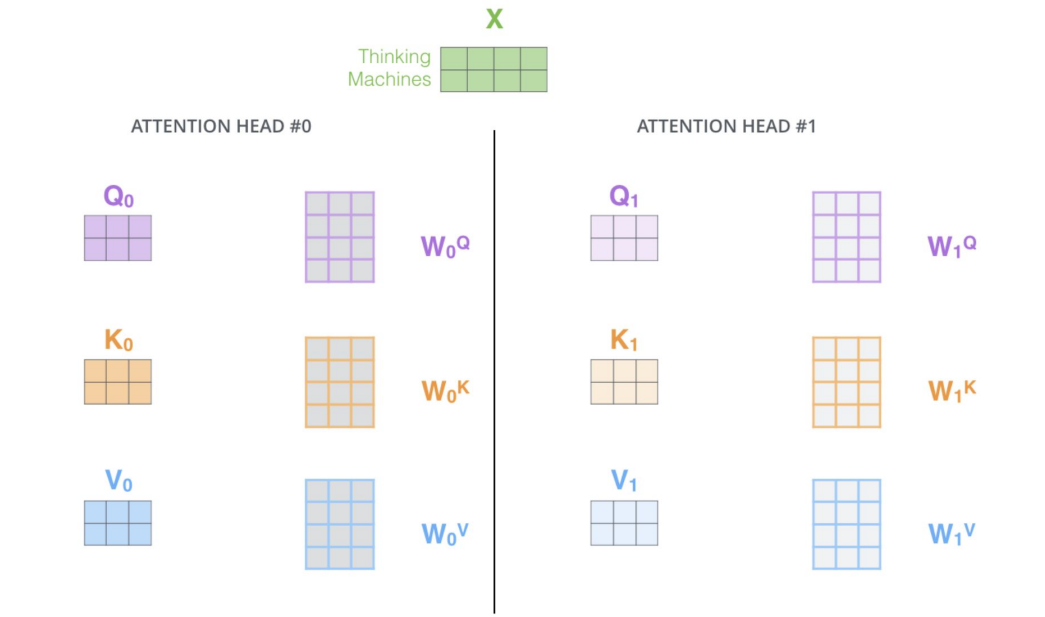
*Image: http://jalammar.github.io/illustrated-transformer/

The self-attention calculation limitations

- In self attention the output z1 is dominated by the word itself
- This limits ability to focus or put emphasis on other word
- Solution is Multi-head Attention

Multi Head Attention

- Instead of single set of 3 weight (Wq, Wk, Wv) , multiple set of weights are used which further results into multiple set of Q,K,V (Query/Key/Value) metrics generation for each word.
- This improves the performance of the attention layer by
  - ✔ Expanding the model's ability to focus on different positions
  - ✔ Giving the attention layer multiple "representation subspaces"
- Original paper use 8 heads
- For each word there are 8 output ,however the subsequent feed forward layer is expecting single input So we need a way to condense these eight down into a single matrix.
- To condense final output, it concatenate the matrices then multiply them by an additional weights matrix Wo.



With multi-headed attention, we maintain separate Q/K/V weight matrices for each head resulting in different Q/K/V matrices. As we did before, we multiply X by the WQ/WK/WV matrices to produce Q/K/V matrices.

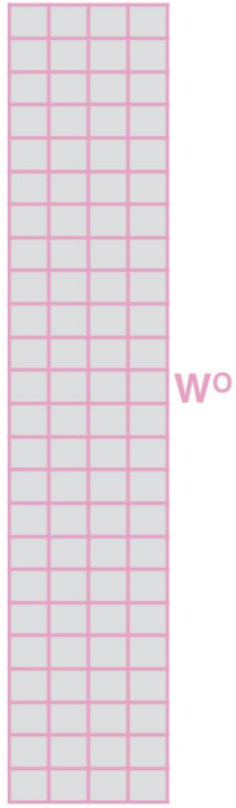
*Image: http://jalammar.github.io/illustrated-transformer/

