### VARIOUS STAGES FOR PREPROCESSING:

Tokenization

Tokenization is, generally, an early step in the NLP process, a step which splits longer strings of text into smaller pieces, or tokens. Larger chunks of text can be tokenized into sentences, sentences can be tokenized into words, etc. Further processing is generally performed after a piece of text has been appropriately tokenized.

Normalization

Before further processing, text needs to be normalized. Normalization generally refers to a series of related tasks meant to put all text on a level playing field: converting all text to the same case (upper or lower), removing punctuation, expanding contractions, converting numbers to their word equivalents, and so on. Normalization puts all words on equal footing and allows processing to proceed uniformly.

Stemming

Stemming is the process of eliminating affixes (suffixed, prefixes, infixes, circumfixes) from a word in order to obtain a word stem.

running → run

Lemmatization

Lemmatization is related to stemming, differing in that lemmatization is able to capture canonical forms based on a word's lemma.

For example, stemming the word "better" would fail to return its citation form (another word for lemma); however, lemmatization would result in the following:

better → good

It should be easy to see why the implementation of a stemmer would be the less difficult feat of the two.

Coreference resolution (anaphora resolution)

Pronouns and other referring expressions should be connected to the right individuals. Coreference resolution finds the mentions in a text that refer to the same real-world entity. For example, in the sentence, “Andrew said he would buy a car” the pronoun “he” refers to the same person, namely to “Andrew”.

Collocation extraction

Collocations are word combinations occurring together more often than would be expected by chance. Collocation examples are “break the rules,” “free time,” “draw a conclusion,” “keep in mind,” “get ready,” and so on.

### WHAT ARE LANGUAGE MODELS:

Statistical Language Modeling

Statistical Language Modeling is the process of building a statistical language model which is meant to provide an estimate of a natural language. For a sequence of input words, the model would assign a probability to the entire sequence, which contributes to the estimated likelihood of various possible sequences. This can be especially useful for NLP applications which generate text.

### THINGS BEFORE GETTING TO BERT:

Recurrent neural networks:

Words are inputs to input and hidden layers. All layers are initialized with same bias and weights. Unroll and backpropagate to correct the weights.

Working example: <https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks/?utm_source=blog&utm_medium=understanding-transformers-nlp-state-of-the-art-models>

For any NLP task such as machine translation, summarizing, POS tagging, speech recognition etc., we have an input sequence of words and we expect a shorter or similar output sequence.

Hence, we use sequence to sequence recurrent neural networks built using encoder and decoder LSTMs.

A potential issue with this encoder-decoder approach is that a neural network needs to be able to compress all the necessary information of a source sentence into a fixed-length vector. This may make it difficult for the neural network to cope with long sentences. The performance of a basic encoder-decoder deteriorates rapidly as the length of an input sentence increases.

So how do we overcome this problem of long sequences? This is where the concept of **attention** mechanism comes into the picture. It aims to predict a word by looking at a few specific parts of the sequence only, rather than the entire sequence.

Source sequence: “Which sport do you like the most?

Target sequence: “I love cricket”

‘You’ in source is responsible for ‘I’ in target. ‘Like’ in source is responsible for ‘love’ in target. So, instead of looking at all the words in the source sequence, we can increase the importance of specific parts of the source sequence that result in the target sequence.

To get an idea of how attention considers the input sequence and output sequence: Attentions attempts to calculate something called as alignment score eij = score(si,hj) [the decoder outputs the hidden state (si) for every time step i in the target sequence, encoder outputs the hidden state (hj) for every time step j in the source sequence]

Calculating attention comes primarily in three steps. Attention(query Q, Key K, Value V):

First, we take the query and each key and compute the similarity between the two to obtain a weight. Frequently used similarity functions include dot product, splice, detector, etc. The second step is typically to use a softmax function to normalize these weights, and finally to weight these weights in conjunction with the corresponding values and obtain the final Attention.

Each time the model predicts an output word, it only uses parts of an input where the most relevant information is concentrated instead of an entire sentence. The encoder works as usual, but the decoder’s hidden state is computed with a context vector, the previous output and the previous hidden state. Context vectors are computed as a weighted sum of annotations generated by the encoder. We have a separate context vector for each target word.

<https://www.analyticsvidhya.com/blog/2019/06/comprehensive-guide-text-summarization-using-deep-learning-python/?utm_source=blog&utm_medium=understanding-transformers-nlp-state-of-the-art-models>

Despite being so good at what it does, there are certain limitations of seq-2-seq models with attention:

* Dealing with long-range dependencies is still challenging
* The sequential nature of the model architecture prevents parallelization. These challenges are addressed by Google Brain’s **Transformer** concept

Self-attention: sometimes called intra-attention, is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence.

Self-attention vs multi head attention?

Calculating the correlation for each input, target word pair is computationally expensive when there is larger corpus. Transformers accelerate such training.