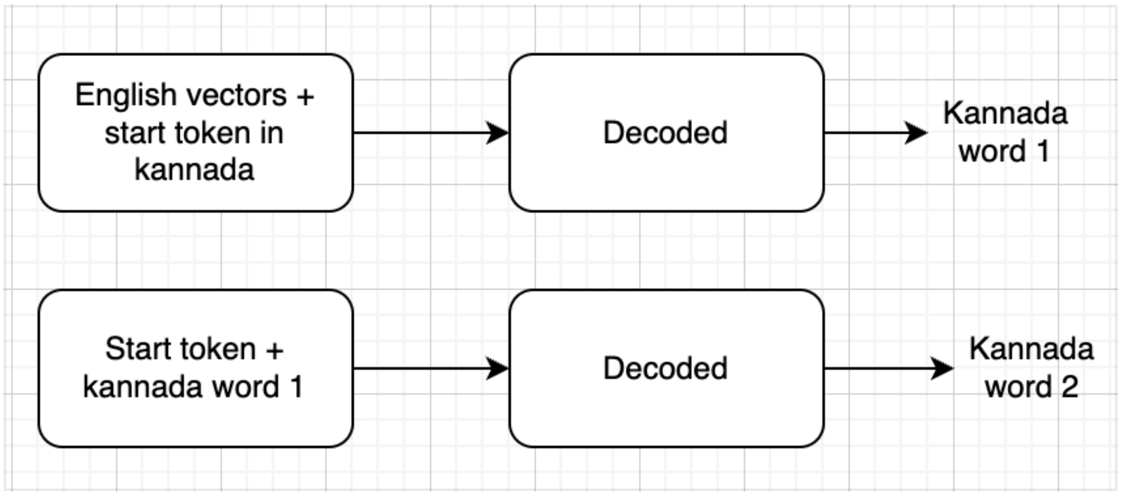
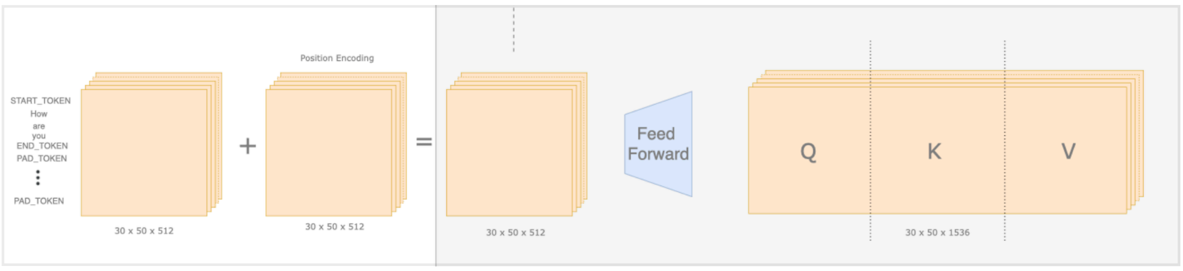
Transformers

Encoder takes English word and generate word vectors, word vector eventually becomes contextually aware, this happens because of multi head attention. Decoder is passed on other language with start and end, along with vectors, the output received is shifted left, given start token we get token 1, given token 1 and start token, we get token 2. Almost like given N gram predict N+1th word.

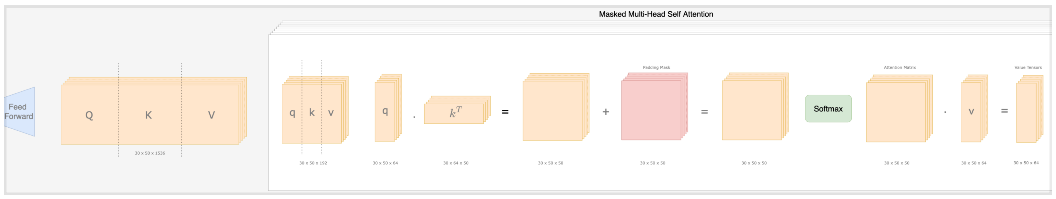
For loss, summation of one hot vector representation of the next word in the translation (true class distribution)
 * probability distribution of the next word translation from the model (predicted class distribution) - cross entropy loss, for every predicted word and sum them up, the loss is propagated back and weights are updated.