Multi Head Attention : Illustration

1) Concatenate all the attention heads

2) Multiply with a weight matrix W^O that was trained jointly with the model

X



3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

=

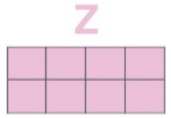


*Image : http://jalammar.github.io/illustrated-transformer/

Multi Head Attention Toy Implementation Overview

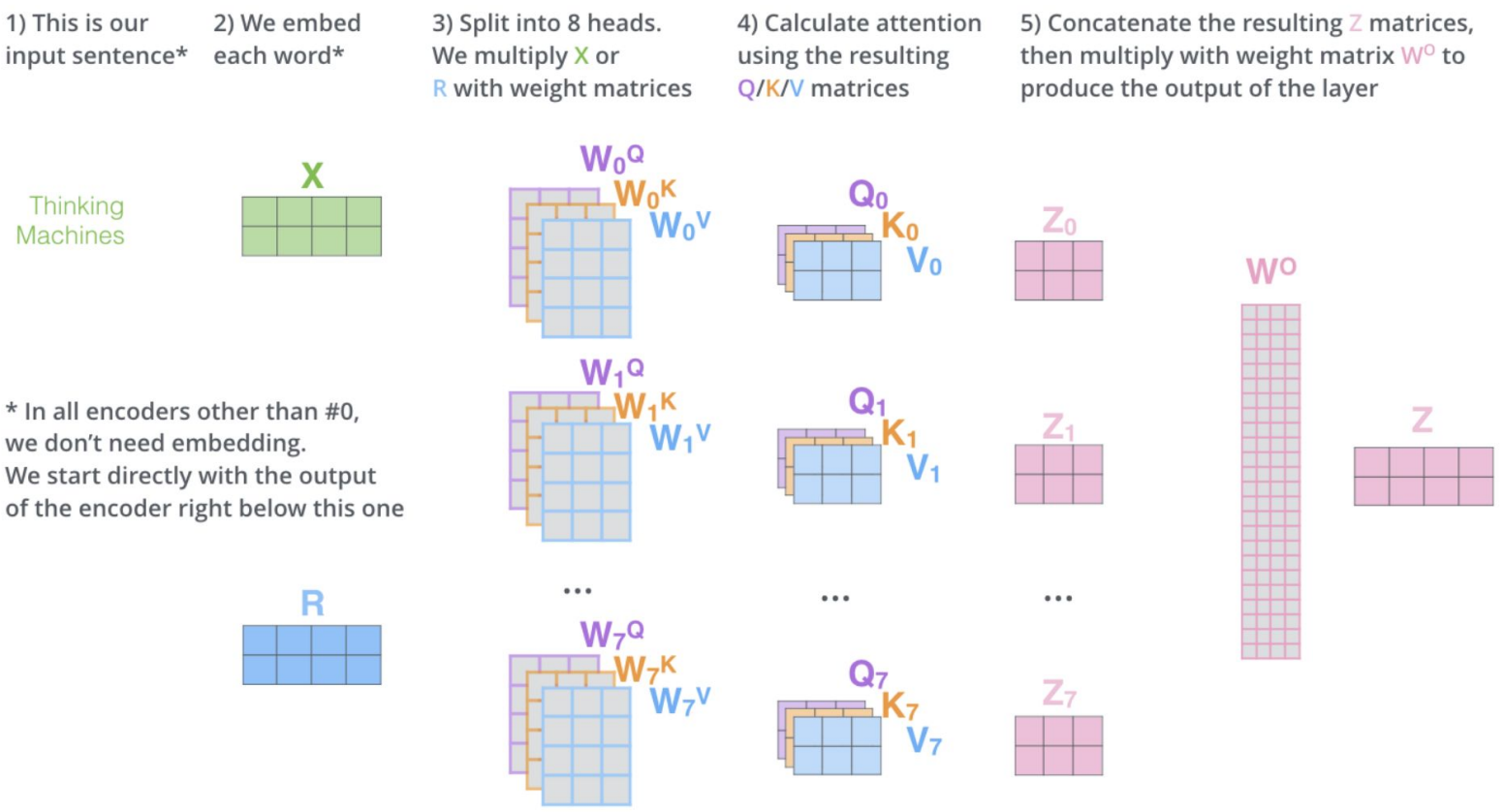
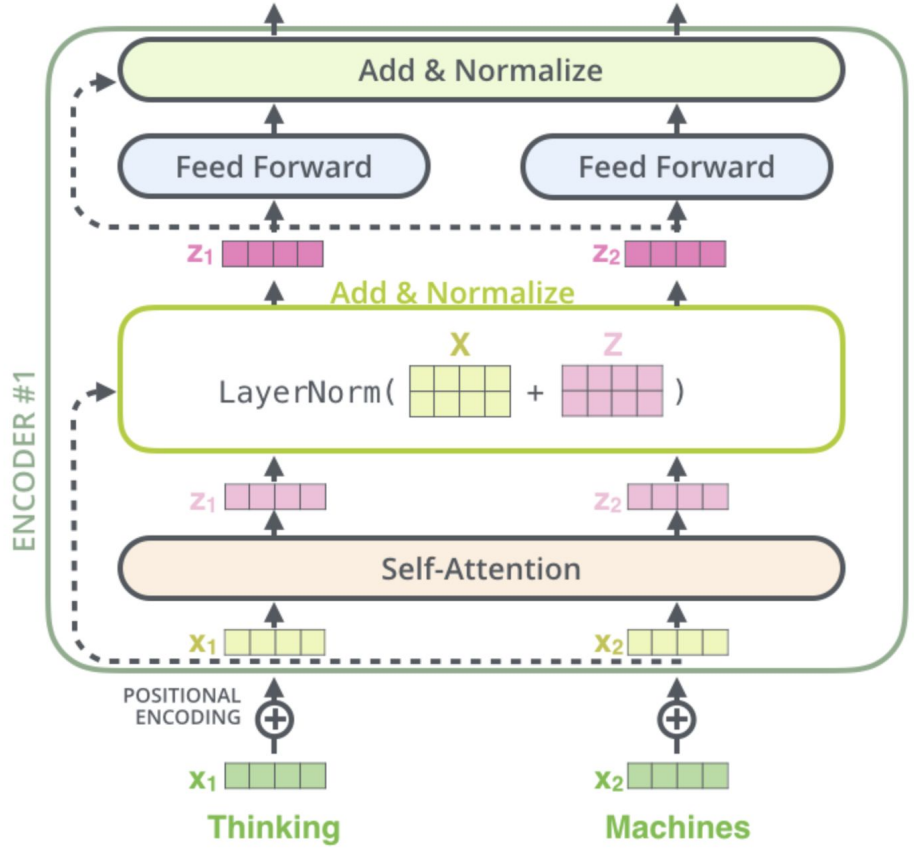


