Assignment 10 - Transformer Application in a Sequence-to-Sequence Task

Simone Vaccari 915222

Alessio De Luca 919790

1 Objective

The objective of this assignment is to deeply analyze the Transformer architecture and subsequently apply a Transformer model to tackle a sequence-to-sequence task. In particular, we will evaluate the model's performances in inverting a sequence of numbers of arbitrary length.

2 Introduction

The Transformer architecture was initially introduced to solve natural language processing (NLP) problems, thanks to its ability of dealing with inputs of different size and creating links between word representations. Both these properties come from one of the building blocks of Transformers: the **scaled dot-product self-attention**. The self-attention block takes N inputs x_n of dimension $D \times 1$, and yields N outputs of the same size. From each input vector, a set of **values**, **queries**, and **keys** are computed:

$$\begin{aligned} v_n &= \beta_v + \Omega_v x_n \quad \forall n = 1...N \\ q_n &= \beta_q + \Omega_q x_n \quad \forall n = 1...N \\ k_n &= \beta_k + \Omega_k x_n \quad \forall n = 1...N \end{aligned}$$

where $\beta_v, \Omega_v, \beta_q, \Omega_q, \beta_k, \Omega_k$ are the shared parameters, independent of the number of inputs N.

Starting from the queries and the keys, **attention weights** are computed, as:

$$a[x_m, x_n] = softmax(k_m^T q_n)$$

They represent the attention that input x_n pays to input x_m, and are such that they're all positive and sum to 1. The presence of the dot product and of the softmax function makes this operation non-linear.

The result of the self-attention block is then given by a linear combination of the values with the attention weights:

$$sa[x_n] = \sum_{m=1}^{N} a[x_m, x_n] v_m$$

All these steps can be written together in a more compact form by using the matrix notation. Furthermore to prevent small changes in the inputs to have little effect on the outputs, the dot products are normally scaled by the square root of the dimension D_q of the queries:

$$Sa[X] = V \cdot Softmax\left[\frac{K^T Q}{\sqrt{D_q}}\right]$$

The scaled dot-product self-attention allows a network to attend over a sequence. Often the self-attention mechanism is extended to multiple heads, and in this case we speak of **multi-head self-attention**. For each of the H heads we have different parameters (values, queries, and

keys) of size $\frac{D}{H}$, where D is the size of the input vectors. The heads are concatenated vertically and combined with a final weight matrix:

$$MhSa[X] = \Omega_c \left[Sa_1[X]^T, Sa_2[X]^T, ..., Sa_H[X]^T \right]$$

Since originally the Transformer model was designed for machine translation, it is composed of an **encoder** for interpreting the input sequence in an attention-based representation and a **decoder** for generating the output in an autoregressive manner from the encoded information. The full Transformer architecture is shown in Figure 1. However, for our purposes, given the simplicity of the task, we will only use the encoder part.

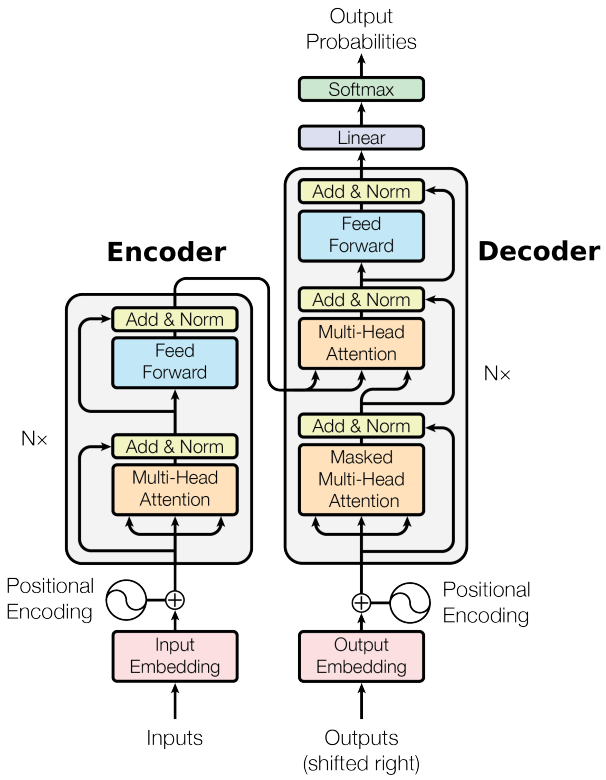

Figure 1: Transformer architecture

A **Sequence-to-Sequence task** represents a task where the input and the output are a sequence, not necessarily of the same length. Popular tasks in this domain include machine translation and summarization. Our challenge consists in, given a sequence of `seq_len` numbers between 0 and `nCategories-1`, reversing the input sequence. Even though this task seems quite straightforward, RNNs can struggle with it due to their incapability of handling long-term dependencies, hence the need of using a Transformer model.

