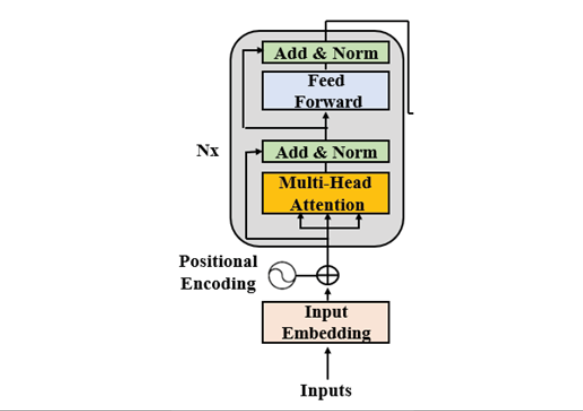
**Book from Newman library: Transformers for NLP**

**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))

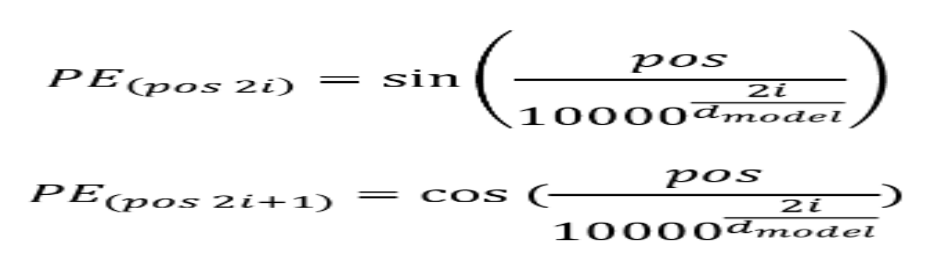
* The multihead attention mechanisms perform the same function from layer 1 to 6. However, they do not perform the same tasks. Each layer learns from the previous layer and explore different ways of associating the tokens in the sequence

**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

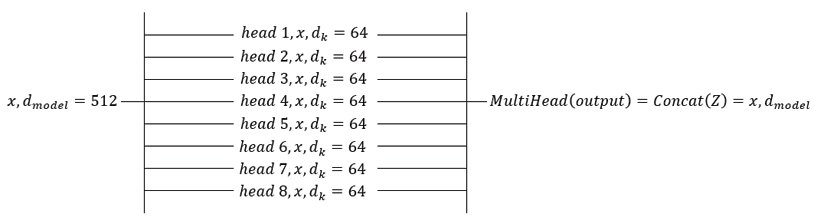


**1c/ Sub-layer 1: Multi-head Attention:**

* 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