Image : http://jalammar.github.io/illustrated-transformer/

The Residuals /Skip connetion

- Each encoder has a residual connection around it and is followed by a layer-normalization step.



The Decoder Side

- Output of **the top encoder is 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.

- First attention layer in a decoder block pays attention to all (past) inputs to the decoder, but the second attention layer uses the output of the encoder, thus can access the whole input sentence to best predict the current word.

Masked Multi Head Attention

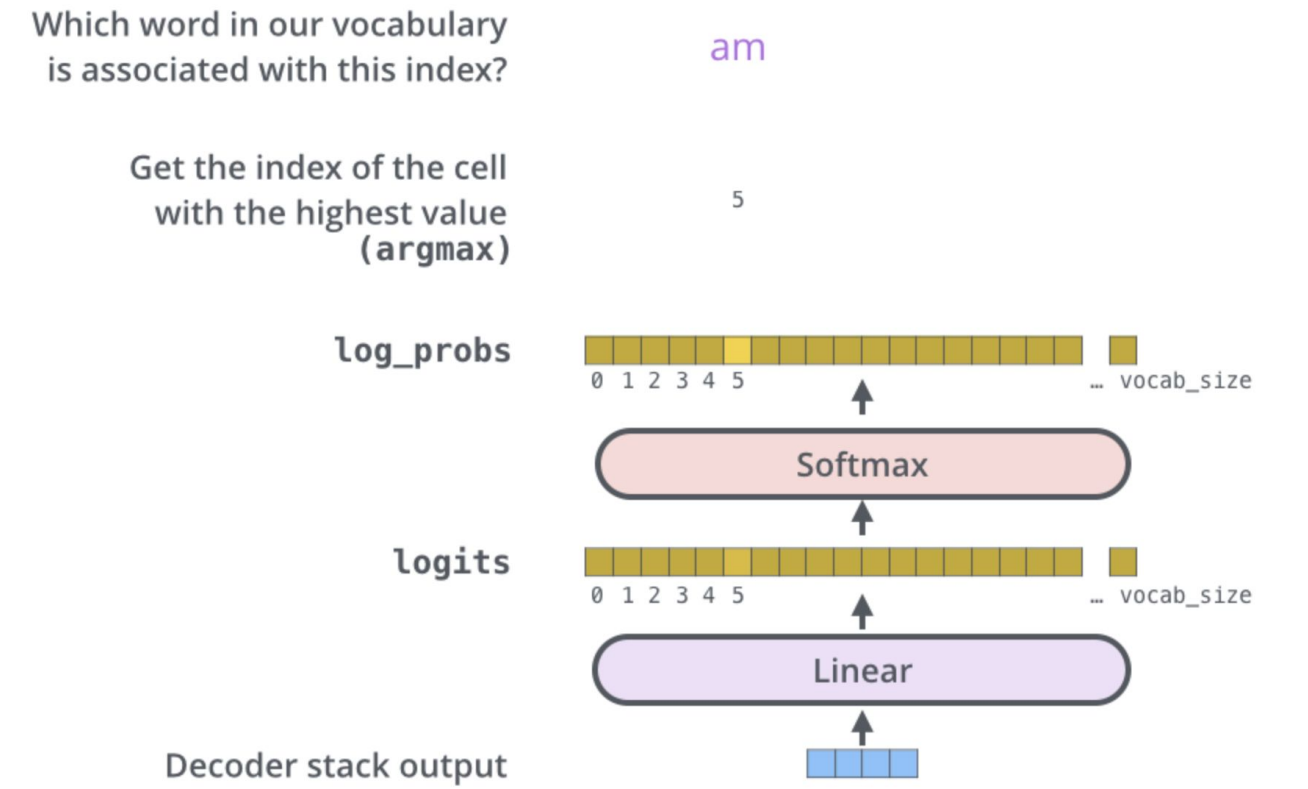
- The self-attention layer is only allowed to attend to earlier positions in the output sequence. This is done by masking future positions (setting them to -inf) before the softmax step in the self-attention calculation.

- Encoder-Decoder Attention" layer works just like multiheaded self-attention, except it creates its Queries matrix from the layer below it and takes the Keys and Values matrix from the output of the encoder stack.

Final Linear and Softmax Layer

- Decoder stack outputs a vector of floats, turning that float into a word is the job of the final Linear layer which is followed by a Softmax Layer.
- Linear layer is a simple fully connected neural network that projects the vector produced by the stack of decoders, into a much, much larger vector called a logits vector.
- Let's assume that our model knows 10,000 unique English words (our model's "output vocabulary") that it's learned from its training dataset. This would make the logits vector 10,000 cells wide – each cell corresponding to the score of a unique word. That is how we interpret the output of the model followed by the Linear layer
- The softmax layer then turns those scores into probabilities (all positive, all add up to 1.0). The cell with the highest probability is chosen, and the word associated with it is produced as the output for this time step.

Final Linear and Softmax Layer : Illustration



This figure starts from the bottom with the vector produced as the output of the decoder stack. It is then turned into an output word.

