Transformer notes

* Encoder has self attention & feed forward NN
* self-attention layer – a layer that helps the encoder look at other words in the input sentence as it encodes a specific word
* As the model processes each word (each position in the input sequence), self attention allows it look at other positions in the input sequence for clues that can help lead to a better encoding for this word.
* decoder has self attention, encoder decoder attention and FFNN layers, Encoder decoder attention layer helps the decoder focus on relevant parts of the input sentence
* after calculating score(q x k), we divide by 8(sq root of key dimension). This results in more stable gradients
* Then we pass score to softmax, so that they are all +ve & all add upto 1.
* This softmax score at a position determines how much each word will be expressed at that position.
* Then we multiply score of each word with its value vector. This will give more value to important words, and less value to irrelevant words(by multiplying with tiny numbers like 0.0001).
* We then sum up all the values, and get the output of self attention layer – at this input position.
* We’ll do this using matrices instead of just vectors, for faster processing.
* since we’re dealing with matrices, we can condense steps two through six in one formula to calculate the outputs of the self-attention layer.
* We add positional encoding to word embeddings, so that model learns the position of each word/distance between different words.
* Why multi-headed attention? It expands the model’s ability to focus on different positions. Yes, in the example above, z1 contains a little bit of every other encoding, but it could be dominated by the actual word itself. Having different heads will allow for different representation subspaces, and then let us choose from them.
* In the decoder, 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 creates queries from layer below it, and keys and values from final encoder outputs.
* Linear layer -> 10k units or vocabulary size. each cell corresponding to the score of a unique word.
* We apply softmax on it to turn score into probability, and then choose word with max probability to be output for the given time stamp.
* Word embedding - building a low-dimensional vector representation from corpus of text, which preserves the contextual similarity of words.
* We one-hot encode input words, feed it into hidden layers, and then try to find efficient representations