3 Procedure

The work starts by creating a custom dataset defined as "reversed dataset", which is composed by sequences of input data with their associated label, with it being the same sequence but reversed. Two variables are defined: the number of categories (i.e. numbers from 0 to N-1) and the length of the sequences. Here's an example of a sequence of 16 characters with 20 categories:

Input data: tensor ([19, 16, 12, 10, 6, 2, 17, 19, 7, 3, 3, 14, 13, 17, 10, 9])
Labels: tensor ([9, 10, 17, 13, 14, 3, 3, 7, 19, 17, 2, 6, 10, 12, 16, 19])

In total, 50,000 samples are generated for training, 1,000 for validation, and 10,000 for testing. Each set is loaded into the DataLoaders in batches of 128 samples.

Every number in the sequence is represented as a one-hot vector of length `nCategories`, with the value 1 for the class to which the number belongs and 0 for all the others.

The Transformer model is now ready to be trained, validated and tested. Given the simplicity of the task, we use a single encoder block and a single head in the Multi-Head Attention. Each input sequence passes through the Transformer encoder which predicts the output for each input token. The performance of the model is evaluated using standard Cross-Entropy loss and computing mean accuracy on the validation set.

4 Results

We performed various experiments changing every time the length of the training sequences and the number of categories, i.e. figures, that compose such sequences. The results are reported in Table 1 (10 categories) and Table 2 (20 categories).

Sequence length	Validation accuracy	Test accuracy
100	100%	100%
150	97.59%	97.61%
200	85.25%	85.27%
250	63.36%	63.36%
300	41.83%	41.85%
350	29.21%	29.24%
400	24.51%	24.46%
450	11.91%	11.91%
500	10.46%	10.40%

Table 1: Performance of the model on different datasets with 10 categories

Sequence length	Validation accuracy	Test accuracy
100	100%	100%
150	48.07%	48.11%
200	37.78%	37.8%
250	24.44%	24.47%
300	4.96%	5.02%
350	4.99%	4.99%
400	4.98%	5.04%
450	5.02%	5.00%
500	5.10%	5.13%

Table 2: Performance of the model on different datasets with 20 categories

Figure 2 shows the trend of the accuracy on the testing set in both the case of sequences with figures from 0 to 9 and from 0 to 19.