Transformer Essential Libraries: Hugging Face

Hugging Face

- Hugging Face is an AI community and Machine Learning platform created in 2016 by Julien Chaumond, Clément Delangue, and Thomas Wolf.
- It was initial organization to provide packages for Transformer.
- Now over 20,000 pre-trained models based on the state-of-the-art transformer architecture are available.
- Hugging face transformer library provide a single API through which any Transformer model can be loaded, trained, and saved
- Next we will discuss few of the provided Hugging Face methods/ packages

Tokenizer (convert the text inputs into numbers)

- Splitting the input into words, subwords, or symbols.
- Mapping each token to an integer
- Adding additional inputs that may be useful to the model

Hugging Face Transformer library

- Transformers library provides the functionality to create and use those shared models.
- The Model Hub contains thousands of pretrained models that anyone can download and use and contribute too.
- The most basic object in the Hugging face Transformers library is the **pipeline()** function. It connects a model with its necessary preprocessing and postprocessing steps, allowing us to directly input any text and get an intelligible answer.

Available Hugging Face Pipelines

- ✔ feature-extraction (get the vector representation of a text)
- ✔ fill-mask
- ✔ ner (named entity recognition)
- ✔ question-answering
- ✔ sentiment-analysis
- ✔ summarization
- ✔ text-generation
- ✔ translation
- ✔ zero-shot-classification

Transformer Pretrained Models

- GPT : The first pretrained Transformer model, used for fine-tuning on various NLP tasks and obtained state-of-the-art results
- BERT: Another large pretrained model, this one designed to produce better summaries of sentences.
- GPT-2 : An improved (and bigger) version of GPT
- DistilBERT: A distilled version of BERT that is 60% faster, 40% lighter in memory.
- BART and T5: Two large pretrained models using the same architecture as the original Transformer model
- GPT-3, an even bigger version of GPT-2 that is able to perform well on a variety of tasks without the need for fine-tuning (called *zero-shot learning*)

Transformer Encoder Models

- At each stage, the attention layers can access all the words in the initial sentence. These models are often characterized as having “bi-directional”.
- Encoder models are best suited for tasks requiring an understanding of the full sentence, such as sentence classification, named entity recognition (and more generally word classification), and extractive question answering.
- Representatives of this family of models include:
  - ✔ ALBERT
  - ✔ BERT
  - ✔ DistilBERT
  - ✔ ELECTRA
  - ✔ RoBERTa

Transformer Decoder Model

- Decoder models use only the decoder of a Transformer model.
- for a given word the attention layers can only access the words positioned before it in the sentence.
- The pretraining of decoder models usually revolves around predicting the next word in the sentence.
- best suited for tasks involving text generation.
- Representatives of this family of models include:
  - ✔ CTRL
  - ✔ GPT
  - ✔ GPT-2
  - ✔ Transformer XL

Encoder-decoder aka sequence-to-sequence

- At each stage, the attention layers of the encoder can access all the words in the initial sentence, whereas the attention layers of the decoder can only access the words positioned before a given word in the input.
- pretraining of these models can be done using the objectives of encoder or decoder models, but usually involves something a bit more complex
- best suited for tasks revolving around generating new sentences depending on a given input, such as summarization, translation, or generative question answering.
- Representatives of this family of models include:
  - ✔ BART
  - ✔ mBART
  - ✔ Marian
  - ✔ T5

Bias of Pretrained models

- Pretraining models on extensive datasets often involves scraping diverse content from the internet, encompassing both high-quality and potentially problematic information.

- The initial model you deploy may inadvertently produce biased content, such as sexism, racism, or homophobia. Despite fine-tuning on your specific data, the inherent bias present in the pretrained model may persist and requires careful consideration.

Tasks Example

- Predicting the next word in a sentence based on the context of the preceding n words.
  - ✔ **causal language modelling :** where the output is dependent on past and present inputs but not future ones.
  - ✔ **Masked Language Modeling:** In this task, the model predicts a masked word within a sentence.

Challenges & concerns in Model Training

- **Data Requirements** : Training a model, especially a large one, necessitates a substantial amount of data.
- **Resource Intensiveness**: The process is costly in terms of both time and compute resources.
- **Environmental Impact**: Large-scale model training contributes to environmental concerns, highlighting the need for sustainable practices.