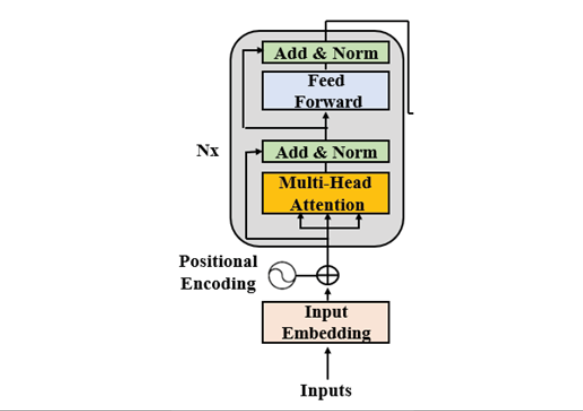
**Book from Newman library: Transformers for NLP**

**1/ The encoder stack:**



* The original encoder layer structure remains for all N=6 layers of the Transformer model. Each layer contains 2 main sub-layers: a multi-headed attention mechanism and a fully connected position-wise feedforward network
* The residual connection surrounds each main sub-layer in the Transformer model. These connections transport the unprocessed input x of the sub-layer to a layer normalization function. This way, we are certain that key information such as positional encoding is not lost on the way.

LayerNormalization (x + Sublayer(x))

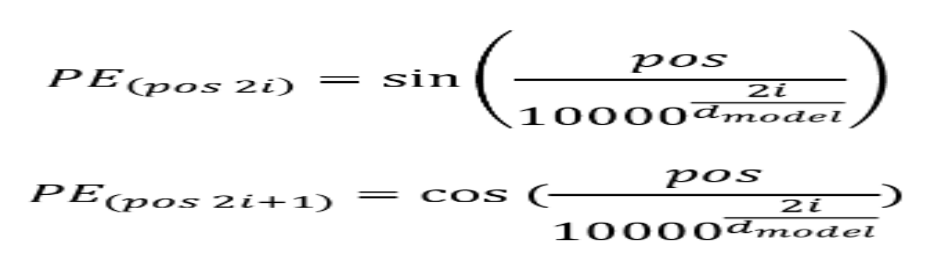
* The multihead attention mechanisms perform the same function from layer 1 to 6. However, they do not perform the same tasks. Each layer learns from the previous layer and explore different ways of associating the tokens in the sequence

**1a/ Input Embedding:**

* The input embedding sub-layer converts the input tokens to vectors of dim d=512. A tokenizer will transform a sentence into token. A tokenizer will generally provide an integer representation.
* A big chunk of info is missing because no additional vector or info indicate a word’s position in a sequence. => Positional Encoding

**1b/ Positional Encoding:**

* We can’t create independent positional vectors that would have a high cost on the training speed of the Transformer and make attention sub-layer very complex to work with. The idea is to add a positional encoding value to the input embedding instead of having additional vectors to describe a position of a token in a sequence

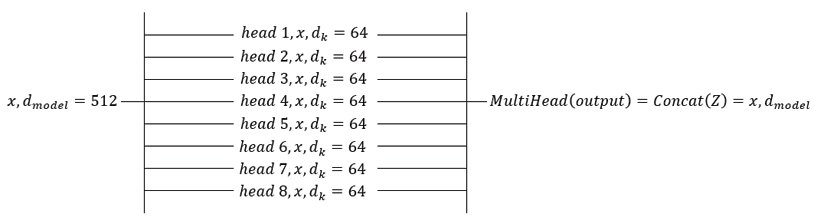


**1c/ Sub-layer 1: Multi-head Attention:**

* The multi-head attention sub-layer contains 8 heads and is followed by post-layer normalization which will add residual connections to the output of the sub-layer and normalize it
* The input of the multi-attention sublayer of the first layer of the encoder stack is a vector that contain the embedding and the positional encoding of EACH WORD. The dim vector of each word has d=512

+ Here we could run a huge calculation by training the model using d\_model = 512 dimension as they are now.

+ We can dun 8 heads in parallel to speed up the training and obtain 8 different representation subspaces of how each word relates to another

- The output of the multi-head will be: Z = (z*0*, z*1*, z*2*, z*3*, z*4*, z*5*, z*6*, z*7*,)

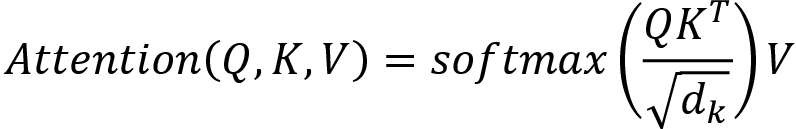
- And then concatenated: MultiHead(output) = Concat(z*0*, z*1*, z*2*, z*3*, z*4*, z*5*, z*6*, z*7*,) = x, d*model­­*

- Inside each head of the attention mechanism, each word vector has 3 representation

+ A query vector Q that has dim of 64, which is activated and trained when a word vector seeks all the key-value pairs of the other word vectors, in cluding itself in self attention