English vectors + start token in kannada
Decoded
Kannada word 1
Start token + kannada word 1
Decoded
Kannada word 2

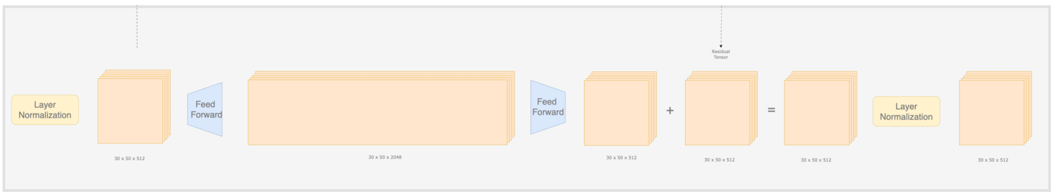
Why batching? Faster training in larger networks, update weights after every batch, this is mini batch gradient descent.



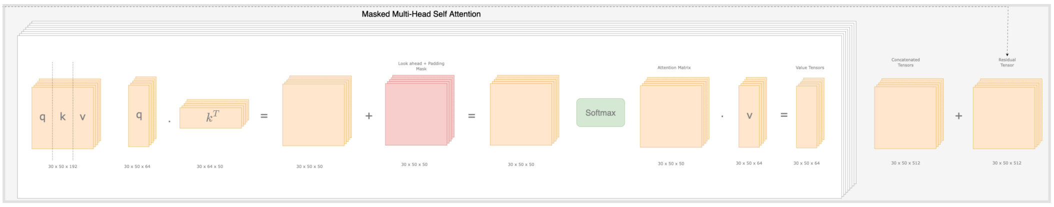
- We take such hyper parameters, batch size = 30, max length per doc = 50, vector size = 512 (each word representation) = 30 * 50 * 512
- Positional encoding are sine and cosine, encoder takes word simulatenously, hence we need to have possitional encodings. Defines order of the words.
- Every word is split up in a Q,K,V vector, are results of feed-forward transformations, mostly useful in multi-head attention. 30 * 50 * (512*3) = 30 * 50 * 1536.



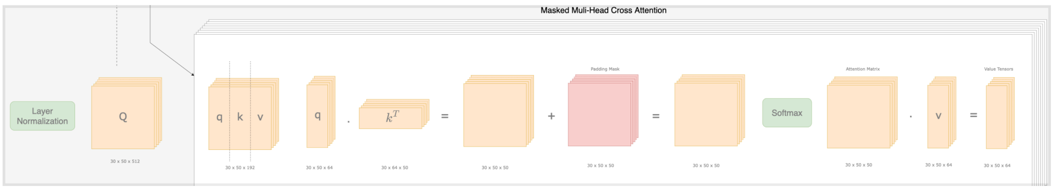
- Multi Head Attention is used to analyse context of each word in the sentence.Self Attention is every word in a sentence is compared to every word in the same sentence. Padding tokens shouldn't be considered, mask them out while learning. 8 parallel processes are going on.
- 1536 broken into 8 pieces, each piece is 64 dimensional vector. 192 dimensional vector representing every word for one head.
- Q x K (transpose) = (30 * 50 * 64) * (30 * 64 * 50) = 30 * 50 * 50, This is essentially Self Attention, because, every word in query is compared with every word in key.
- The out is scaled multiplication values between Q and K, (dividing every value with a constant), square root of key dimension size for one head = 8, which is concatenated with Padding Mask, to prevent padding tokens from propagating values. Using softmax, 1 to pass through, 0 for avoiding. Which is passed on to softmax, to get 0, the number passed on to softmax is extremely low, but not - infinity, because if enitre row is - infinity, then it puts us 0/0 problem.
- We then get an Attention Matrix, which is of dimension, 30 * number of max words * number of max words.
- Then multiply value matrix to get value matrix, which are contextually aware and are of size (30 * 50 * 64), which we get from each head, so 30 * 50 * 512 is a concatenated tensor.
- Sometimes because of how deep the network is there might be vanishing gradient, parameters don't change as gradients vanish, and no learning happens, skip connections help by enabling better propagation in forward direction, and loss in backward propagation. Residual tensor is added to concatenated tensor.
- Layer normalisation is performed, activation during forward, gradients during backwards aren't abnormally high, normalisation is subtracting values by mean and dividing by standard deviation, normalisation happens across feature layer (512), each value of the tensor is normalised, we add epsilon, in order to prevent 0/0, we can use gamma and beta to track learning across different.