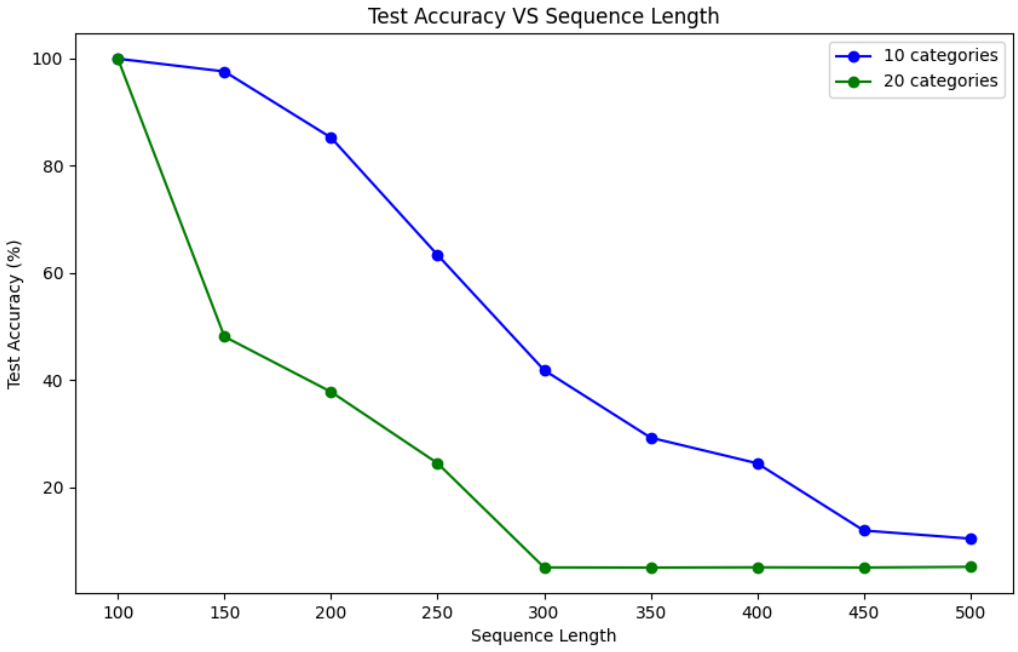

Figure 2: Test accuracy trend with increasing sequence length

As we can easily visualize, the accuracy drastically decreases as the sequence length grows, expressing the difficulty of the model in correctly inverting sequences that are too long or too complex. By comparing the two curves, we can observe that, as expected, given the same sequence length the model that deals with less categories performs better.

We also proceeded in displaying the attention maps of some of our trained Transformers.

In Figure 3, we report the attention maps of two models trained on the dataset with sequences of 100 numbers and 10 categories, and on the dataset with 200 sequence length and 20 categories, respectively.

While the first plot shows that the Transformer model has learnt to pay attention to the token in the symmetrical position (w.r.t. the center of the sequence), the second one suggests that the model attends to certain values that it should not.

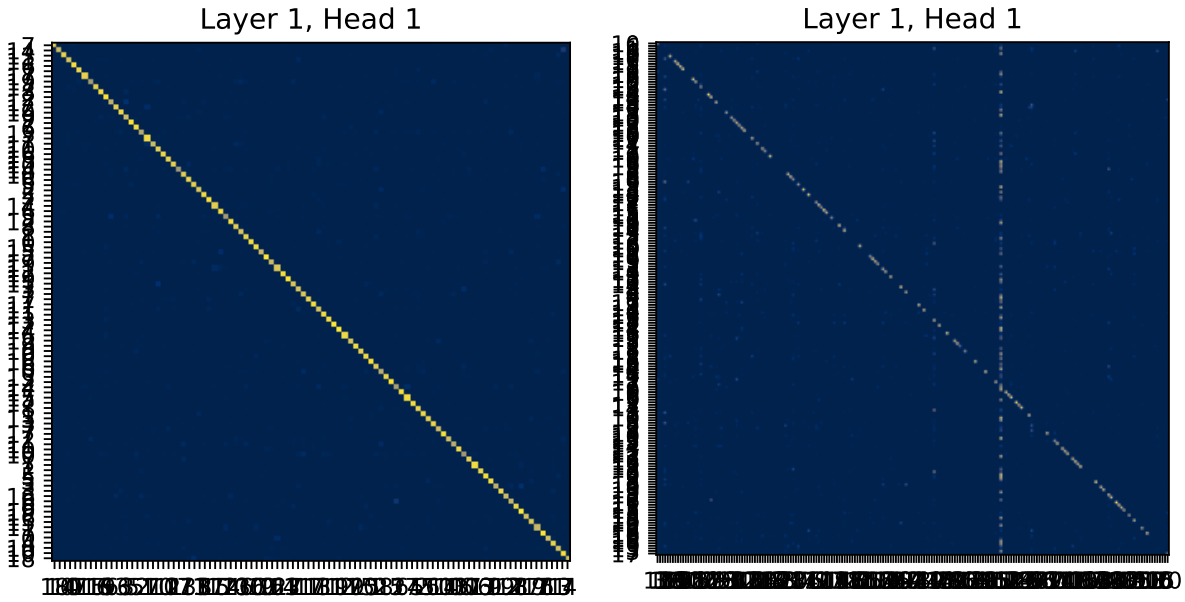

Figure 3: Attention maps of Transformers trained on two different custom datasets

5 Conclusions

This study demonstrates the applicability of Transformer models to sequence-to-sequence tasks, specifically focusing on inverting sequences of numbers. Despite the simplicity of the task, our experiments reveal significant insights into the behavior of Transformers under varying conditions. The use of only the encoder part of the architecture, due to the straightforward nature of the task, has proven effective yet highlights certain limitations.

Our findings indicate that the Transformer model's performance decreases as the sequence length increases, reflecting the inherent difficulty in managing long-term dependencies, even for an architecture designed to handle such challenges. This drop in accuracy is more pronounced when the number of categories within the sequences increases.

Overall, our experiments emphasize the importance of considering both sequence length and category complexity during model design and training. In order to address these challenges effectively, architectural adjustments or advanced training techniques may be necessary.