Task 3

**The problems with RNNs, LSTMs and GRUs:**

* Inability to model long term dependencies in sequential data
* Computationally expensive as there is no scope for parallelization

**Transformer model architecture:**

The Transformer architecture uses stacked self-attention and fully connected layers for both the encoder and decoder, as shown in the left and right halves of Figure 1.

1. **Encoder and Decoder stacks**

**Encoder:** composed of a stack of N=6 identical layers. Each layer is divided into two sublayers - a multi head self-attention layer and a position-wise fully connected feed forward neural network. A residual connection is added to the output of each sublayer and the result is normalized, to address the problem of vanishing gradients. The output of each sublayer is LayerNorm(x + Sublayer(x)), where Sublayer(x) is the function implemented by the sublayer.

**Decoder:** also composed of stack of N=6 identical layers. In addition to the two

sub-layers in each encoder layer, the decoder inserts a third sub-layer, a multi head attention over the output of the encoder stack which basically helps the decoder to “look at/infer” from the output of the encoder. Like in the encoder, residual connections followed by layer normalization is employed. Here, the self-attention block is modified in a way such that predictions for a word can only depend on previous words (meaning when predicting word i, the model can only use the information from words 1-i (the words before it)). This technique is called masking.

1. **Attention**

An attention function maps a query and a set of key-value pairs to an output. The output is a weighted sum of the values, where the weight for each value is determined by how well the query matches the corresponding key. All of these (query, keys, values, and output) are vectors. The weight is calculated using a compatibility function that measures the similarity between the query and each key.

**2.1 Scaled Dot Product Attention**

The input consists of queries and keys of dimensions *dk* and values of dimension *dv* . Then, we compute the dot product of the queries with the keys, divide by 1/ and applies a softmax to get attention weights, which are then used to compute the weighted sum of values. This is expressed as:



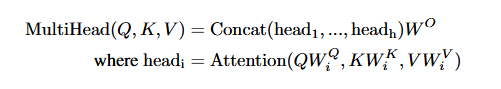
The two main attention mechanisms are additive attention and dot-product (multiplicative) attention. Dot-product attention is similar to additive attention but includes a scaling factor of Additive attention uses a feed-forward network for compatibility, while dot-product attention is faster and more efficient due to optimized matrix multiplication.

For small, both mechanisms perform similarly, but for larger , additive attention outperforms dot-product attention without scaling. This is because larger dot products can cause the softmax function to have very small gradients, which slows down learning. The scaling by helps mitigate this issue.

* 1. **Multi Head Attention**

Multi-head attention processes the queries, keys, and values in parallel. This allows the model to attend to different parts of the input at once. The results from each attention head are then concatenated and projected to produce the final output.

The formula is:



* 1. **Applications of attention in the model**

The Transformer uses multi-head attention in three ways:

* Encoder-Decoder Attention: Queries come from the previous decoder layer, and keys/values come from the encoder output. This allows each position in the decoder to attend to all positions in the input sequence.
* Self-Attention in the Encoder: Queries, keys, and values come from the previous encoder layer, allowing each position in the encoder to attend to all positions in that layer.
* Self-Attention in the Decoder: Each position in the decoder attends to all previous positions, but to maintain the auto-regressive property, we mask future positions (set to −∞) to prevent leftward information flow.

This setup enables the model to effectively capture dependencies across the input output sequences.

1. **Position wise feed-forward networks**

Each layer in the encoder and decoder has a feed-forward network (FFN), which applies two linear transformations with a ReLU activation in between, to each position independently. The function is:



Although the transformations are the same across positions, the parameters (weights and biases) differ between layers. This layer is called the multi-layer perceptron and is placed between the attention blocks. They offer an extra capacity to store facts.

1. **Embeddings and softmax**

Learned embeddings are used to convert the input tokens to vectors and output vectors to tokens (dimensions *dmodel*). The usual learned linear transformation followed by a softmax to convert decoder output to next-token probabilities.

In this model, instead of using a separate matrix for the embedding layer and the output layer, the same matrix *W* is shared between the input embedding layer and the output layer. Also, the embeddings are scaled by to prevent numerical instability.

1. **Positional encoding**

To make use of the order of the sequence, information about the relative/absolute position of the tokens is added using positional encodings (dimensions *dmodel*). They are added to the input embeddings at the bottom of the encoder and decoder stacks.

**Why self-attention:**

* Reduces computational complexity per layer especially for large sequences
* Employs a good amount of parallelization and so training becomes much faster
* Self-attention is highly effective at capturing long-range dependencies in sequences because all positions are connected with a constant number of sequential operations. It has a constant path length, which makes learning long-range dependencies easier.

Recurrent networks and convolutions usually have longer path lengths due to their sequential connections, making them less efficient for capturing long-range dependencies especially as the sequences get large.