- The output tensor, is passed through feed forward layer to capture additional context, for every word we will have a tensor of 512 dimension which is contextually aware.

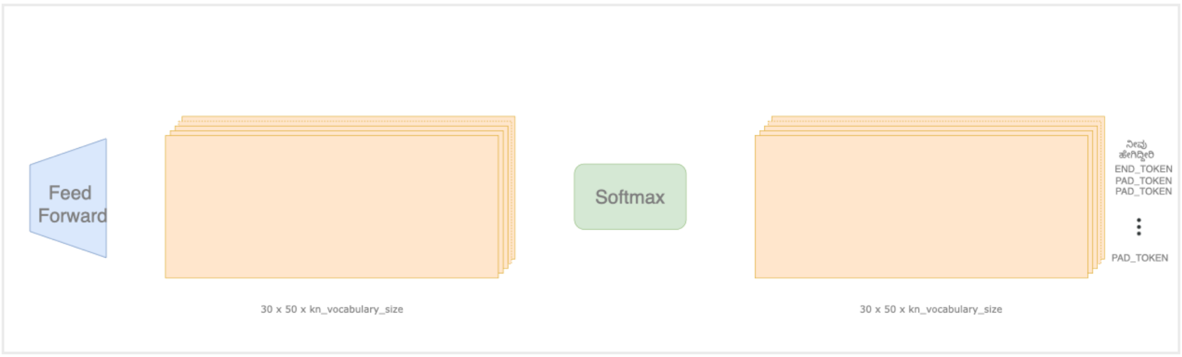


- In the decoder architecture, essentially the same thing is performed like encoder until obtaining an attention matrix after multiplying Query and Key tensors, but we add a lookahead mask along with padding mask, this prevents it from accessing future words, and during training we apply mask so it avoids looking at future words. Then softmax is applied to see how much attention every word must be given, which is multiplied with value matrix, to get value tensors which is concatenated from eight other heads.
- Residual tensors and layer normalisation is applied.



- We get a batch, which is a set of query vectors which is passed on to masked multihued cross attention layer, unlike earlier, every word in Kannada sentence (target) is compared with every word in English sentence (source). The target is inside the query tensors.
- The MH cross attention gets a piece of input, which K,V tensors (obtained as an output from feed forward layer before being passed on), so we have Q, K, V. This K, V will add English information (source information) to the Kannada query tensors (target), so we generate appropriate word based on what the English word was.
- The Q, K, V is passed on to Cross Attention, as shown in architecture below, we don't include, look ahead here because every Kannada word should be able to see the English information, this is passed to softmax so the model knows how much attention should each Kannada word pay to each English context, added on to value tensor, which is concatenated (every Kannada word has some English context) which is added with residual tensors (extra propagated information).

- The Encoder and Decoder can be repeated N times



The tensors is passed to feed forward layer to expand it to Kannda vocabulary size, and this vector is passed on to softmax, we get a probability distribution and get most probable word as prediction, it is going to be compared to the labels. This comparison would generate cross entropy loss and it is back propagated.

Link: https://www.youtube.com/watch?v=Nw_PJdmydZY&t=22s