1 Introduction

We began our journey with word vectors. We looked at vectors on 2D plane in using python to understand analogies and importance of word vectors. Then we moved to RNNS (Recurrent Neural Networks). This architecture had vanishing gradients problem (gradients diminish exponentially as they are propagated back through time) which prevented it from learning long-term dependences in sequences. This issue was partially resolved by LSTMS (Long Short-Term Memory). We used LSTMs for neural machine translation. This was an interesting task. However, it was slow due to its sequential processing of input embeddings. We then moved to transformers to utilize the GPUs parallel processing capabilities.

2 Tasks

2.1 Week 1

2.2 Rooshan Khan: Attention Method in bert.py

In this method we had to implement Attention method from class BertSelfAttention. The attention mechanism is given by:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{1}$$

where Q, K, and V represent the query, key, and value matrices, respectively, and d_k is the dimension of the key vectors.

I used method *torch.matmul* to multiply Q and transpose of K. I multiplied the result with *attention_mask* to apply the mask. The dimensions of attention_mask are $[bs, 1, 1, seq\_len]$. The attention mask distinguishes between non-padding tokens and padding tokens. The non-padding tokens have a value of 0 while padding tokens have a value of a large negative number. The dimensions of key_layer,query_layer and value_layer are $[bs, num\_attention\_heads, seq\_len, attention\_head\_size]$

Now I will tell how I concatenated all heads. When we transpose the tensor we change the shape of the tensor from $[bs, num\_attention\_heads, seq\_len, attention\_head\_size]$ to $[bs, seq\_len, num\_attention\_heads, attention\_head\_size]$. This enables us to reshape the tensor to $[bs, seq\_len, num\_attention\_heads * attention\_head\_size]$. After applying the transpose method the data sequence does not follow a contiguous order so we need to use **contiguous** method before using the **view** method.