# Project 2: Transformer Language Modelling

This project deals with the implementation of language modelling and self-attention mechanism using a "Transformer" encoder.

In Part 1, a transformer encoder was built from scratch using a neural network architecture (linear layers, matmul, softmax) in PyTorch to predict how many times a character in a string occurred before itself for each position in the string (maxing out at 2). A multi-head attention architecture was designed having max 4 multi-head attentions, however, only a single head was used for evaluation except for the exploration pieces. With a **single** attention head, **2** transformer layers, **5** epochs, d_model = **128**, d_internal = **100**, and a training run-time of approximately **4.7 minutes**, the transformer encoder was able to predict the character count with an accuracy of **99.68%**.

In Part 2, a transformer language model was implemented to consume a chunk of characters and predict the next character at each position simultaneously. For this, a built-in **TransformerEncoder** and **TransformerEncoderLayer** was used followed by linear/softmax layer to give log probabilities of each next character out of total 26 alphabets and a space character. Using a training chunk length of **20 characters**, **10** epochs, **2** transformer layers, **8** attention heads, d_model=**128**, and a training run-time of around 7.3 minutes, the model was able to achieve a perplexity value of **5.72** on the dev dataset.

**Exploration 1: Attention Masks**

For the attention masks in Part 1, the goal was to observe if a character in any position of the string was attending to its occurrence in any previous position. In Figure 1 (Appendix), three attention masks are provided for a character string which was run through a three-layered transformer architecture used in Part 1.

In the attention mask for the first transformer layer (left), it is evident that each character (row) is primarily attending to its previous characters while giving a minute attention to characters ahead of itself in the string. Moreover, it is giving higher attention to a similar character while a portion of attention is also given to other previous characters. This distinction is obvious in the attention map from second transformer layer (middle) where majority of the attention is being given to similar occurrences of the character previously while minute attention is still being given to other characters. In the third transformer layer (right), it is evident that almost all the attention is being given to previous occurrences of the same character in the string with all the other attention getting close to zero (dark region).

A discrepancy, however, still exists in the attention maps for the first 2-3 characters which may be explained by the fact that first characters don't see any similar character previously. So, they are bound to get divided attention to some other characters in the sequence, and still, it is evident that they are only giving attention to similar characters occurring next to them in the string. As we advance further down the string, this discrepancy is resolved as characters now have seen a bunch of string characters and they give attention to only previous occurrences of similar characters.

From the attention maps of the 3 transformer layers, it is evident that as we introduce more and more transformer layers, the attention gets more refined to what the mode is expected to do in the wake of the downstream task i.e., recognizing contextual relation or language modelling task.

**Exploration 2: Multi-Head Self Attention**

In Part 1, a multi-head self-attention approach was implemented from scratch using neural network architecture (linear layers, softmax, concat) following Alammar blog post. The significant change was to get the output of each attention head, then concatenate them horizontally before matrix multiplication with another matrix (**W**$^0$) having dimensions **dv x d_model** which brought the dimensions back to **seq_len x d_model.**

Using a similar architecture, the accuracy was measured on dev data using different transformer layers and attention heads as show in Table 1 (Appendix). Using 4 attention heads instead of only **1** while using a single transformer layer, the accuracy jumped to **98%** from **89%.** Another comparison was made by using **2** transformer layers and it was found that a single transformer layer having 4 attention heads performed in a similar fashion as an architecture trained using two transformer layers and a single attention head. The max. accuracy was achieved using two transformer layers with 4 attention heads out of all the comparisons which gives support to the idea that transformer architectures having multiple layers and multiple attention heads perform better for language modelling tasks.

**Appendix:**

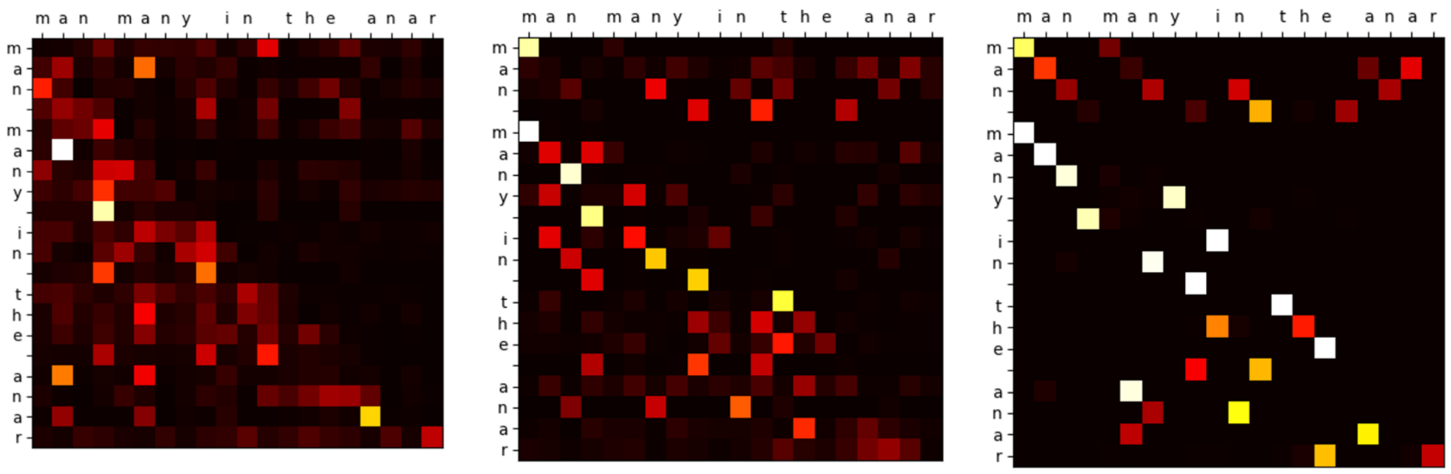

**Figure 1: Attention Masks produced by different transformer layers. Left: First Transformer Layer. Middle: Second Transformer Layer. Right: Third Transformer Layer**

| Transformer Layers | Attention Heads | Epochs | Accuracy |
|---|---|---|---|
| 1 | 1 | 5 | 0.893 |
| 1 | 4 | 5 | **0.989** |
| 2 | 1 | 5 | 0.983 |
| 2 | 4 | 5 | **0.998** |

**Table 1: Performance of different transformer attention heads and layers**