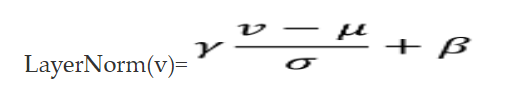
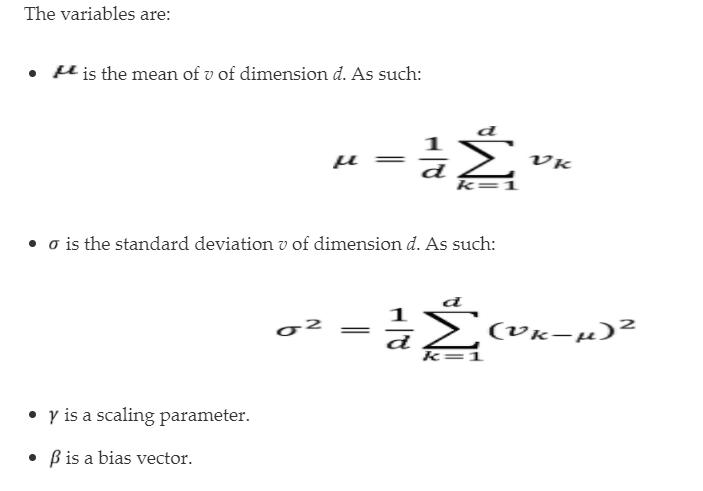
+ A key vector K that has dim = 64, which will be trained to provide an attention value

+ A value vector V that has dim = 64, which will be trained to provide another attention value



Self-attention mechanism: <https://colab.research.google.com/drive/1rPk3ohrmVclqhH7uQ7qys4oznDdAhpzF?authuser=1>

**1d/ Post-layer normalization:**

* Post-LN contains an add function and a layer noromalization process. The goal of the residual connections is to make sure critical information is not lost.
* There are many layer normalization methods exist but it is defined as: 
* 

**1e/ Sub-layer 2: feedforward networks:**

* The input of the FFN is the model = 512 output of the Post layer normalization of the previous sub layer
* Description of FFN

+ The FFN in the encoder and decoder are fully connected

+ The FFN is position-wise network. Each position is processed separately and in an identical way

+ The FFN contain 2 layers and applies a ReLU activation function

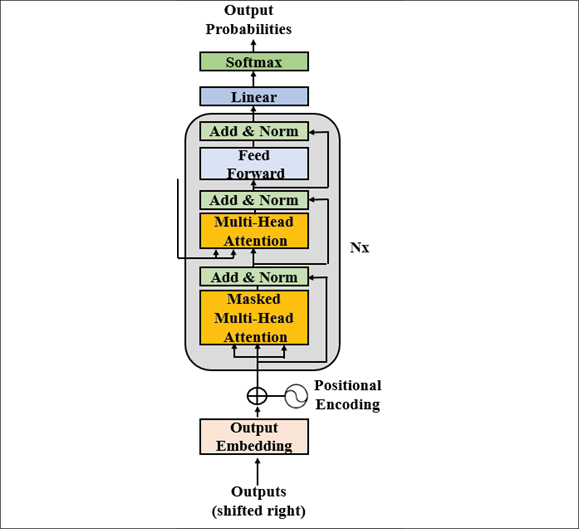
+ The input and output of the FFN layers is d\_model = 512, but the inner layer is larger with d\_ff = 2048

+ The FFN can be viewed as performing 2 kernel size 1 convolution

* We can describe an optimized and standardized FFN as:

FFN(x) = max(0, xW1 + b1)W2 =b2

**2/ Decoder Stack:**



* The structure of the decoder layer remains the same as the encodr for all N = 6 layers of the Transformer model.
* Each layers contain 3 sublayers: a multi-headed masked attention mechanism, a multiheaded attention mechanism, and a fully connected position-wise feedforward network
* Note: the masked multi-head attention mechanism. In this sub-layer output, at a given position in the diagram, the following words are masked so that the Transformer bases its assumption on its inferences without seeing the rest of the sequence. That way, in this model, it can’t see future parts of the sequence.

**2a/ Output embedding and positional encoding:**

* The structure for embedding and positional encoder are the same for decoder compared to the encoder

**2b/ The attention layers:**

* The Transformer is an auto-regressive model. It uses the previous output sequences as an additional input. The multi-head attention layers of the decoder use the same process as the encoder.
* However, the mased multi-head attention sub-layer 1 only lets attention apply to the positions up to and including the current position. The future words are hidden from the Transformer and this forces it to learn how to predict.
* The post layer normalization follows the masked multi-head attention sub-layer 1 as the encoder.
* The multi-head attention sub-layer 2 also only attends to the positions up to the current position the Transformer is predicting to avoid seeing the sequence it must predict.
* The multi-head attention sub-layer 2 draws information from the encoder by taking encoder (K, V) into account during the dot-product attention operations. This sub-layer also draws information from the masked multi-head attention sub-layer 1 (masked attention) by also taking sub-layer 1(Q) into account during the dot-product attention operations. The decoder thus uses the trained information of the encoder. We can define the input of the self-attention multi-head sub-layer of a decoder as:

Input\_Attention=(Output\_decoder\_sub\_layer-1(Q), Output\_encoder\_layer(K,V))

* A post-layer normalization process follows the masked multi-head attention sub-layer 1 as in the encoder.
* The Transformer then goes to the FFN sub-layer, followed by a Post-LN and the linear layer.

**2c/ The FFN sublayer, post layer normalization and linear layer.**

The FFN sub-layer has the same structure as the FFN of the encoder stack. The Post-LN of the FFN works as the layer normalization of the encoder stack.

The Transformer produces an output sequence of only one element at a time:

Output sequence= (y1, y2, … yn)

The linear layer produces an output sequence with a linear function that varies per model but relies on the standard method:

y = w\*x + b

x and b are learned parameters.

The linear layer will thus produce the next probable elements of a sequence that a softmax function will convert into a probable element.

The decoder layer as the encoder layer will then go from layer l to layer l+1 up to the top layer of the N=6-layer transformer stack.